
Amazon Simple Storage Service

Developer Guide

API Version 2006-03-01



Amazon Simple Storage Service: Developer Guide

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

The AWS Documentation website is getting a new look!

Try it now and let us know what you think. [Switch to the new look >>](#)

You can return to the original look by selecting English in the language selector above.

Table of Contents

What is Amazon S3?	1
How Do I...?	1
Introduction	2
Overview of Amazon S3 and This Guide	2
Advantages of Using Amazon S3	2
Amazon S3 Concepts	3
Buckets	3
Objects	3
Keys	3
Regions	4
Amazon S3 Data Consistency Model	4
Amazon S3 Features	6
Storage Classes	6
Bucket Policies	6
AWS Identity and Access Management	7
Access Control Lists	7
Versioning	7
Operations	8
Amazon S3 Application Programming Interfaces (API)	8
The REST Interface	8
The SOAP Interface	8
Paying for Amazon S3	9
Related Services	9
Making Requests	10
About Access Keys	10
AWS Account Access Keys	10
IAM User Access Keys	10
Temporary Security Credentials	11
Request Endpoints	11
Making Requests over IPv6	12
Getting Started with IPv6	12
Using IPv6 Addresses in IAM Policies	13
Testing IP Address Compatibility	14
Using Dual-Stack Endpoints	14
Making Requests Using the AWS SDKs	19
Using AWS Account or IAM User Credentials	19
Using IAM User Temporary Credentials	26
Using Federated User Temporary Credentials	34
Making Requests Using the REST API	44
Dual-Stack Endpoints (REST API)	45
Virtual Hosting of Buckets	45
Request Redirection and the REST API	50
Buckets	53
Creating a Bucket	53
About Permissions	55
Managing Public Access to Buckets	55
Accessing a Bucket	55
Bucket Configuration Options	56
Restrictions and Limitations	58
Rules for Naming	58
Examples of Creating a Bucket	59
Using the Amazon S3 Console	60
Using the AWS SDK for Java	60
Using the AWS SDK for .NET	61

Using the AWS SDK for Ruby Version 3	62
Using Other AWS SDKs	62
Deleting or Emptying a Bucket	62
Delete a Bucket	63
Empty a Bucket	65
Default Encryption for a Bucket	66
How to Set Up Amazon S3 Default Bucket Encryption	67
Moving to Default Encryption from Using Bucket Policies for Encryption Enforcement	68
Using Default Encryption with Replication	68
Monitoring Default Encryption with CloudTrail and CloudWatch	69
More Info	69
Bucket Website Configuration	69
Using the AWS Management Console	69
Using the AWS SDK for Java	70
Using the AWS SDK for .NET	71
Using the SDK for PHP	72
Using the REST API	73
Transfer Acceleration	73
Why use Transfer Acceleration?	74
Getting Started	74
Requirements for Using Amazon S3 Transfer Acceleration	75
Transfer Acceleration Examples	76
Requester Pays Buckets	80
Configure with the Console	81
Configure with the REST API	81
Charge Details	83
Access Control	84
Billing and Usage Reporting	84
Billing Reports	84
Usage Report	86
Understanding Billing and Usage Reports	87
Using Cost Allocation Tags	95
Objects	98
Object Key and Metadata	99
Object Keys	99
Object Metadata	101
Storage Classes	103
Storage Classes for Frequently Accessed Objects	104
Storage Class That Automatically Optimizes Frequently and Infrequently Accessed Objects	104
Storage Classes for Infrequently Accessed Objects	105
Storage Classes for Archiving Objects	106
Comparing the Amazon S3 Storage Classes	106
Setting the Storage Class of an Object	107
Subresources	108
Versioning	108
Object Tagging	110
API Operations Related to Object Tagging	112
Object Tagging and Additional Information	113
Managing Object Tags	116
Lifecycle Management	119
When Should I Use Lifecycle Configuration?	120
How Do I Configure a Lifecycle?	120
Additional Considerations	120
Lifecycle Configuration Elements	127
Examples of Lifecycle Configuration	133
Setting Lifecycle Configuration	143
Cross-Origin Resource Sharing (CORS)	151

Cross-Origin Resource Sharing: Use-case Scenarios	152
How Do I Configure CORS on My Bucket?	152
How Does Amazon S3 Evaluate the CORS Configuration on a Bucket?	154
Enabling CORS	154
Troubleshooting CORS	160
Operations on Objects	160
Getting Objects	161
Uploading Objects	169
Copying Objects	210
Listing Object Keys	221
Deleting Objects	227
Selecting Content from Objects	245
Restoring Archived Objects	248
Querying Archived Objects	253
Storage Class Analysis	257
How to Set Up Storage Class Analysis	257
Storage Class Analysis	258
How Can I Export Storage Class Analysis Data?	260
Storage Class Analysis Export File Layout	261
Amazon S3 Analytics REST APIs	262
Security	263
Data Protection	263
Internetwork Privacy	264
Data Encryption	264
Identity and Access Management	301
Introduction	301
Using Bucket Policies and User Policies	341
Managing Access with ACLs	403
Blocking Public Access	414
Logging and Monitoring	421
Compliance Validation	422
Inventory	422
Resilience	431
Backup Encryption	432
Versioning	432
Locking Objects	453
Infrastructure Security	461
Configuration and Vulnerability Analysis	462
Security Best Practices	463
Amazon S3 Preventative Security Best Practices	463
Amazon S3 Monitoring and Auditing Best Practices	465
Batch Operations	468
Terminology	468
The Basics: Jobs	468
How a Job Works	469
Specifying a Manifest	469
Creating a Job	470
Creating a Job Request	470
Creating a Job Response	471
Granting Permissions for Batch Operations	471
Related Resources	475
Operations	475
PUT Object Copy	475
Initiate Restore Object	476
Invoke a Lambda Function	477
Put Object ACL	484
Put Object Tagging	484

Managing Jobs	485
Listing Jobs	485
Viewing Job Details	485
Assigning Job Priority	486
Job Status	486
Tracking Job Failure	488
Notifications and Logging	488
Completion Reports	489
Examples	489
Completion Report Examples	489
Cross Account Copy	491
AWS CLI Examples	495
Java Examples	498
Hosting a Static Website	503
Website Endpoints	504
Key Differences Between the Amazon Website and the REST API Endpoint	505
Configuring a Bucket for Website Hosting	505
Enabling Website Hosting	506
Configuring Index Document Support	506
Permissions Required for Website Access	508
(Optional) Configuring Web Traffic Logging	508
(Optional) Custom Error Document Support	509
(Optional) Configuring a Redirect	510
Example Walkthroughs	517
Example: Setting up a Static Website	517
Example: Setting up a Static Website Using a Custom Domain	519
Example: Speed Up Your Website with Amazon CloudFront	525
Clean Up Example Resources	528
Notifications	530
Overview	530
How to Enable Event Notifications	532
Event Notification Types and Destinations	533
Supported Event Types	533
Supported Destinations	534
Configuring Notifications with Object Key Name Filtering	534
Examples of Valid Notification Configurations with Object Key Name Filtering	535
Examples of Notification Configurations with Invalid Prefix/Suffix Overlapping	537
Granting Permissions to Publish Event Notification Messages to a Destination	539
Granting Permissions to Invoke an AWS Lambda Function	539
Granting Permissions to Publish Messages to an SNS Topic or an SQS Queue	539
Example Walkthrough 1	541
Walkthrough Summary	541
Step 1: Create an Amazon SNS Topic	542
Step 2: Create an Amazon SQS Queue	542
Step 3: Add a Notification Configuration to Your Bucket	544
Step 4: Test the Setup	546
Example Walkthrough 2	546
Event Message Structure	546
Replication	551
Types of Object Replication	551
When to Use Replication	551
When to Use CRR	551
When to Use SRR	552
Requirements for Replication	552
What Does Amazon S3 Replicate?	553
What Is Replicated?	553
What Isn't Replicated?	554

Related Topics	555
Overview of Setting Up Replication	555
Replication Configuration Overview	556
Setting Up Permissions for Replication	564
Additional Replication Configurations	567
Changing the Replica Owner	568
Replicating Encrypted Objects	570
Replication Walkthroughs	575
Example 1: Configuring for Buckets in the Same Account	575
Example 2: Configuring for Buckets in Different Accounts	584
Example 3: Changing Replica Owner	585
Example 4: Replicating Encrypted Objects	589
Replication Status Information	594
Related Topics	595
Troubleshooting Replication	595
Related Topics	596
Replication Additional Considerations	596
Lifecycle Configuration and Object Replicas	597
Versioning Configuration and Replication Configuration	597
Logging Configuration and Replication Configuration	597
CRR and the Destination Region	598
Pausing Replication	598
Related Topics	598
Request Routing	599
Request Redirection and the REST API	599
DNS Routing	599
Temporary Request Redirection	600
Permanent Request Redirection	602
Request Redirection Examples	602
DNS Considerations	602
Optimizing Amazon S3 Performance	604
Performance Guidelines	604
Measure Performance	605
Scale Horizontally	605
Use Byte-Range Fetches	605
Retry Requests	605
Combine Amazon S3 and Amazon EC2 the Same Region	606
Use Transfer Acceleration to Minimize Latency	606
Use the Latest AWS SDKs	606
Performance Design Patterns	606
Caching Frequently Accessed Content	607
Timeouts and Retries for Latency-Sensitive Apps	607
Horizontal Scaling and Request Parallelization	608
Accelerating Geographically Disparate Data Transfers	609
Monitoring	610
Monitoring Tools	610
Automated Tools	610
Manual Tools	611
Monitoring Metrics with CloudWatch	611
Metrics and Dimensions	612
Amazon S3 CloudWatch Daily Storage Metrics for Buckets	612
Amazon S3 CloudWatch Request Metrics	612
Amazon S3 CloudWatch Dimensions	615
Accessing CloudWatch Metrics	616
Related Resources	617
Metrics Configurations for Buckets	617
Best-Effort CloudWatch Metrics Delivery	618

Filtering Metrics Configurations	618
How to Add Metrics Configurations	618
Logging with Amazon S3	619
Logging API Calls with AWS CloudTrail	621
Amazon S3 Information in CloudTrail	621
Using CloudTrail Logs with Amazon S3 Server Access Logs and CloudWatch Logs	625
Example: Amazon S3 Log File Entries	626
Related Resources	628
Identify Amazon S3 Requests Using CloudTrail	628
How CloudTrail Captures Requests Made to Amazon S3	628
Enabling CloudTrail Event Logging for S3 Buckets and Objects	629
Identifying Requests Made to Amazon S3 in a CloudTrail Log	629
Using AWS CloudTrail to Identify Amazon S3 Signature Version 2 Requests	631
Using CloudTrail to Identify Access to Amazon S3 Objects	633
Related Resources	628
BitTorrent	636
How You are Charged for BitTorrent Delivery	636
Using BitTorrent to Retrieve Objects Stored in Amazon S3	637
Publishing Content Using Amazon S3 and BitTorrent	637
Error Handling	639
The REST Error Response	639
Response Headers	639
Error Response	640
The SOAP Error Response	640
Amazon S3 Error Best Practices	641
Retry InternalErrors	641
Tune Application for Repeated SlowDown errors	641
Isolate Errors	641
Troubleshooting Amazon S3	643
Troubleshooting Amazon S3 by Symptom	643
Significant Increases in HTTP 503 Responses to Requests to Buckets with Versioning Enabled	643
Unexpected Behavior When Accessing Buckets Set with CORS	643
Getting Amazon S3 Request IDs for AWS Support	644
Using HTTP to Obtain Request IDs	644
Using a Web Browser to Obtain Request IDs	644
Using AWS SDKs to Obtain Request IDs	644
Using the AWS CLI to Obtain Request IDs	646
Related Topics	646
Server Access Logging	647
How to Enable Server Access Logging	647
Log Object Key Format	648
How Are Logs Delivered?	649
Best Effort Server Log Delivery	649
Bucket Logging Status Changes Take Effect Over Time	649
Enabling Logging Using the Console	649
Enabling Logging Programmatically	650
Enabling Logging	650
Granting the Log Delivery Group WRITE and READ_ACP Permissions	650
Example: AWS SDK for .NET	651
Related Resources	652
Log Format	653
Additional Logging for Copy Operations	657
Custom Access Log Information	661
Programming Considerations for Extensible Server Access Log Format	661
Deleting Log Files	661
Related Resources	662
Using Amazon S3 access logs to identify Amazon S3 requests	662

Enabling Amazon S3 Access Logs for Requests	662
Querying Amazon S3 Access Logs for Requests	664
Using Amazon S3 Log Files to Identify SigV2 Requests	666
Using Amazon S3 Log Files to Identify Object Access	667
Related Resources	668
AWS SDKs and Explorers	669
Specifying the Signature Version in Request Authentication	670
AWS Signature Version 2 Turned Off (Deprecated) for Amazon S3	671
Moving from Signature Version 2 to Signature Version 4	673
Setting Up the AWS CLI	675
Using the AWS SDK for Java	676
The Java API Organization	677
Testing the Amazon S3 Java Code Examples	677
Using the AWS SDK for .NET	677
The .NET API Organization	678
Running the Amazon S3 .NET Code Examples	678
Using the AWS SDK for PHP and Running PHP Examples	678
AWS SDK for PHP Levels	679
Running PHP Examples	679
Related Resources	679
Using the AWS SDK for Ruby - Version 3	679
The Ruby API Organization	680
Testing the Ruby Script Examples	680
Using the AWS SDK for Python (Boto)	681
Using the AWS Mobile SDKs for iOS and Android	681
More Info	681
Using the AWS Amplify JavaScript Library	681
More Info	681
Appendices	682
Appendix A: Using the SOAP API	682
Common SOAP API Elements	682
Authenticating SOAP Requests	682
Setting Access Policy with SOAP	683
Appendix B: Authenticating Requests (AWS Signature Version 2)	684
Authenticating Requests Using the REST API	686
Signing and Authenticating REST Requests	688
Browser-Based Uploads Using POST	697
Resources	713
SQL Reference	714
SELECT Command	714
SELECT List	714
FROM Clause	714
WHERE Clause	718
LIMIT Clause (Amazon S3 Select only)	718
Attribute Access	718
Case Sensitivity of Header/Attribute Names	719
Using Reserved Keywords as User-Defined Terms	720
Scalar Expressions	720
Data Types	721
Data Type Conversions	721
Supported Data Types	721
Operators	721
Logical Operators	721
Comparison Operators	722
Pattern Matching Operators	722
Math Operators	722
Operator Precedence	722

Reserved Keywords	723
SQL Functions	727
Aggregate Functions (Amazon S3 Select only)	727
Conditional Functions	728
Conversion Functions	729
Date Functions	729
String Functions	735
Document History	738
Earlier Updates	741
AWS Glossary	755

What is Amazon S3?

Amazon Simple Storage Service is storage for the Internet. It is designed to make web-scale computing easier for developers.

Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

This guide explains the core concepts of Amazon S3, such as buckets and objects, and how to work with these resources using the Amazon S3 application programming interface (API).

How Do I...?

Information	Relevant Sections
General product overview and pricing	Amazon S3
Get a quick hands-on introduction to Amazon S3	Amazon Simple Storage Service Getting Started Guide
Learn about Amazon S3 key terminology and concepts	Introduction to Amazon S3 (p. 2)
How do I work with buckets?	Working with Amazon S3 Buckets (p. 53)
How do I work with objects?	Working with Amazon S3 Objects (p. 98)
How do I make requests?	Making Requests (p. 10)
How do I manage access to my resources?	Identity and Access Management in Amazon S3 (p. 301)

Introduction to Amazon S3

This introduction to Amazon Simple Storage Service (Amazon S3) provides a detailed summary of this web service. After reading this section, you should have a good idea of what it offers and how it can fit in with your business.

Topics

- [Overview of Amazon S3 and This Guide](#) (p. 2)
- [Advantages of Using Amazon S3](#) (p. 2)
- [Amazon S3 Concepts](#) (p. 3)
- [Amazon S3 Features](#) (p. 6)
- [Amazon S3 Application Programming Interfaces \(API\)](#) (p. 8)
- [Paying for Amazon S3](#) (p. 9)
- [Related Services](#) (p. 9)

Overview of Amazon S3 and This Guide

Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web.

This guide describes how you send requests to create buckets, store and retrieve your objects, and manage permissions on your resources. The guide also describes access control and the authentication process. Access control defines who can access objects and buckets within Amazon S3, and the type of access (for example, READ and WRITE). The authentication process verifies the identity of a user who is trying to access Amazon Web Services (AWS).

Advantages of Using Amazon S3

Amazon S3 is intentionally built with a minimal feature set that focuses on simplicity and robustness. Following are some of the advantages of using Amazon S3:

- **Creating buckets** – Create and name a bucket that stores data. Buckets are the fundamental container in Amazon S3 for data storage.
- **Storing data** – Store an infinite amount of data in a bucket. Upload as many objects as you like into an Amazon S3 bucket. Each object can contain up to 5 TB of data. Each object is stored and retrieved using a unique developer-assigned key.
- **Downloading data** – Download your data or enable others to do so. Download your data anytime you like, or allow others to do the same.
- **Permissions** – Grant or deny access to others who want to upload or download data into your Amazon S3 bucket. Grant upload and download permissions to three types of users. Authentication mechanisms can help keep data secure from unauthorized access.
- **Standard interfaces** – Use standards-based REST and SOAP interfaces designed to work with any internet-development toolkit.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 Concepts

This section describes key concepts and terminology you need to understand to use Amazon S3 effectively. They are presented in the order that you will most likely encounter them.

Topics

- [Buckets \(p. 3\)](#)
- [Objects \(p. 3\)](#)
- [Keys \(p. 3\)](#)
- [Regions \(p. 4\)](#)
- [Amazon S3 Data Consistency Model \(p. 4\)](#)

Buckets

A bucket is a container for objects stored in Amazon S3. Every object is contained in a bucket. For example, if the object named `photos/puppy.jpg` is stored in the `johnsmith` bucket, then it is addressable using the URL `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg`.

Buckets serve several purposes:

- They organize the Amazon S3 namespace at the highest level.
- They identify the account responsible for storage and data transfer charges.
- They play a role in access control.
- They serve as the unit of aggregation for usage reporting.

You can configure buckets so that they are created in a specific AWS Region. For more information, see [Accessing a Bucket \(p. 55\)](#). You can also configure a bucket so that every time an object is added to it, Amazon S3 generates a unique version ID and assigns it to the object. For more information, see [Using Versioning \(p. 432\)](#).

For more information about buckets, see [Working with Amazon S3 Buckets \(p. 53\)](#).

Objects

Objects are the fundamental entities stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3. The metadata is a set of name-value pairs that describe the object. These include some default metadata, such as the date last modified, and standard HTTP metadata, such as `Content-Type`. You can also specify custom metadata at the time the object is stored.

An object is uniquely identified within a bucket by a key (name) and a version ID. For more information, see [Keys \(p. 3\)](#) and [Using Versioning \(p. 432\)](#).

Keys

A key is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. The combination of a bucket, key, and version ID uniquely identify each object. So you

can think of Amazon S3 as a basic data map between "bucket + key + version" and the object itself. Every object in Amazon S3 can be uniquely addressed through the combination of the web service endpoint, bucket name, key, and optionally, a version. For example, in the URL `http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsd1`, "doc" is the name of the bucket and "2006-03-01/AmazonS3.wsd1" is the key.

For more information about object keys, see [Object Keys](#).

Regions

You can choose the geographical AWS Region where Amazon S3 will store the buckets that you create. You might choose a Region to optimize latency, minimize costs, or address regulatory requirements. Objects stored in a Region never leave the Region unless you explicitly transfer them to another Region. For example, objects stored in the EU (Ireland) Region never leave it.

Note

You can only access Amazon S3 and its features in AWS Regions that are enabled for your account.

For a list of Amazon S3 Regions and endpoints, see [Regions and Endpoints](#) in the *AWS General Reference*.

Amazon S3 Data Consistency Model

Amazon S3 provides read-after-write consistency for PUTS of new objects in your S3 bucket in all Regions with one caveat. The caveat is that if you make a HEAD or GET request to the key name (to find if the object exists) before creating the object, Amazon S3 provides eventual consistency for read-after-write.

Amazon S3 offers eventual consistency for overwrite PUTS and DELETES in all Regions.

Updates to a single key are atomic. For example, if you PUT to an existing key, a subsequent read might return the old data or the updated data, but it never returns corrupted or partial data.

Amazon S3 achieves high availability by replicating data across multiple servers within AWS data centers. If a PUT request is successful, your data is safely stored. However, information about the changes must replicate across Amazon S3, which can take some time, and so you might observe the following behaviors:

- A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
- A process replaces an existing object and immediately tries to read it. Until the change is fully propagated, Amazon S3 might return the previous data.
- A process deletes an existing object and immediately tries to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.
- A process deletes an existing object and immediately lists keys within its bucket. Until the deletion is fully propagated, Amazon S3 might list the deleted object.

Note

Amazon S3 does not currently support object locking. If two PUT requests are simultaneously made to the same key, the request with the latest timestamp wins. If this is an issue, you will need to build an object-locking mechanism into your application.

Updates are key-based. There is no way to make atomic updates across keys. For example, you cannot make the update of one key dependent on the update of another key unless you design this functionality into your application.

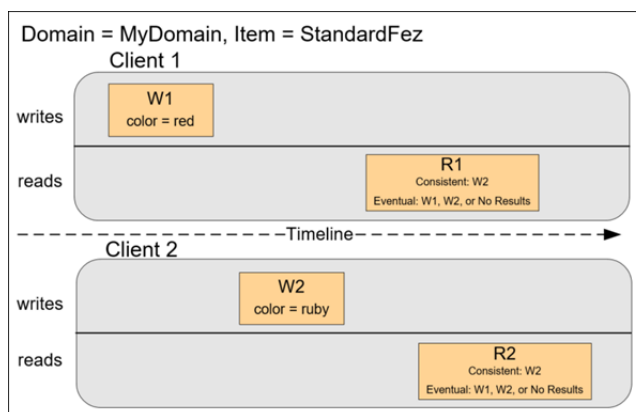
The following table describes the characteristics of an eventually consistent read and a consistent read.

Eventually Consistent Read	Consistent Read
Stale reads possible	No stale reads
Lowest read latency	Potential higher read latency
Highest read throughput	Potential lower read throughput

Concurrent Applications

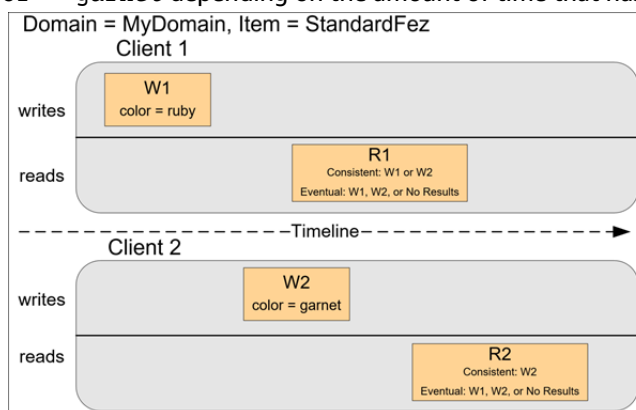
This section provides examples of eventually consistent and consistent read requests when multiple clients are writing to the same items.

In this example, both W1 (write 1) and W2 (write 2) complete before the start of R1 (read 1) and R2 (read 2). For a consistent read, R1 and R2 both return `color = ruby`. For an eventually consistent read, R1 and R2 might return `color = red` or `color = ruby` depending on the amount of time that has elapsed.



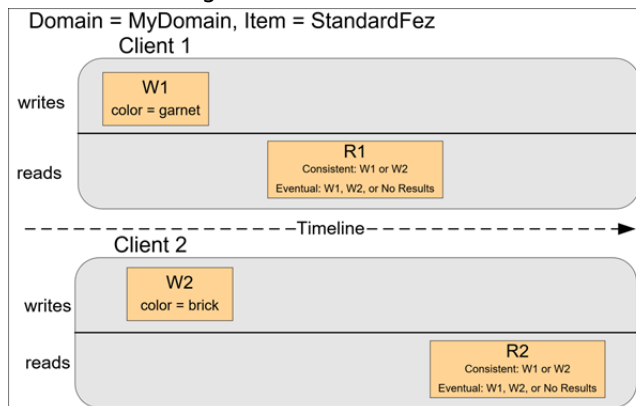
In the next example, W2 does not complete before the start of R1. Therefore, R1 might return `color = ruby` or `color = garnet` for either a consistent read or an eventually consistent read. Also, depending on the amount of time that has elapsed, an eventually consistent read might return no results.

For a consistent read, R2 returns `color = garnet`. For an eventually consistent read, R2 might return `color = ruby` or `color = garnet` depending on the amount of time that has elapsed.



In the last example, Client 2 performs W2 before Amazon S3 returns a success for W1, so the outcome of the final value is unknown (`color = garnet` or `color = brick`). Any subsequent reads (consistent

read or eventually consistent) might return either value. Also, depending on the amount of time that has elapsed, an eventually consistent read might return no results.



Amazon S3 Features

This section describes important Amazon S3 features.

Topics

- [Storage Classes \(p. 6\)](#)
- [Bucket Policies \(p. 6\)](#)
- [AWS Identity and Access Management \(p. 7\)](#)
- [Access Control Lists \(p. 7\)](#)
- [Versioning \(p. 7\)](#)
- [Operations \(p. 8\)](#)

Storage Classes

Amazon S3 offers a range of storage classes designed for different use cases. These include Amazon S3 STANDARD for general-purpose storage of frequently accessed data, Amazon S3 STANDARD_IA for long-lived, but less frequently accessed data, and GLACIER for long-term archive.

For more information, see [Amazon S3 Storage Classes \(p. 103\)](#).

Bucket Policies

Bucket policies provide centralized access control to buckets and objects based on a variety of conditions, including Amazon S3 operations, requesters, resources, and aspects of the request (for example, IP address). The policies are expressed in the *access policy language* and enable centralized management of permissions. The permissions attached to a bucket apply to all of the objects in that bucket.

Both individuals and companies can use bucket policies. When companies register with Amazon S3, they create an *account*. Thereafter, the company becomes synonymous with the account. Accounts are financially responsible for the AWS resources that they (and their employees) create. Accounts have the power to grant bucket policy permissions and assign employees permissions based on a variety of conditions. For example, an account could create a policy that gives a user write access:

- To a particular S3 bucket

- From an account's corporate network
- During business hours

An account can grant one user limited read and write access, but allow another to create and delete buckets also. An account could allow several field offices to store their daily reports in a single bucket. It could allow each office to write only to a certain set of names (for example, "Nevada/*" or "Utah/*") and only from the office's IP address range.

Unlike access control lists (described later), which can add (grant) permissions only on individual objects, policies can either add or deny permissions across all (or a subset) of objects within a bucket. With one request, an account can set the permissions of any number of objects in a bucket. An account can use wildcards (similar to regular expression operators) on Amazon Resource Names (ARNs) and other values. The account could then control access to groups of objects that begin with a common prefix or end with a given extension, such as *.html*.

Only the bucket owner is allowed to associate a policy with a bucket. Policies (written in the access policy language) *allow* or *deny* requests based on the following:

- Amazon S3 bucket operations (such as `PUT` `acl`), and object operations (such as `PUT` `Object`, or `GET` `Object`)
- Requester
- Conditions specified in the policy

An account can control access based on specific Amazon S3 operations, such as `GetObject`, `GetObjectVersion`, `DeleteObject`, or `DeleteBucket`.

The conditions can be such things as IP addresses, IP address ranges in CIDR notation, dates, user agents, HTTP referrer, and transports (HTTP and HTTPS).

For more information, see [Using Bucket Policies and User Policies \(p. 341\)](#).

AWS Identity and Access Management

You can use AWS Identity and Access Management (IAM) to manage access to your Amazon S3 resources.

For example, you can use IAM with Amazon S3 to control the type of access a user or group of users has to specific parts of an Amazon S3 bucket your AWS account owns.

For more information about IAM, see the following:

- [AWS Identity and Access Management \(IAM\)](#)
- [Getting Started](#)
- [IAM User Guide](#)

Access Control Lists

You can control access to each of your buckets and objects using an access control list (ACL). For more information, see [Managing Access with ACLs \(p. 403\)](#).

Versioning

You can use *versioning* to keep multiple versions of an object in the same bucket. For more information, see [Object Versioning \(p. 108\)](#).

Operations

Following are the most common operations that you'll execute through the API.

Common Operations

- **Create a bucket** – Create and name your own bucket in which to store your objects.
- **Write an object** – Store data by creating or overwriting an object. When you write an object, you specify a unique key in the namespace of your bucket. This is also a good time to specify any access control you want on the object.
- **Read an object** – Read data back. You can download the data via HTTP or BitTorrent.
- **Delete an object** – Delete some of your data.
- **List keys** – List the keys contained in one of your buckets. You can filter the key list based on a prefix.

These operations and all other functionality are described in detail throughout this guide.

Amazon S3 Application Programming Interfaces (API)

The Amazon S3 architecture is designed to be programming language-neutral, using AWS supported interfaces to store and retrieve objects.

Amazon S3 provides a REST and a SOAP interface. They are similar, but there are some differences. For example, in the REST interface, metadata is returned in HTTP headers. Because we only support HTTP requests of up to 4 KB (not including the body), the amount of metadata you can supply is restricted.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The REST Interface

The REST API is an HTTP interface to Amazon S3. Using REST, you use standard HTTP requests to create, fetch, and delete buckets and objects.

You can use any toolkit that supports HTTP to use the REST API. You can even use a browser to fetch objects, as long as they are anonymously readable.

The REST API uses the standard HTTP headers and status codes, so that standard browsers and toolkits work as expected. In some areas, we have added functionality to HTTP (for example, we added headers to support access control). In these cases, we have done our best to add the new functionality in a way that matched the style of standard HTTP usage.

The SOAP Interface

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The SOAP API provides a SOAP 1.1 interface using document literal encoding. The most common way to use SOAP is to download the WSDL (see <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>),

use a SOAP toolkit such as Apache Axis or Microsoft .NET to create bindings, and then write code that uses the bindings to call Amazon S3.

Paying for Amazon S3

Pricing for Amazon S3 is designed so that you don't have to plan for the storage requirements of your application. Most storage providers force you to purchase a predetermined amount of storage and network transfer capacity: If you exceed that capacity, your service is shut off or you are charged high overage fees. If you do not exceed that capacity, you pay as though you used it all.

Amazon S3 charges you only for what you actually use, with no hidden fees and no overage charges. This gives developers a variable-cost service that can grow with their business while enjoying the cost advantages of the AWS infrastructure.

Before storing anything in Amazon S3, you must register with the service and provide a payment method that is charged at the end of each month. There are no setup fees to begin using the service. At the end of the month, your payment method is automatically charged for that month's usage.

For information about paying for Amazon S3 storage, see [Amazon S3 Pricing](#).

Related Services

After you load your data into Amazon S3, you can use it with other AWS services. The following are the services you might use most frequently:

- **Amazon Elastic Compute Cloud (Amazon EC2)** – This service provides virtual compute resources in the cloud. For more information, see the [Amazon EC2 product details page](#).
- **Amazon EMR** – This service enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It uses a hosted Hadoop framework running on the web-scale infrastructure of Amazon EC2 and Amazon S3. For more information, see the [Amazon EMR product details page](#).
- **AWS Snowball** – This service accelerates transferring large amounts of data into and out of AWS using physical storage devices, bypassing the internet. Each AWS Snowball device type can transport data at faster-than internet speeds. This transport is done by shipping the data in the devices through a regional carrier. For more information, see the [AWS Snowball product details page](#).

Making Requests

Topics

- [About Access Keys \(p. 10\)](#)
- [Request Endpoints \(p. 11\)](#)
- [Making Requests to Amazon S3 over IPv6 \(p. 12\)](#)
- [Making Requests Using the AWS SDKs \(p. 19\)](#)
- [Making Requests Using the REST API \(p. 44\)](#)

Amazon S3 is a REST service. You can send requests to Amazon S3 using the REST API or the AWS SDK (see [Sample Code and Libraries](#)) wrapper libraries that wrap the underlying Amazon S3 REST API, simplifying your programming tasks.

Every interaction with Amazon S3 is either authenticated or anonymous. Authentication is a process of verifying the identity of the requester trying to access an Amazon Web Services (AWS) product. Authenticated requests must include a signature value that authenticates the request sender. The signature value is, in part, generated from the requester's AWS access keys (access key ID and secret access key). For more information about getting access keys, see [How Do I Get Security Credentials?](#) in the *AWS General Reference*.

If you are using the AWS SDK, the libraries compute the signature from the keys you provide. However, if you make direct REST API calls in your application, you must write the code to compute the signature and add it to the request.

About Access Keys

The following sections review the types of access keys that you can use to make authenticated requests.

AWS Account Access Keys

The account access keys provide full access to the AWS resources owned by the account. The following are examples of access keys:

- Access key ID (a 20-character, alphanumeric string). For example: AKIAIOSFODNN7EXAMPLE
- Secret access key (a 40-character string). For example: wJalrXUtnFEMI/K7MDENG/bPxRfCYEXAMPLEKEY

The access key ID uniquely identifies an AWS account. You can use these access keys to send authenticated requests to Amazon S3.

IAM User Access Keys

You can create one AWS account for your company; however, there may be several employees in the organization who need access to your organization's AWS resources. Sharing your AWS account access keys reduces security, and creating individual AWS accounts for each employee might not be practical.

Also, you cannot easily share resources such as buckets and objects because they are owned by different accounts. To share resources, you must grant permissions, which is additional work.

In such scenarios, you can use AWS Identity and Access Management (IAM) to create users under your AWS account with their own access keys and attach IAM user policies granting appropriate resource access permissions to them. To better manage these users, IAM enables you to create groups of users and grant group-level permissions that apply to all users in that group.

These users are referred to as IAM users that you create and manage within AWS. The parent account controls a user's ability to access AWS. Any resources an IAM user creates are under the control of and paid for by the parent AWS account. These IAM users can send authenticated requests to Amazon S3 using their own security credentials. For more information about creating and managing users under your AWS account, go to the [AWS Identity and Access Management product details page](#).

Temporary Security Credentials

In addition to creating IAM users with their own access keys, IAM also enables you to grant temporary security credentials (temporary access keys and a security token) to any IAM user to enable them to access your AWS services and resources. You can also manage users in your system outside AWS. These are referred to as federated users. Additionally, users can be applications that you create to access your AWS resources.

IAM provides the AWS Security Token Service API for you to request temporary security credentials. You can use either the AWS STS API or the AWS SDK to request these credentials. The API returns temporary security credentials (access key ID and secret access key), and a security token. These credentials are valid only for the duration you specify when you request them. You use the access key ID and secret key the same way you use them when sending requests using your AWS account or IAM user access keys. In addition, you must include the token in each request you send to Amazon S3.

An IAM user can request these temporary security credentials for their own use or hand them out to federated users or applications. When requesting temporary security credentials for federated users, you must provide a user name and an IAM policy defining the permissions you want to associate with these temporary security credentials. The federated user cannot get more permissions than the parent IAM user who requested the temporary credentials.

You can use these temporary security credentials in making requests to Amazon S3. The API libraries compute the necessary signature value using those credentials to authenticate your request. If you send requests using expired credentials, Amazon S3 denies the request.

For information on signing requests using temporary security credentials in your REST API requests, see [Signing and Authenticating REST Requests \(p. 688\)](#). For information about sending requests using AWS SDKs, see [Making Requests Using the AWS SDKs \(p. 19\)](#).

For more information about IAM support for temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*.

For added security, you can require multifactor authentication (MFA) when accessing your Amazon S3 resources by configuring a bucket policy. For information, see [Adding a Bucket Policy to Require MFA \(p. 375\)](#). After you require MFA to access your Amazon S3 resources, the only way you can access these resources is by providing temporary credentials that are created with an MFA key. For more information, see the [AWS Multi-Factor Authentication](#) detail page and [Configuring MFA-Protected API Access](#) in the *IAM User Guide*.

Request Endpoints

You send REST requests to the service's predefined endpoint. For a list of all AWS services and their corresponding endpoints, go to [Regions and Endpoints](#) in the *AWS General Reference*.

Making Requests to Amazon S3 over IPv6

Amazon Simple Storage Service (Amazon S3) supports the ability to access S3 buckets using the Internet Protocol version 6 (IPv6), in addition to the IPv4 protocol. Amazon S3 dual-stack endpoints support requests to S3 buckets over IPv6 and IPv4. There are no additional charges for accessing Amazon S3 over IPv6. For more information about pricing, see [Amazon S3 Pricing](#).

Topics

- [Getting Started Making Requests over IPv6 \(p. 12\)](#)
- [Using IPv6 Addresses in IAM Policies \(p. 13\)](#)
- [Testing IP Address Compatibility \(p. 14\)](#)
- [Using Amazon S3 Dual-Stack Endpoints \(p. 14\)](#)

Getting Started Making Requests over IPv6

To make a request to an S3 bucket over IPv6, you need to use a dual-stack endpoint. The next section describes how to make requests over IPv6 by using dual-stack endpoints.

The following are some things you should know before trying to access a bucket over IPv6:

- The client and the network accessing the bucket must be enabled to use IPv6.
- Both virtual hosted-style and path style requests are supported for IPv6 access. For more information, see [Amazon S3 Dual-Stack Endpoints \(p. 14\)](#).
- If you use source IP address filtering in your AWS Identity and Access Management (IAM) user or bucket policies, you need to update the policies to include IPv6 address ranges. For more information, see [Using IPv6 Addresses in IAM Policies \(p. 13\)](#).
- When using IPv6, server access log files output IP addresses in an IPv6 format. You need to update existing tools, scripts, and software that you use to parse Amazon S3 log files so that they can parse the IPv6 formatted Remote IP addresses. For more information, see [Amazon S3 Server Access Log Format \(p. 653\)](#) and [Amazon S3 Server Access Logging \(p. 647\)](#).

Note

If you experience issues related to the presence of IPv6 addresses in log files, contact [AWS Support](#).

Making Requests over IPv6 by Using Dual-Stack Endpoints

You make requests with Amazon S3 API calls over IPv6 by using dual-stack endpoints. The Amazon S3 API operations work the same way whether you're accessing Amazon S3 over IPv6 or over IPv4. Performance should be the same too.

When using the REST API, you access a dual-stack endpoint directly. For more information, see [Dual-Stack Endpoints \(p. 14\)](#).

When using the AWS Command Line Interface (AWS CLI) and AWS SDKs, you can use a parameter or flag to change to a dual-stack endpoint. You can also specify the dual-stack endpoint directly as an override of the Amazon S3 endpoint in the config file.

You can use a dual-stack endpoint to access a bucket over IPv6 from any of the following:

- The AWS CLI, see [Using Dual-Stack Endpoints from the AWS CLI \(p. 15\)](#).
- The AWS SDKs, see [Using Dual-Stack Endpoints from the AWS SDKs \(p. 16\)](#).

- The REST API, see [Making Requests to Dual-Stack Endpoints by Using the REST API \(p. 45\)](#).

Features Not Available over IPv6

The following features are not currently supported when accessing an S3 bucket over IPv6:

- Static website hosting from an S3 bucket
- BitTorrent

Using IPv6 Addresses in IAM Policies

Before trying to access a bucket using IPv6, you must ensure that any IAM user or S3 bucket policies that are used for IP address filtering are updated to include IPv6 address ranges. IP address filtering policies that are not updated to handle IPv6 addresses may result in clients incorrectly losing or gaining access to the bucket when they start using IPv6. For more information about managing access permissions with IAM, see [Identity and Access Management in Amazon S3 \(p. 301\)](#).

IAM policies that filter IP addresses use [IP Address Condition Operators](#). The following bucket policy identifies the 54.240.143.* range of allowed IPv4 addresses by using IP address condition operators. Any IP addresses outside of this range will be denied access to the bucket (`examplebucket`). Since all IPv6 addresses are outside of the allowed range, this policy prevents IPv6 addresses from being able to access `examplebucket`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "IpAddress": {"aws:SourceIp": "54.240.143.0/24"}
      }
    }
  ]
}
```

You can modify the bucket policy's `Condition` element to allow both IPv4 (54.240.143.0/24) and IPv6 (2001:DB8:1234:5678::/64) address ranges as shown in the following example. You can use the same type of `Condition` block shown in the example to update both your IAM user and bucket policies.

```
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": [
          "54.240.143.0/24",
          "2001:DB8:1234:5678::/64"
        ]
      }
    }
  ]
}
```

Before using IPv6 you must update all relevant IAM user and bucket policies that use IP address filtering to allow IPv6 address ranges. We recommend that you update your IAM policies with your organization's IPv6 address ranges in addition to your existing IPv4 address ranges. For an example of a bucket policy that allows access over both IPv6 and IPv4, see [Restricting Access to Specific IP Addresses \(p. 372\)](#).

You can review your IAM user policies using the IAM console at <https://console.aws.amazon.com/iam/>. For more information about IAM, see the [IAM User Guide](#). For information about editing S3 bucket policies, see [How Do I Add an S3 Bucket Policy?](#) in the *Amazon Simple Storage Service Console User Guide*.

Testing IP Address Compatibility

If you are using Linux/Unix or Mac OS X, you can test whether you can access a dual-stack endpoint over IPv6 by using the `curl` command as shown in the following example:

Example

```
curl -v http://s3.dualstack.us-west-2.amazonaws.com/
```

You get back information similar to the following example. If you are connected over IPv6 the connected IP address will be an IPv6 address.

```
* About to connect() to s3-us-west-2.amazonaws.com port 80 (#0)
* Trying IPv6 address... connected
* Connected to s3.dualstack.us-west-2.amazonaws.com (IPv6 address) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.18.1 (x86_64-unknown-linux-gnu) libcurl/7.18.1 OpenSSL/1.0.1t
zlib/1.2.3
> Host: s3.dualstack.us-west-2.amazonaws.com
```

If you are using Microsoft Windows 7 or Windows 10, you can test whether you can access a dual-stack endpoint over IPv6 or IPv4 by using the `ping` command as shown in the following example.

```
ping ipv6.s3.dualstack.us-west-2.amazonaws.com
```

Using Amazon S3 Dual-Stack Endpoints

Amazon S3 dual-stack endpoints support requests to S3 buckets over IPv6 and IPv4. This section describes how to use dual-stack endpoints.

Topics

- [Amazon S3 Dual-Stack Endpoints \(p. 14\)](#)
- [Using Dual-Stack Endpoints from the AWS CLI \(p. 15\)](#)
- [Using Dual-Stack Endpoints from the AWS SDKs \(p. 16\)](#)
- [Using Dual-Stack Endpoints from the REST API \(p. 18\)](#)

Amazon S3 Dual-Stack Endpoints

When you make a request to a dual-stack endpoint, the bucket URL resolves to an IPv6 or an IPv4 address. For more information about accessing a bucket over IPv6, see [Making Requests to Amazon S3 over IPv6 \(p. 12\)](#).

When using the REST API, you directly access an Amazon S3 endpoint by using the endpoint name (URI). You can access an S3 bucket through a dual-stack endpoint by using a virtual hosted-style or a path-style endpoint name. Amazon S3 supports only regional dual-stack endpoint names, which means that you must specify the region as part of the name.

Use the following naming conventions for the dual-stack virtual hosted-style and path-style endpoint names:

- Virtual hosted-style dual-stack endpoint:

`bucketname.s3.dualstack.aws-region.amazonaws.com`

- Path-style dual-stack endpoint:

`s3.dualstack.aws-region.amazonaws.com/bucketname`

For more information about endpoint name style, see [Accessing a Bucket \(p. 55\)](#). For a list of Amazon S3 endpoints, see [Regions and Endpoints](#) in the *AWS General Reference*.

Important

You can use transfer acceleration with dual-stack endpoints. For more information, see [Getting Started with Amazon S3 Transfer Acceleration \(p. 74\)](#).

When using the AWS Command Line Interface (AWS CLI) and AWS SDKs, you can use a parameter or flag to change to a dual-stack endpoint. You can also specify the dual-stack endpoint directly as an override of the Amazon S3 endpoint in the config file. The following sections describe how to use dual-stack endpoints from the AWS CLI and the AWS SDKs.

Using Dual-Stack Endpoints from the AWS CLI

This section provides examples of AWS CLI commands used to make requests to a dual-stack endpoint. For instructions on setting up the AWS CLI, see [Setting Up the AWS CLI \(p. 675\)](#).

You set the configuration value `use_dualstack_endpoint` to `true` in a profile in your AWS Config file to direct all Amazon S3 requests made by the `s3` and `s3api` AWS CLI commands to the dual-stack endpoint for the specified region. You specify the region in the config file or in a command using the `--region` option.

When using dual-stack endpoints with the AWS CLI, both `path` and `virtual` addressing styles are supported. The addressing style, set in the config file, controls if the bucket name is in the hostname or part of the URL. By default, the CLI will attempt to use virtual style where possible, but will fall back to path style if necessary. For more information, see [AWS CLI Amazon S3 Configuration](#).

You can also make configuration changes by using a command, as shown in the following example, which sets `use_dualstack_endpoint` to `true` and `addressing_style` to `virtual` in the default profile.

```
$ aws configure set default.s3.use_dualstack_endpoint true
$ aws configure set default.s3.addressing_style virtual
```

If you want to use a dual-stack endpoint for specified AWS CLI commands only (not all commands), you can use either of the following methods:

- You can use the dual-stack endpoint per command by setting the `--endpoint-url` parameter to `https://s3.dualstack.aws-region.amazonaws.com` or `http://s3.dualstack.aws-region.amazonaws.com` for any `s3` or `s3api` command.

```
$ aws s3api list-objects --bucket bucketname --endpoint-url https://s3.dualstack.aws-region.amazonaws.com
```

- You can set up separate profiles in your AWS Config file. For example, create one profile that sets `use_dualstack_endpoint` to `true` and a profile that does not set `use_dualstack_endpoint`.

When you run a command, specify which profile you want to use, depending upon whether or not you want to use the dual-stack endpoint.

Note

When using the AWS CLI you currently cannot use transfer acceleration with dual-stack endpoints. However, support for the AWS CLI is coming soon. For more information, see [Using Transfer Acceleration from the AWS Command Line Interface \(AWS CLI\)](#) (p. 76).

Using Dual-Stack Endpoints from the AWS SDKs

This section provides examples of how to access a dual-stack endpoint by using the AWS SDKs.

AWS SDK for Java Dual-Stack Endpoint Example

The following example shows how to enable dual-stack endpoints when creating an Amazon S3 client using the AWS SDK for Java.

For instructions on creating and testing a working Java sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

public class DualStackEndpoints {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";

        try {
            // Create an Amazon S3 client with dual-stack endpoints enabled.
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .withDualstackEnabled(true)
                .build();

            s3Client.listObjects(bucketName);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

If you are using the AWS SDK for Java on Windows, you might have to set the following Java virtual machine (JVM) property:

```
java.net.preferIPv6Addresses=true
```

AWS .NET SDK Dual-Stack Endpoint Example

When using the AWS SDK for .NET you use the `AmazonS3Config` class to enable the use of a dual-stack endpoint as shown in the following example.

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DualStackEndpointTest
    {
        private const string bucketName = "**** bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            var config = new AmazonS3Config
            {
                UseDualstackEndpoint = true,
                RegionEndpoint = bucketRegion
            };
            client = new AmazonS3Client(config);
            Console.WriteLine("Listing objects stored in a bucket");
            ListingObjectsAsync().Wait();
        }

        private static async Task ListingObjectsAsync()
        {
            try
            {
                var request = new ListObjectsV2Request
                {
                    BucketName = bucketName,
                    MaxKeys = 10
                };
                ListObjectsV2Response response;
                do
                {
                    response = await client.ListObjectsV2Async(request);

                    // Process the response.
                    foreach (S3Object entry in response.S3Objects)
                    {
                        Console.WriteLine("key = {0} size = {1}",
                            entry.Key, entry.Size);
                    }
                    Console.WriteLine("Next Continuation Token: {0}",
                        response.NextContinuationToken);
                    request.ContinuationToken = response.NextContinuationToken;
                } while (response.IsTruncated == true);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("An AmazonS3Exception was thrown. Exception: " +
                    amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }
    }
}
```

```
}  
  }  
    }  
  }  
}
```

For a full .NET sample for listing objects, see [Listing Keys Using the AWS SDK for .NET \(p. 224\)](#).

For information about how to create and test a working .NET sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Using Dual-Stack Endpoints from the REST API

For information about making requests to dual-stack endpoints by using the REST API, see [Making Requests to Dual-Stack Endpoints by Using the REST API \(p. 45\)](#).

Making Requests Using the AWS SDKs

Topics

- [Making Requests Using AWS Account or IAM User Credentials \(p. 19\)](#)
- [Making Requests Using IAM User Temporary Credentials \(p. 26\)](#)
- [Making Requests Using Federated User Temporary Credentials \(p. 34\)](#)

You can send authenticated requests to Amazon S3 using either the AWS SDK or by making the REST API calls directly in your application. The AWS SDK API uses the credentials that you provide to compute the signature for authentication. If you use the REST API directly in your applications, you must write the necessary code to compute the signature for authenticating your request. For a list of available AWS SDKs go to, [Sample Code and Libraries](#).

Making Requests Using AWS Account or IAM User Credentials

You can use your AWS account or IAM user security credentials to send authenticated requests to Amazon S3. This section provides examples of how you can send authenticated requests using the AWS SDK for Java, AWS SDK for .NET, and AWS SDK for PHP. For a list of available AWS SDKs, go to [Sample Code and Libraries](#).

Topics

- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Java \(p. 20\)](#)
- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for .NET \(p. 21\)](#)
- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for PHP \(p. 23\)](#)
- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Ruby \(p. 24\)](#)

Each of these AWS SDKs uses an SDK-specific credentials provider chain to find and use credentials and perform actions on behalf of the credentials owner. What all these credentials provider chains have in common is that they all look for your local AWS credentials file.

The easiest way to configure credentials for your AWS SDKs is to use an AWS credentials file. If you use the AWS Command Line Interface (AWS CLI), you may already have a local AWS credentials file configured. Otherwise, use the following procedure to set up a credentials file:

To create a local AWS credentials file

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Create a new user with permissions limited to the services and actions that you want your code to have access to. For more information about creating a new IAM user, see [Creating IAM Users \(Console\)](#), and follow the instructions through step 8.
3. Choose **Download .csv** to save a local copy of your AWS credentials.
4. On your computer, navigate to your home directory, and create an `.aws` directory. On Unix-based systems, such as Linux or OS X, this is in the following location:

```
~/ .aws
```

On Windows, this is in the following location:

```
%HOMEPATH%\ .aws
```

5. In the `.aws` directory, create a new file named `credentials`.
6. Open the `credentials.csv` file that you downloaded from the IAM console, and copy its contents into the `credentials` file using the following format:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

7. Save the `credentials` file, and delete the `.csv` file that you downloaded in step 3.

Your shared credentials file is now configured on your local computer, and it's ready to be used with the AWS SDKs.

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Java

To send authenticated requests to Amazon S3 using your AWS account or IAM user credentials, do the following:

- Use the `AmazonS3ClientBuilder` class to create an `AmazonS3Client` instance.
- Execute one of the `AmazonS3Client` methods to send requests to Amazon S3. The client generates the necessary signature from the credentials that you provide and includes it in the request.

The following example performs the preceding tasks. For information on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListObjectsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;

import java.io.IOException;
import java.util.List;

public class MakingRequests {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();
        }
```

```
// Get a list of objects in the bucket, two at a time, and
// print the name and size of each object.
ListObjectsRequest listRequest = new
ListObjectsRequest().withBucketName(bucketName).withMaxKeys(2);
ObjectListing objects = s3Client.listObjects(listRequest);
while (true) {
    List<S3ObjectSummary> summaries = objects.getObjectSummaries();
    for (S3ObjectSummary summary : summaries) {
        System.out.printf("Object \"%s\" retrieved with size %d\n",
summary.getKey(), summary.getSize());
    }
    if (objects.isTruncated()) {
        objects = s3Client.listNextBatchOfObjects(objects);
    } else {
        break;
    }
}
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for .NET

To send authenticated requests using your AWS account or IAM user credentials:

- Create an instance of the `AmazonS3Client` class.
- Execute one of the `AmazonS3Client` methods to send requests to Amazon S3. The client generates the necessary signature from the credentials that you provide and includes it in the request it sends to Amazon S3.

The following C# example shows how to perform the preceding tasks. For information about running the .NET examples in this guide and for instructions on how to store your credentials in a configuration file, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Example

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class MakeS3RequestTest
```

```
{
    private const string bucketName = "**** bucket name ****";
    // Specify your bucket region (an example region is shown).
    private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
    private static IAmazonS3 client;

    public static void Main()
    {
        using (client = new AmazonS3Client(bucketRegion))
        {
            Console.WriteLine("Listing objects stored in a bucket");
            ListingObjectsAsync().Wait();
        }
    }

    static async Task ListingObjectsAsync()
    {
        try
        {
            ListObjectsRequest request = new ListObjectsRequest
            {
                BucketName = bucketName,
                MaxKeys = 2
            };
            do
            {
                ListObjectsResponse response = await client.ListObjectsAsync(request);
                // Process the response.
                foreach (S3Object entry in response.S3Objects)
                {
                    Console.WriteLine("key = {0} size = {1}",
                        entry.Key, entry.Size);
                }

                // If the response is truncated, set the marker to get the next
                // set of keys.
                if (response.IsTruncated)
                {
                    request.Marker = response.NextMarker;
                }
                else
                {
                    request = null;
                }
            } while (request != null);
        }
        catch (AmazonS3Exception e)
        {
            Console.WriteLine("Error encountered on server. Message:'{0}' when writing\nan object", e.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine("Unknown encountered on server. Message:'{0}' when\nwriting an object", e.Message);
        }
    }
}
```

Note

You can create the `AmazonS3Client` client without providing your security credentials. Requests sent using this client are anonymous requests, without a signature. Amazon S3 returns an error if you send anonymous requests for a resource that is not publicly available.

For working examples, see [Working with Amazon S3 Objects \(p. 98\)](#) and [Working with Amazon S3 Buckets \(p. 53\)](#). You can test these examples using your AWS Account or an IAM user credentials.

For example, to list all the object keys in your bucket, see [Listing Keys Using the AWS SDK for .NET \(p. 224\)](#).

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for PHP

This section explains how to use a class from version 3 of the AWS SDK for PHP to send authenticated requests using your AWS account or IAM user credentials. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example shows how the client makes a request using your security credentials to list all of the buckets for your account.

Example

```
require 'vendor/autoload.php';

use Aws\Sts\StsClient;
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';

$s3 = new S3Client([
    'region' => 'us-east-1',
    'version' => 'latest',
]);

// Retrieve the list of buckets.
$result = $s3->listBuckets();

try {
    // Retrieve a paginator for listing objects.
    $objects = $s3->getPaginator('ListObjects', [
        'Bucket' => $bucket
    ]);

    echo "Keys retrieved!" . PHP_EOL;

    // Print the list of objects to the page.
    foreach ($objects as $object) {
        echo $object['Key'] . PHP_EOL;
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}
```

Note

You can create the `S3Client` client without providing your security credentials. Requests sent using this client are anonymous requests, without a signature. Amazon S3 returns an error if you send anonymous requests for a resource that is not publicly available.

For working examples, see [Operations on Objects \(p. 160\)](#). You can test these examples using your AWS account or IAM user credentials.

For an example of listing object keys in a bucket, see [Listing Keys Using the AWS SDK for PHP \(p. 226\)](#).

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Ruby

Before you can use version 3 of the AWS SDK for Ruby to make calls to Amazon S3, you must set the AWS access credentials that the SDK uses to verify your access to your buckets and objects. If you have shared credentials set up in the AWS credentials profile on your local system, version 3 of the SDK for Ruby can use those credentials without your having to declare them in your code. For more information about setting up shared credentials, see [Making Requests Using AWS Account or IAM User Credentials \(p. 19\)](#).

The following Ruby code snippet uses the credentials in a shared AWS credentials file on a local computer to authenticate a request to get all of the object key names in a specific bucket. It does the following:

1. Creates an instance of the `Aws::S3::Resource` class.
2. Makes a request to Amazon S3 by enumerating objects in a bucket using the `bucket` method of `Aws::S3::Resource`. The client generates the necessary signature value from the credentials in the AWS credentials file on your computer, and includes it in the request it sends to Amazon S3.
3. Prints the array of object key names to the terminal.

Example

```
# This snippet example does the following:
# Creates an instance of the Aws::S3::Resource class.
# Makes a request to Amazon S3 by enumerating objects in a bucket using the bucket method
# of Aws::S3::Resource.
# The client generates the necessary signature value from the credentials in the AWS
# credentials file on your computer,
# and includes it in the request it sends to Amazon S3.
# Prints the array of object key names to the terminal.
# The credentials that are used for this example come from a local AWS credentials file on
# the computer that is running this application.
# The credentials are for an IAM user who can list objects in the bucket that the user
# specifies when they run the application.

# Use the Amazon S3 modularized gem for version 3 of the AWS Ruby SDK.
require 'aws-sdk-s3'

# Get an Amazon S3 resource.
s3 = Aws::S3::Resource.new(region: 'us-west-2')

# Create an array of up to the first 100 object keynames in the bucket.
bucket = s3.bucket('example_bucket').objects.collect(&:key)

# Print the array to the terminal.
puts bucket
```

If you don't have a local AWS credentials file, you can still create the `Aws::S3::Resource` resource and execute code against Amazon S3 buckets and objects. Requests that are sent using version 3 of the SDK for Ruby are anonymous, with no signature by default. Amazon S3 returns an error if you send anonymous requests for a resource that's not publicly available.

You can use and expand the previous code snippet for SDK for Ruby applications, as in the following more robust example. The credentials that are used for this example come from a local AWS credentials file on the computer that is running this application. The credentials are for an IAM user who can list objects in the bucket that the user specifies when they run the application.

```
# This snippet example does the following:
# Creates an instance of the Aws::S3::Resource class.
# Makes a request to Amazon S3 by enumerating objects in a bucket using the bucket method
  of Aws::S3::Resource.
# The client generates the necessary signature value from the credentials in the AWS
  credentials file on your computer,
# and includes it in the request it sends to Amazon S3.
# Prints the array of object key names to the terminal.
# The credentials that are used for this example come from a local AWS credentials file on
  the computer that is running this application.
# The credentials are for an IAM user who can list objects in the bucket that the user
  specifies when they run the application.

# Use the Amazon S3 modularized gem for version 3 of the AWS Ruby SDK.
require 'aws-sdk-s3'

# Usage: ruby auth_request_test.rb OPERATION BUCKET
# Currently only the list operation is supported

# The operation to perform on the bucket.
operation = 'list' # default
operation = ARGV[0] if (ARGV.length > 0)

if ARGV.length > 1
  bucket_name = ARGV[1]
else
  exit 1
end

# Get an Amazon S3 resource.
s3 = Aws::S3::Resource.new(region: 'us-west-2')

# Get the bucket by name.
bucket = s3.bucket(bucket_name)

case operation
when 'list'
  if bucket.exists?
    # Enumerate the bucket contents and object etags.
    puts "Contents of '%s':" % bucket_name
    puts '  Name => GUID'

    bucket.objects.limit(50).each do |obj|
      puts "    #{obj.key} => #{obj.etag}"
    end
  else
    puts "The bucket '%s' does not exist!" % bucket_name
  end
else
  puts "Unknown operation: '%s'! Only list is supported." % operation
end
```

Making Requests Using IAM User Temporary Credentials

Topics

- [Making Requests Using IAM User Temporary Credentials - AWS SDK for Java \(p. 26\)](#)
- [Making Requests Using IAM User Temporary Credentials - AWS SDK for .NET \(p. 28\)](#)
- [Making Requests Using AWS Account or IAM User Temporary Credentials - AWS SDK for PHP \(p. 30\)](#)
- [Making Requests Using IAM User Temporary Credentials - AWS SDK for Ruby \(p. 31\)](#)

An AWS Account or an IAM user can request temporary security credentials and use them to send authenticated requests to Amazon S3. This section provides examples of how to use the AWS SDK for Java, .NET, and PHP to obtain temporary security credentials and use them to authenticate your requests to Amazon S3.

Making Requests Using IAM User Temporary Credentials - AWS SDK for Java

An IAM user or an AWS Account can request temporary security credentials (see [Making Requests \(p. 10\)](#)) using the AWS SDK for Java and use them to access Amazon S3. These credentials expire after the specified session duration. To use IAM temporary security credentials, do the following:

1. Create an instance of the `AWSSecurityTokenServiceClient` class. For information about providing credentials, see [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#).
2. Assume the desired role by calling the `assumeRole()` method of the Security Token Service (STS) client.
3. Start a session by calling the `getSessionToken()` method of the STS client. You provide session information to this method using a `GetSessionTokenRequest` object.

The method returns the temporary security credentials.

4. Package the temporary security credentials into a `BasicSessionCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
5. Create an instance of the `AmazonS3Client` class using the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 will return an error.

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary credentials are valid for only one hour. You can specify the session duration only if you use IAM user credentials to request a session.

The following example lists a set of object keys in the specified bucket. The example obtains temporary security credentials for a two-hour session and uses them to send an authenticated request to Amazon S3.

If you want to test the sample using IAM user credentials, you will need to create an IAM user under your AWS Account. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.AWSSessionCredentialsProvider;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetSessionTokenRequest;
import com.amazonaws.services.securitytoken.model.GetSessionTokenResult;

public class MakingRequestsWithIAMTempCredentials {
    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";

        try {
            // Creating the STS client is part of your trusted code. It has
            // the security credentials you use to obtain temporary security credentials.
            AWSSecurityTokenService stsClient =
                AWSSecurityTokenServiceClientBuilder.standard()
                    .withCredentials(new ProfileCredentialsProvider())
                    .withRegion(clientRegion)
                    .build();

            // Start a session.
            GetSessionTokenRequest getSessionTokenRequest = new
                GetSessionTokenRequest().withDurationSeconds(7200);
            // The duration can be set to more than 3600 seconds only if temporary
            // credentials are requested by an IAM user rather than an account owner.
            GetSessionTokenResult sessionTokenResult =
                stsClient.getSessionToken(getSessionTokenRequest);
            Credentials sessionCredentials = sessionTokenResult
                .getCredentials();

            .withSessionToken(sessionTokenResult.getCredentials().getSessionToken())
                .withExpiration(sessionTokenResult.getCredentials().getExpiration());

            // Package the temporary security credentials as a BasicSessionCredentials
            object
            // for an Amazon S3 client object to use.
            BasicSessionCredentials basicSessionCredentials = new BasicSessionCredentials(
                sessionCredentials.getAccessKeyId(),
                sessionCredentials.getSecretAccessKey(),
                sessionCredentials.getSessionToken());

            // Provide temporary security credentials so that the Amazon S3 client
            // can send authenticated requests to Amazon S3. You create the client
            // using the basicSessionCredentials object.
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new
                AWSSessionCredentialsProvider(basicSessionCredentials))
                .withRegion(clientRegion)
                .build();

            // Verify that getting the session token worked and the permissions are set
            correctly
            // by getting a set of object keys from the bucket.
            ObjectListing objects = s3Client.listObjects(bucketName);
            System.out.println("No. of Objects: " + objects.getObjectSummaries().size());
```

```
        } catch (AmazonServiceException e) {  
            // The call was transmitted successfully, but Amazon S3 couldn't process  
            // it, so it returned an error response.  
            e.printStackTrace();  
        } catch (SdkClientException e) {  
            // Amazon S3 couldn't be contacted for a response, or the client  
            // couldn't parse the response from Amazon S3.  
            e.printStackTrace();  
        }  
    }  
}
```

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Making Requests Using IAM User Temporary Credentials - AWS SDK for .NET

An IAM user or an AWS account can request temporary security credentials using the AWS SDK for .NET and use them to access Amazon S3. These credentials expire after the session duration. To get temporary security credentials and access Amazon S3, do the following:

1. Create an instance of the AWS Security Token Service client, `AmazonSecurityTokenServiceClient`. For information about providing credentials, see [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#).
2. Start a session by calling the `GetSessionToken` method of the STS client you created in the preceding step. You provide session information to this method using a `GetSessionTokenRequest` object.

The method returns your temporary security credentials.

3. Package the temporary security credentials in an instance of the `SessionAWSCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
4. Create an instance of the `AmazonS3Client` class by passing in the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

Note

If you obtain temporary security credentials using your AWS account security credentials, those credentials are valid for only one hour. You can specify a session duration only if you use IAM user credentials to request a session.

The following C# example lists object keys in the specified bucket. For illustration, the example obtains temporary security credentials for a default one-hour session and uses them to send authenticated request to Amazon S3.

If you want to test the sample using IAM user credentials, you need to create an IAM user under your AWS account. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*. For more information about making requests, see [Making Requests \(p. 10\)](#).

For instructions on creating and testing a working example, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class TempCredExplicitSessionStartTest
    {
        private const string bucketName = "**** bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        public static void Main()
        {
            ListObjectsAsync().Wait();
        }

        private static async Task ListObjectsAsync()
        {
            try
            {
                // Credentials use the default AWS SDK for .NET credential search chain.
                // On local development machines, this is your default profile.
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials = await
GetTemporaryCredentialsAsync();

                // Create a client by providing temporary security credentials.
                using (s3Client = new AmazonS3Client(tempCredentials, bucketRegion))
                {
                    var listObjectRequest = new ListObjectsRequest
                    {
                        BucketName = bucketName
                    };
                    // Send request to Amazon S3.
                    ListObjectsResponse response = await
s3Client.ListObjectsAsync(listObjectRequest);
                    List<S3Object> objects = response.S3Objects;
                    Console.WriteLine("Object count = {0}", objects.Count);
                }
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message, s3Exception.InnerException);
            }
            catch (AmazonSecurityTokenServiceException stsException)
            {
                Console.WriteLine(stsException.Message, stsException.InnerException);
            }
        }

        private static async Task<SessionAWSCredentials> GetTemporaryCredentialsAsync()
        {
            using (var stsClient = new AmazonSecurityTokenServiceClient())
            {
                var getSessionTokenRequest = new GetSessionTokenRequest
                {
                    DurationSeconds = 7200 // seconds
                };

                GetSessionTokenResponse sessionTokenResponse =

```

```
        await stsClient.GetSessionTokenAsync(getSessionTokenRequest);

        Credentials credentials = sessionTokenResponse.Credentials;

        var sessionCredentials =
            new SessionAWSCredentials(credentials.AccessKeyId,
                                       credentials.SecretAccessKey,
                                       credentials.SessionToken);

        return sessionCredentials;
    }
}
}
```

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Making Requests Using AWS Account or IAM User Temporary Credentials - AWS SDK for PHP

This topic guides explains how to use classes from version 3 of the AWS SDK for PHP to request temporary security credentials and use them to access Amazon S3. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

An IAM user or an AWS account can request temporary security credentials using version 3 of the AWS SDK for PHP. It can then use the temporary credentials to access Amazon S3. The credentials expire when the session duration expires. By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration (from 1 to 36 hours) when requesting the temporary security credentials. For more information about temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*. For more information about making requests, see [Making Requests \(p. 10\)](#).

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary security credentials are valid for only one hour. You can specify the session duration only if you use IAM user credentials to request a session.

Example

The following PHP example lists object keys in the specified bucket using temporary security credentials. The example obtains temporary security credentials for a default one-hour session, and uses them to send authenticated request to Amazon S3. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

If you want to test the example using IAM user credentials, you need to create an IAM user under your AWS account. For information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*. For an example of setting the session duration when using IAM user credentials to request a session, see [Making Requests Using Federated User Temporary Credentials - AWS SDK for PHP \(p. 39\)](#).

```
require 'vendor/autoload.php';

use Aws\Sts\StsClient;
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
```



```
$sts = new StsClient([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

$sessionToken = $sts->getSessionToken();

$s3 = new S3Client([
    'region' => 'us-east-1',
    'version' => 'latest',
    'credentials' => [
        'key' => $sessionToken['Credentials']['AccessKeyId'],
        'secret' => $sessionToken['Credentials']['SecretAccessKey'],
        'token' => $sessionToken['Credentials']['SessionToken']
    ]
]);

$result = $s3->listBuckets();

try {
    // Retrieve a paginator for listing objects.
    $objects = $s3->getPaginator('ListObjects', [
        'Bucket' => $bucket
    ]);

    echo "Keys retrieved!" . PHP_EOL;

    // List objects
    foreach ($objects as $object) {
        echo $object['Key'] . PHP_EOL;
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Making Requests Using IAM User Temporary Credentials - AWS SDK for Ruby

An IAM user or an AWS account can request temporary security credentials using AWS SDK for Ruby and use them to access Amazon S3. These credentials expire after the session duration. By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration (from 1 to 36 hours) when requesting the temporary security credentials. For information about requesting temporary security credentials, see [Making Requests \(p. 10\)](#).

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary security credentials are valid for only one hour. You can specify session duration only if you use IAM user credentials to request a session.

The following Ruby example creates a temporary user to list the items in a specified bucket for one hour. To use this example, you must have AWS credentials that have the necessary permissions to create new AWS Security Token Service (AWS STS) clients, and list Amazon S3 buckets.

```
# This snippet example does the following:
```

```
# The following Ruby example creates a temporary user to list the items in a specified
# bucket
# for one hour. To use this example, you must have AWS credentials that have the necessary
# permissions to create new AWS Security Token Service (AWS STS) clients, and list Amazon
# S3 buckets using temporary security credentials
# using your AWS account security credentials, the temporary security credentials are valid
# for only one hour. You can
# specify session duration only if you use &IAM; user credentials to request a session.

require 'aws-sdk-core'
require 'aws-sdk-s3'
require 'aws-sdk-iam'

USAGE = <<DOC

Usage: assumeroles_create_bucket_policy.rb -b BUCKET -u USER [-r REGION] [-d] [-h]

    Assumes a role for USER to list items in BUCKET for one hour.

    BUCKET is required and must already exist.

    USER is required and if not found, is created.

    If REGION is not supplied, defaults to us-west-2.

    -d gives you extra (debugging) information.

    -h displays this message and quits.

DOC

def print_debug(debug, s)
  if debug
    puts s
  end
end

# Get the user if they exist, otherwise create them
def get_user(region, user_name, debug)
  iam = Aws::IAM::Resource.new(region: region)

  # See if user exists
  user = iam.user(user_name)

  # If user does not exist, create them
  if user == nil
    user = iam.create_user(user_name: user_name)
    iam.wait_until(:user_exists, user_name: user_name)
    print_debug(debug, "Created new user #{user_name}")
  else
    print_debug(debug, "Found user #{user_name} in region #{region}")
  end

  user
end

# main
region = 'us-west-2'
user_name = ''
bucket_name = ''

i = 0

while i < ARGV.length
  case ARGV[i]
```

```
when '-b'
  i += 1
  bucket_name = ARGV[i]

when '-u'
  i += 1
  user_name = ARGV[i]

when '-r'
  i += 1

  region = ARGV[i]

when '-h'
  puts USAGE
  exit 0

else
  puts 'Unrecognized option: ' + ARGV[i]
  puts USAGE
  exit 1

end

i += 1
end

if bucket_name == ''
  puts 'You must supply a bucket name'
  puts USAGE
  exit 1
end

if user_name == ''
  puts 'You must supply a user name'
  puts USAGE
  exit 1
end

# Create a new Amazon STS client and get temporary credentials. This uses a role that was
# already created.
begin
  creds = Aws::AssumeRoleCredentials.new(
    client: Aws::STS::Client.new(region: region),
    role_arn: "arn:aws:iam::111122223333:role/assumedrolelist",
    role_session_name: "assumerole-s3-list"
  )

  # Create an Amazon S3 resource with temporary credentials.
  s3 = Aws::S3::Resource.new(region: region, credentials: creds)

  puts "Contents of '%s':" % bucket_name
  puts '  Name => GUID'

  s3.bucket(bucket_name).objects.limit(50).each do |obj|
    puts "  #{obj.key} => #{obj.etag}"
  end
rescue StandardError => ex
  puts 'Caught exception accessing bucket ' + bucket_name + ':'
  puts ex.message
end
```

Making Requests Using Federated User Temporary Credentials

You can request temporary security credentials and provide them to your federated users or applications who need to access your AWS resources. This section provides examples of how you can use the AWS SDK to obtain temporary security credentials for your federated users or applications and send authenticated requests to Amazon S3 using those credentials. For a list of available AWS SDKs, see [Sample Code and Libraries](#).

Note

Both the AWS account and an IAM user can request temporary security credentials for federated users. However, for added security, only an IAM user with the necessary permissions should request these temporary credentials to ensure that the federated user gets at most the permissions of the requesting IAM user. In some applications, you might find it suitable to create an IAM user with specific permissions for the sole purpose of granting temporary security credentials to your federated users and applications.

Making Requests Using Federated User Temporary Credentials - AWS SDK for Java

You can provide temporary security credentials for your federated users and applications so that they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. By default, the session duration is one hour. You can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications.

Note

For added security when requesting temporary security credentials for federated users and applications, we recommend that you use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, see [AWS Identity and Access Management FAQs](#).

To provide security credentials and send authenticated request to access resources, do the following:

- Create an instance of the `AWSSecurityTokenServiceClient` class. For information about providing credentials, see [Using the AWS SDK for Java \(p. 676\)](#).
- Start a session by calling the `getFederationToken()` method of the Security Token Service (STS) client. Provide session information, including the user name and an IAM policy, that you want to attach to the temporary credentials. You can provide an optional session duration. This method returns your temporary security credentials.
- Package the temporary security credentials in an instance of the `BasicSessionCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
- Create an instance of the `AmazonS3Client` class using the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

Example

The example lists keys in the specified S3 bucket. In the example, you obtain temporary security credentials for a two-hour session for your federated user and use the credentials to send authenticated requests to Amazon S3. To run the example, you need to create an IAM user with an attached policy that allows the user to request temporary security credentials and list your AWS resources. The following policy accomplishes this:

```
{
  "Statement": [{
    "Action": ["s3:ListBucket",
      "sts:GetFederationToken*"],
    "Effect": "Allow",
    "Resource": "*"
  }]
}
```

For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

After creating an IAM user and attaching the preceding policy, you can run the following example. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.AWSSessionCredentialsProvider;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.S3Actions;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;

import java.io.IOException;

public class MakingRequestsWithFederatedTempCredentials {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Specify bucket name ****";
        String federatedUser = "**** Federated user name ****";
        String resourceARN = "arn:aws:s3:::" + bucketName;

        try {
            AWSSecurityTokenService stsClient = AWSSecurityTokenServiceClientBuilder
                .standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            GetFederationTokenRequest getFederationTokenRequest = new
            GetFederationTokenRequest();
            getFederationTokenRequest.setDurationSeconds(7200);
            getFederationTokenRequest.setName(federatedUser);

            // Define the policy and add it to the request.
            Policy policy = new Policy();
```

```
policy.withStatements(new Statement(Effect.Allow)
    .withActions(S3Actions.ListObjects)
    .withResources(new Resource(resourceARN)));
getFederationTokenRequest.setPolicy(policy.toJson());

// Get the temporary security credentials.
GetFederationTokenResult federationTokenResult =
stsClient.getFederationToken(getFederationTokenRequest);
Credentials sessionCredentials = federationTokenResult.getCredentials();

// Package the session credentials as a BasicSessionCredentials
// object for an Amazon S3 client object to use.
BasicSessionCredentials basicSessionCredentials = new BasicSessionCredentials(
    sessionCredentials.getAccessKeyId(),
    sessionCredentials.getSecretAccessKey(),
    sessionCredentials.getSessionToken());
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
    .withCredentials(new
AWSStaticCredentialsProvider(basicSessionCredentials))
    .withRegion(clientRegion)
    .build();

// To verify that the client works, send a listObjects request using
// the temporary security credentials.
ObjectListing objects = s3Client.listObjects(bucketName);
System.out.println("No. of Objects = " + objects.getObjectSummaries().size());
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Making Requests Using Federated User Temporary Credentials - AWS SDK for .NET

You can provide temporary security credentials for your federated users and applications so that they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. By default, the duration of a session is one hour. You can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications. For information about sending authenticated requests, see [Making Requests \(p. 10\)](#).

Note

When requesting temporary security credentials for federated users and applications, for added security, we suggest that you use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, see [AWS Identity and Access Management FAQs](#).

You do the following:

- Create an instance of the AWS Security Token Service client, `AmazonSecurityTokenServiceClient` class. For information about providing credentials, see [Using the AWS SDK for .NET \(p. 677\)](#).
- Start a session by calling the `GetFederationToken` method of the STS client. You need to provide session information, including the user name and an IAM policy that you want to attach to the temporary credentials. Optionally, you can provide a session duration. This method returns your temporary security credentials.
- Package the temporary security credentials in an instance of the `SessionAWSCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
- Create an instance of the `AmazonS3Client` class by passing the temporary security credentials. You use this client to send requests to Amazon S3. If you send requests using expired credentials, Amazon S3 returns an error.

Example

The following C# example lists the keys in the specified bucket. In the example, you obtain temporary security credentials for a two-hour session for your federated user (User1), and use the credentials to send authenticated requests to Amazon S3.

- For this exercise, you create an IAM user with minimal permissions. Using the credentials of this IAM user, you request temporary credentials for others. This example lists only the objects in a specific bucket. Create an IAM user with the following policy attached:

```
{
  "Statement": [{
    "Action": [ "s3:ListBucket",
      "sts:GetFederationToken" ],
    "Effect": "Allow",
    "Resource": "*"
  }]
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

- Use the IAM user security credentials to test the following example. The example sends authenticated request to Amazon S3 using temporary security credentials. The example specifies the following policy when requesting temporary security credentials for the federated user (User1), which restricts access to listing objects in a specific bucket (`YourBucketName`). You must update the policy and provide your own existing bucket name.

```
{
  "Statement": [
    {
      "Sid": "1",
      "Action": [ "s3:ListBucket" ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::YourBucketName"
    }
  ]
}
```

- **Example**

Update the following sample and provide the bucket name that you specified in the preceding federated user access policy. For instructions on how to create and test a working example, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class TempFederatedCredentialsTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth1;
        private static IAmazonS3 client;

        public static void Main()
        {
            ListObjectsAsync().Wait();
        }

        private static async Task ListObjectsAsync()
        {
            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                // Credentials use the default AWS SDK for .NET credential search chain.
                // On local development machines, this is your default profile.
                SessionAWSCredentials tempCredentials =
                    await GetTemporaryFederatedCredentialsAsync();

                // Create a client by providing temporary security credentials.
                using (client = new AmazonS3Client(bucketRegion))
                {
                    ListObjectsRequest listObjectRequest = new ListObjectsRequest();
                    listObjectRequest.BucketName = bucketName;

                    ListObjectsResponse response = await
client.ListObjectsAsync(listObjectRequest);
                    List<S3Object> objects = response.S3Objects;
                    Console.WriteLine("Object count = {0}", objects.Count);

                    Console.WriteLine("Press any key to continue...");
                    Console.ReadKey();
                }
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
        }
    }
}
```



```

    }
}

private static async Task<SessionAWSCredentials>
GetTemporaryFederatedCredentialsAsync()
{
    AmazonSecurityTokenServiceConfig config = new
AmazonSecurityTokenServiceConfig();
    AmazonSecurityTokenServiceClient stsClient =
        new AmazonSecurityTokenServiceClient(
            config);

    GetFederationTokenRequest federationTokenRequest =
        new GetFederationTokenRequest();
    federationTokenRequest.DurationSeconds = 7200;
    federationTokenRequest.Name = "User1";
    federationTokenRequest.Policy = @"{
        "Statement":
        [
            {
                "Sid": "Stmt1311212314284",
                "Action": [ "s3:ListBucket" ],
                "Effect": "Allow",
                "Resource": "arn:aws:s3::" + bucketName + @""
            }
        ]
    }";

    GetFederationTokenResponse federationTokenResponse =
        await stsClient.GetFederationTokenAsync(federationTokenRequest);
    Credentials credentials = federationTokenResponse.Credentials;

    SessionAWSCredentials sessionCredentials =
        new SessionAWSCredentials(credentials.AccessKeyId,
            credentials.SecretAccessKey,
            credentials.SessionToken);

    return sessionCredentials;
}
}
}

```

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Making Requests Using Federated User Temporary Credentials - AWS SDK for PHP

This topic explains how to use classes from version 3 of the AWS SDK for PHP to request temporary security credentials for federated users and applications and use them to access resources stored in Amazon S3. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

You can provide temporary security credentials to your federated users and applications so they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. These credentials expire when the session duration expires. By default, the session duration is one hour. You can explicitly set a different value for the duration when requesting the temporary security credentials for federated users and applications. For more information about temporary security

credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*. For information about providing temporary security credentials to your federated users and applications, see [Making Requests \(p. 10\)](#).

For added security when requesting temporary security credentials for federated users and applications, we recommend using a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For information about identity federation, see [AWS Identity and Access Management FAQs](#).

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

Example

The following PHP example lists keys in the specified bucket. In the example, you obtain temporary security credentials for an hour session for your federated user (User1). Then you use the temporary security credentials to send authenticated requests to Amazon S3.

For added security when requesting temporary credentials for others, you use the security credentials of an IAM user who has permissions to request temporary security credentials. To ensure that the IAM user grants only the minimum application-specific permissions to the federated user, you can also limit the access permissions of this IAM user. This example lists only objects in a specific bucket. Create an IAM user with the following policy attached:

```
{
  "Statement": [{
    "Action": ["s3:ListBucket",
      "sts:GetFederationToken*"],
    "Effect": "Allow",
    "Resource": "*"
  }]
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

You can now use the IAM user security credentials to test the following example. The example sends an authenticated request to Amazon S3 using temporary security credentials. When requesting temporary security credentials for the federated user (User1), the example specifies the following policy, which restricts access to list objects in a specific bucket. Update the policy with your bucket name.

```
{
  "Statement": [
    {
      "Sid": "1",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::YourBucketName"
    }
  ]
}
```

In the following example, when specifying the policy resource, replace `YourBucketName` with the name of your bucket.:

```
require 'vendor/autoload.php';

use Aws\Sts\StsClient;
use Aws\S3\S3Client;
```

```
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';

// In real applications, the following code is part of your trusted code. It has
// the security credentials that you use to obtain temporary security credentials.
$sts = new StsClient(
    [
        'version' => 'latest',
        'region' => 'us-east-1']
);

// Fetch the federated credentials.
$sessionToken = $sts->getFederationToken([
    'Name' => 'User1',
    'DurationSeconds' => '3600',
    'Policy' => json_encode([
        'Statement' => [
            'Sid' => 'randomstatementid' . time(),
            'Action' => ['s3:ListBucket'],
            'Effect' => 'Allow',
            'Resource' => 'arn:aws:s3:::' . $bucket
        ]
    ])
]);

// The following will be part of your less trusted code. You provide temporary
// security credentials so the code can send authenticated requests to Amazon S3.

$s3 = new S3Client([
    'region' => 'us-east-1',
    'version' => 'latest',
    'credentials' => [
        'key' => $sessionToken['Credentials']['AccessKeyId'],
        'secret' => $sessionToken['Credentials']['SecretAccessKey'],
        'token' => $sessionToken['Credentials']['SessionToken']
    ]
]);

try {
    $result = $s3->listObjects([
        'Bucket' => $bucket
    ]);
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Making Requests Using Federated User Temporary Credentials - AWS SDK for Ruby

You can provide temporary security credentials for your federated users and applications so that they can send authenticated requests to access your AWS resources. When requesting temporary credentials from the IAM service, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. By default, the session duration is one hour. However, if you are requesting temporary credentials using IAM user credentials, you can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications. For information about temporary security credentials for your federated users and applications, see [Making Requests \(p. 10\)](#).

Note

For added security when you request temporary security credentials for federated users and applications, you might want to use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, see [AWS Identity and Access Management FAQs](#).

Example

The following Ruby code example allows a federated user with a limited set of permissions to lists keys in the specified bucket.

```
require 'aws-sdk-s3'
require 'aws-sdk-iam'

USAGE = <<DOC

Usage: ruby auth_federation_token_request_test.rb -b BUCKET -u USER [-r REGION] [-d] [-h]

  Creates a federated policy for USER to list items in BUCKET for one hour.

  BUCKET is required and must already exist.

  USER is required and if not found, is created.

  If REGION is not supplied, defaults to us-west-2.

  -d gives you extra (debugging) information.

  -h displays this message and quits.

DOC

def print_debug(debug, s)
  if debug
    puts s
  end
end

# Get the user if they exist, otherwise create them
def get_user(region, user_name, debug)
  iam = Aws::IAM::Client.new(region: 'us-west-2')

  # See if user exists
  user = iam.user(user_name)

  # If user does not exist, create them
  if user == nil
    user = iam.create_user(user_name: user_name)
    iam.wait_until(:user_exists, user_name: user_name)
  end
end
```

```
    print_debug(debug, "Created new user #{user_name}")
  else
    print_debug(debug, "Found user #{user_name} in region #{region}")
  end

  user
end

# main
region = 'us-west-2'
user_name = ''
bucket_name = ''

i = 0

while i < ARGV.length
  case ARGV[i]

    when '-b'
      i += 1
      bucket_name = ARGV[i]

    when '-u'
      i += 1
      user_name = ARGV[i]

    when '-r'
      i += 1
      region = ARGV[i]

    when '-h'
      puts USAGE
      exit 0

    else
      puts 'Unrecognized option: ' + ARGV[i]
      puts USAGE
      exit 1

  end

  i += 1
end

if bucket_name == ''
  puts 'You must supply a bucket name'
  puts USAGE
  exit 1
end

if user_name == ''
  puts 'You must supply a user name'
  puts USAGE
  exit 1
end

# Create a new STS client and get temporary credentials.
sts = Aws::STS::Client.new(region: region)

creds = sts.get_federation_token({
  duration_seconds: 3600,
  name: user_name,
  policy: "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Sid\":\"Stmt1\",\"Effect\":\"Allow\", \"Action\":\"s3:ListBucket\", \"Resource\":\"arn:aws:s3::#{bucket_name}\"}]}",
})
```

```
# Create an Amazon S3 resource with temporary credentials.
s3 = Aws::S3::Resource.new(region: region, credentials: creds)

puts "Contents of '%s':" % bucket_name
puts '  Name => GUID'

begin
  s3.bucket(bucket_name).objects.limit(50).each do |obj|
    puts "    #{obj.key} => #{obj.etag}"
  end
rescue StandardError => ex
  puts 'Caught exception accessing bucket ' + bucket_name + ':'
  puts ex.message
end
```

Making Requests Using the REST API

This section contains information on how to make requests to Amazon S3 endpoints by using the REST API. For a list of Amazon S3 endpoints, see [Regions and Endpoints](#) in the *AWS General Reference*.

Topics

- [Making Requests to Dual-Stack Endpoints by Using the REST API \(p. 45\)](#)
- [Virtual Hosting of Buckets \(p. 45\)](#)
- [Request Redirection and the REST API \(p. 50\)](#)

When making requests by using the REST API, you can use virtual hosted-style or path-style URIs for the Amazon S3 endpoints. For more information, see [Working with Amazon S3 Buckets \(p. 53\)](#).

Example Virtual Hosted-Style Request

Following is an example of a virtual hosted-style request to delete the `puppy.jpg` file from the bucket named `examplebucket`.

```
DELETE /puppy.jpg HTTP/1.1
Host: examplebucket.s3.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

Example Path-Style Request

Following is an example of a path-style version of the same request.

```
DELETE /examplebucket/puppy.jpg HTTP/1.1
Host: s3-us-west-2.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

Currently Amazon S3 supports virtual hosted-style and path-style access in all Regions but this will be changing (see the following **Important** note.) The path-style syntax requires that you use the Region-specific endpoint when attempting to access a bucket. For example, if you have a bucket called `mybucket` that resides in the EU (Ireland) Region, you want to use path-style syntax, and the object is named `puppy.jpg`, the correct URI is `http://s3-eu-west-1.amazonaws.com/mybucket/puppy.jpg`.

You will receive an HTTP response code 307 Temporary Redirect error and a message indicating what the correct URI is for your resource if you try to access a bucket outside the US East (N. Virginia) Region with path-style syntax that uses either of the following:

- `http://s3.amazonaws.com`
- An endpoint for a Region different from the one where the bucket resides. For example, if you use `http://s3-us-west-1.amazonaws.com` for a bucket that was created in the US West (N. California) Region.

Important

Buckets created after September 30, 2020, will support only virtual hosted-style requests. Path-style requests will continue to be supported for buckets created on or before this date. For more information, see [Amazon S3 Path Deprecation Plan – The Rest of the Story](#).

Making Requests to Dual-Stack Endpoints by Using the REST API

When using the REST API, you can directly access a dual-stack endpoint by using a virtual hosted-style or a path style endpoint name (URI). All Amazon S3 dual-stack endpoint names include the region in the name. Unlike the standard IPv4-only endpoints, both virtual hosted-style and a path-style endpoints use region-specific endpoint names.

Example Virtual Hosted-Style Dual-Stack Endpoint Request

You can use a virtual hosted-style endpoint in your REST request as shown in the following example that retrieves the `puppy.jpg` object from the bucket named `examplebucket`.

```
GET /puppy.jpg HTTP/1.1
Host: examplebucket.s3.dualstack.us-west-2.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

Example Path-Style Dual-Stack Endpoint Request

Or you can use a path-style endpoint in your request as shown in the following example.

```
GET /examplebucket/puppy.jpg HTTP/1.1
Host: s3.dualstack.us-west-2.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

For more information about dual-stack endpoints, see [Using Amazon S3 Dual-Stack Endpoints \(p. 14\)](#).

Virtual Hosting of Buckets

Topics

- [HTTP Host Header Bucket Specification \(p. 46\)](#)
- [Examples \(p. 47\)](#)
- [Customizing Amazon S3 URLs with CNAMEs \(p. 48\)](#)
- [Limitations \(p. 49\)](#)
- [Backward Compatibility \(p. 50\)](#)

In general, virtual hosting is the practice of serving multiple websites from a single web server. One way to differentiate sites is by using the apparent hostname of the request instead of just the path name part of the URI. An ordinary Amazon S3 REST request specifies a bucket by using the first slash-delimited component of the Request-URI path. Or, you can use Amazon S3 virtual hosting to address a bucket in a REST API call by using the HTTP `Host` header. In practice, Amazon S3 interprets `Host` as meaning that most buckets are automatically accessible (for limited types of requests) at `http://bucketname.s3.amazonaws.com`. Furthermore, by naming your bucket after your registered domain name and by making that name a DNS alias for Amazon S3, you can completely customize the URL of your Amazon S3 resources, for example, `http://my.bucketname.com/`.

Besides the attractiveness of customized URLs, a second benefit of virtual hosting is the ability to publish to the "root directory" of your bucket's virtual server. This ability can be important because many existing applications search for files in this standard location. For example, `favicon.ico`, `robots.txt`, `crossdomain.xml` are all expected to be found at the root.

Currently Amazon S3 supports virtual hosted-style and path-style access in all Regions but this will be changing (see the following **Important** note.) The path-style syntax requires that you use the Region-specific endpoint when attempting to access a bucket. For example, if you have a bucket called `mybucket` that resides in the EU (Ireland) Region, you want to use path-style syntax, and the object is named `puppy.jpg`, the correct URI is `http://s3-eu-west-1.amazonaws.com/mybucket/puppy.jpg`.

You will receive an HTTP response code 307 Temporary Redirect error and a message indicating what the correct URI is for your resource if you try to access a bucket outside the US East (N. Virginia) Region with path-style syntax that uses either of the following:

- `http://s3.amazonaws.com`
- An endpoint for a Region different from the one where the bucket resides. For example, if you use `http://s3-us-west-1.amazonaws.com` for a bucket that was created in the US West (N. California) Region.

Important

Buckets created after September 30, 2020, will support only virtual hosted-style requests. Path-style requests will continue to be supported for buckets created on or before this date. For more information, see [Amazon S3 Path Deprecation Plan – The Rest of the Story](#).

If you use the US East (N. Virginia) endpoint (`s3.amazonaws.com`) instead of the Region-specific endpoint (for example, `s3-us-west-1.amazonaws.com`), Amazon S3 routes any virtual hosted-style requests to the US East (N. Virginia) Region by default. When you create a bucket in any Region that was launched before March 20, 2019, Amazon S3 updates the DNS to reroute the request to the correct location, which might take time. In the meantime, the default rule applies and your virtual hosted-style request goes to the US East (N. Virginia) Region. Amazon S3 then redirects it with an HTTP 307 redirect to the correct Region.

For S3 buckets in Regions launched after March 20, 2019, the DNS doesn't route your request directly to the AWS Region where your bucket resides. It returns an HTTP 400 Bad Request error instead.

For more information, see [Request Redirection and the REST API \(p. 599\)](#).

When using virtual hosted-style buckets with SSL, the SSL wild-card certificate only matches buckets that do not contain periods. To work around this, use HTTP or write your own certificate verification logic.

HTTP Host Header Bucket Specification

As long as your `GET` request does not use the SSL endpoint, you can specify the bucket for the request by using the HTTP `Host` header. The `Host` header in a REST request is interpreted as follows:

- If the `Host` header is omitted or its value is `s3.amazonaws.com`, the bucket for the request will be the first slash-delimited component of the Request-URI, and the key for the request will be the rest of the Request-URI. This is the ordinary method, as illustrated by the first and second examples in this section. Omitting the `Host` header is valid only for HTTP 1.0 requests.
- Otherwise, if the value of the `Host` header ends in `.s3.amazonaws.com`, the bucket name is the leading component of the `Host` header's value up to `.s3.amazonaws.com`. The key for the request is the Request-URI. This interpretation exposes buckets as subdomains of `s3.amazonaws.com`, as illustrated by the third and fourth examples in this section.
- Otherwise, the bucket for the request is the lowercase value of the `Host` header, and the key for the request is the Request-URI. This interpretation is useful when you have registered the same DNS name as your bucket name and have configured that name to be a CNAME alias for Amazon S3. The procedure for registering domain names and configuring DNS is beyond the scope of this guide, but the result is illustrated by the final example in this section.

Examples

This section provides example URLs and requests.

Example Path Style Method

This example uses `johnsmith.net` as the bucket name and `homepage.html` as the key name.

The URL is as follows:

```
http://s3.amazonaws.com/johnsmith.net/homepage.html
```

The request is as follows:

```
GET /johnsmith.net/homepage.html HTTP/1.1
Host: s3.amazonaws.com
```

The request with HTTP 1.0 and omitting the `host` header is as follows:

```
GET /johnsmith.net/homepage.html HTTP/1.0
```

For information about DNS-compatible names, see [Limitations \(p. 49\)](#). For more information about keys, see [Keys \(p. 3\)](#).

Example Virtual Hosted-Style Method

This example uses `johnsmith.net` as the bucket name and `homepage.html` as the key name.

The URL is as follows:

```
http://johnsmith.net.s3.amazonaws.com/homepage.html
```

The request is as follows:

```
GET /homepage.html HTTP/1.1
Host: johnsmith.net.s3.amazonaws.com
```

The virtual hosted-style method requires the bucket name to be DNS-compliant.

Example Virtual Hosted–Style Method for a Bucket in a Region Other Than US East (N. Virginia) Region

This example uses `johnsmith.eu` as the name for a bucket in the EU (Ireland) Region and `homepage.html` as the key name.

The URL is as follows:

```
http://johnsmith.eu.s3-eu-west-1.amazonaws.com/homepage.html
```

The request is as follows:

```
GET /homepage.html HTTP/1.1
Host: johnsmith.eu.s3-eu-west-1.amazonaws.com
```

Instead of using the Region-specific endpoint, you can also use the US East (N. Virginia) Region endpoint no matter what Region the bucket resides in.

```
http://johnsmith.eu.s3.amazonaws.com/homepage.html
```

The request is as follows:

```
GET /homepage.html HTTP/1.1
Host: johnsmith.eu.s3.amazonaws.com
```

Example CNAME Method

This example uses `www.johnsmith.net` as the bucket name and `homepage.html` as the key name. To use this method, you must configure your DNS name as a CNAME alias for `bucketname.s3.amazonaws.com`.

The URL is as follows:

```
http://www.johnsmith.net/homepage.html
```

The example is as follows:

```
GET /homepage.html HTTP/1.1
Host: www.johnsmith.net
```

Customizing Amazon S3 URLs with CNAMEs

Depending on your needs, you might not want `s3.amazonaws.com` to appear on your website or service. For example, if you host your website images on Amazon S3, you might prefer `http://images.johnsmith.net/` instead of `http://johnsmith-images.s3.amazonaws.com/`.

The bucket name must be the same as the CNAME. So `http://images.johnsmith.net/filename` would be the same as `http://images.johnsmith.net.s3.amazonaws.com/filename` if a CNAME were created to map `images.johnsmith.net` to `images.johnsmith.net.s3.amazonaws.com`.

Any bucket with a DNS-compatible name can be referenced as follows: `http://[BucketName].s3.amazonaws.com/[Filename]`, for example, `http://images.johnsmith.net.s3.amazonaws.com/mydog.jpg`. By using CNAME, you can map `images.johnsmith.net` to an Amazon S3 hostname so that the previous URL could become `http://images.johnsmith.net/mydog.jpg`.

The CNAME DNS record should alias your domain name to the appropriate virtual hosted-style hostname. For example, if your bucket name and domain name are `images.johnsmith.net`, the CNAME record should alias to `images.johnsmith.net.s3.amazonaws.com`.

```
images.johnsmith.net CNAME images.johnsmith.net.s3.amazonaws.com.
```

Setting the alias target to `s3.amazonaws.com` also works, but it may result in extra HTTP redirects.

Amazon S3 uses the hostname to determine the bucket name. For example, suppose that you have configured `www.example.com` as a CNAME for `www.example.com.s3.amazonaws.com`. When you access `http://www.example.com`, Amazon S3 receives a request similar to the following:

Example

```
GET / HTTP/1.1
Host: www.example.com
Date: date
Authorization: signatureValue
```

Amazon S3 sees only the original hostname `www.example.com` and is unaware of the CNAME mapping used to resolve the request. So the CNAME and the bucket name must be the same.

Any Amazon S3 endpoint can be used in a CNAME. For example, `s3-ap-southeast-1.amazonaws.com` can be used in CNAMEs. For more information about endpoints, see [Request Endpoints \(p. 11\)](#).

To associate a hostname with an Amazon S3 bucket using CNAMEs

1. Select a hostname that belongs to a domain you control. This example uses the `images` subdomain of the `johnsmith.net` domain.
2. Create a bucket that matches the hostname. In this example, the host and bucket names are `images.johnsmith.net`.

Note

The bucket name must exactly match the hostname.

3. Create a CNAME record that defines the hostname as an alias for the Amazon S3 bucket. For example:

```
images.johnsmith.net CNAME images.johnsmith.net.s3.amazonaws.com
```

Important

For request routing reasons, the CNAME record must be defined exactly as shown in the preceding example. Otherwise, it might appear to operate correctly but eventually results in unpredictable behavior.

Note

The procedure for configuring DNS depends on your DNS server or DNS provider. For specific information, see your server documentation or contact your provider.

Limitations

Virtual host URLs are supported for non-SSL (HTTP) requests only.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Backward Compatibility

Early versions of Amazon S3 incorrectly ignored the HTTP `Host` header. Applications that depend on this undocumented behavior must be updated to set the `Host` header correctly. Because Amazon S3 determines the bucket name from `Host` when it is present, the most likely symptom of this problem is to receive an unexpected `NoSuchBucket` error result code.

Request Redirection and the REST API

Topics

- [Redirects and HTTP User-Agents \(p. 50\)](#)
- [Redirects and 100-Continue \(p. 50\)](#)
- [Redirect Example \(p. 51\)](#)

This section describes how to handle HTTP redirects by using the Amazon S3 REST API. For general information about Amazon S3 redirects, see [Request Redirection and the REST API \(p. 599\)](#) in the Amazon Simple Storage Service API Reference.

Redirects and HTTP User-Agents

Programs that use the Amazon S3 REST API should handle redirects either at the application layer or the HTTP layer. Many HTTP client libraries and user agents can be configured to correctly handle redirects automatically; however, many others have incorrect or incomplete redirect implementations.

Before you rely on a library to fulfill the redirect requirement, test the following cases:

- Verify all HTTP request headers are correctly included in the redirected request (the second request after receiving a redirect) including HTTP standards such as Authorization and Date.
- Verify non-GET redirects, such as PUT and DELETE, work correctly.
- Verify large PUT requests follow redirects correctly.
- Verify PUT requests follow redirects correctly if the 100-continue response takes a long time to arrive.

HTTP user-agents that strictly conform to RFC 2616 might require explicit confirmation before following a redirect when the HTTP request method is not GET or HEAD. It is generally safe to follow redirects generated by Amazon S3 automatically, as the system will issue redirects only to hosts within the `amazonaws.com` domain and the effect of the redirected request will be the same as that of the original request.

Redirects and 100-Continue

To simplify redirect handling, improve efficiencies, and avoid the costs associated with sending a redirected request body twice, configure your application to use 100-continues for PUT operations. When your application uses 100-continue, it does not send the request body until it receives an acknowledgement. If the message is rejected based on the headers, the body of the message is not sent. For more information about 100-continue, go to [RFC 2616 Section 8.2.3](#)

Note

According to RFC 2616, when using `Expect: Continue` with an unknown HTTP server, you should not wait an indefinite period before sending the request body. This is because some HTTP servers do not recognize 100-continue. However, Amazon S3 does recognize if your request contains an `Expect: Continue` and will respond with a provisional 100-continue status or a final status code. Additionally, no redirect error will occur after receiving the provisional 100 continue go-ahead. This will help you avoid receiving a redirect response while you are still writing the request body.

Redirect Example

This section provides an example of client-server interaction using HTTP redirects and 100-continue.

Following is a sample PUT to the `quotes.s3.amazonaws.com` bucket.

```
PUT /nelson.txt HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 returns the following:

```
HTTP/1.1 307 Temporary Redirect
Location: http://quotes.s3-4c25d83b.amazonaws.com/nelson.txt?rk=8d47490b
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Mon, 15 Oct 2007 22:18:46 GMT

Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the
    specified temporary endpoint. Continue to use the
    original request endpoint for future requests.
  </Message>
  <Endpoint>quotes.s3-4c25d83b.amazonaws.com</Endpoint>
  <Bucket>quotes</Bucket>
</Error>
```

The client follows the redirect response and issues a new request to the `quotes.s3-4c25d83b.amazonaws.com` temporary endpoint.

```
PUT /nelson.txt?rk=8d47490b HTTP/1.1
Host: quotes.s3-4c25d83b.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 returns a 100-continue indicating the client should proceed with sending the request body.

```
HTTP/1.1 100 Continue
```

The client sends the request body.

```
ha ha\n
```

Amazon S3 returns the final response.

```
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2007 22:18:48 GMT
```

```
ETag: "a2c8d6b872054293afd41061e93bc289"  
Content-Length: 0  
Server: AmazonS3
```

Working with Amazon S3 Buckets

To upload your data (photos, videos, documents etc.) to Amazon S3, you must first create an S3 bucket in one of the AWS Regions. You can then upload any number of objects to the bucket.

In terms of implementation, buckets and objects are resources, and Amazon S3 provides APIs for you to manage them. For example, you can create a bucket and upload objects using the Amazon S3 API. You can also use the Amazon S3 console to perform these operations. The console uses the Amazon S3 APIs to send requests to Amazon S3.

This section explains how to work with buckets. For information about working with objects, see [Working with Amazon S3 Objects](#) (p. 98).

An Amazon S3 bucket name is globally unique, and the namespace is shared by all AWS accounts. This means that after a bucket is created, the name of that bucket cannot be used by another AWS account in any AWS Region until the bucket is deleted. You should not depend on specific bucket naming conventions for availability or security verification purposes. For bucket naming guidelines, see [Bucket Restrictions and Limitations](#) (p. 58).

Amazon S3 creates buckets in a Region you specify. To optimize latency, minimize costs, or address regulatory requirements, choose any AWS Region that is geographically close to you. For example, if you reside in Europe, you might find it advantageous to create buckets in the EU (Ireland) or EU (Frankfurt) Regions. For a list of Amazon S3 Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.

Note

Objects that belong to a bucket that you create in a specific AWS Region never leave that Region, unless you explicitly transfer them to another Region. For example, objects that are stored in the EU (Ireland) Region never leave it.

Topics

- [Creating a Bucket](#) (p. 53)
- [Managing Public Access to Buckets](#) (p. 55)
- [Accessing a Bucket](#) (p. 55)
- [Bucket Configuration Options](#) (p. 56)
- [Bucket Restrictions and Limitations](#) (p. 58)
- [Examples of Creating a Bucket](#) (p. 59)
- [Deleting or Emptying a Bucket](#) (p. 62)
- [Amazon S3 Default Encryption for S3 Buckets](#) (p. 66)
- [Managing Bucket Website Configuration](#) (p. 69)
- [Amazon S3 Transfer Acceleration](#) (p. 73)
- [Requester Pays Buckets](#) (p. 80)
- [Buckets and Access Control](#) (p. 84)
- [Billing and Usage Reporting for S3 Buckets](#) (p. 84)

Creating a Bucket

Amazon S3 provides APIs for creating and managing buckets. By default, you can create up to 100 buckets in each of your AWS accounts. If you need more buckets, you can increase your account bucket

limit to a maximum of 1,000 buckets by submitting a service limit increase. To learn how to submit a bucket limit increase, see [AWS Service Limits](#) in the *AWS General Reference*.

When you create a bucket, you provide a name and the AWS Region where you want to create the bucket. For information about naming buckets, see [Rules for Bucket Naming](#) (p. 58).

You can store any number of objects in a bucket.

You can create a bucket using any of the following methods:

- Using the console
- Programmatically, using the AWS SDKs

Note

If you need to, you can also make the Amazon S3 REST API calls directly from your code. However, this can be cumbersome because it requires you to write code to authenticate your requests. For more information, see [PUT Bucket](#) in the *Amazon Simple Storage Service API Reference*.

When using the AWS SDKs, you first create a client and then use the client to send a request to create a bucket. When you create the client, you can specify an AWS Region. US East (N. Virginia) is the default Region. Note the following:

- If you create a client by specifying the US East (N. Virginia) Region, the client uses the following endpoint to communicate with Amazon S3:

```
s3.amazonaws.com
```

Note

- You can use this client to create a bucket in any AWS Region that was launched until March 20, 2019. To create a bucket in Regions that were launched after March 20, 2019, you must create a client specific to the Region in which you want to create the bucket. For more information about enabling or disabling an AWS Region, see [AWS Regions and Endpoints](#) in the *AWS General Reference*.
- Buckets created after September 30, 2020, will support only virtual hosted-style requests. Path-style requests will continue to be supported for buckets created on or before this date. For more information, see [Amazon S3 Path Deprecation Plan – The Rest of the Story](#).

In your create bucket request:

- If you don't specify a Region, Amazon S3 creates the bucket in the US East (N. Virginia) Region.
- If you specify an AWS Region, Amazon S3 creates the bucket in the specified Region.
- If you create a client by specifying any other AWS Region, each of these Regions maps to the Region-specific endpoint:

```
s3.<region>.amazonaws.com
```

For example, if you create a client by specifying the eu-west-1 Region, it maps to the following Region-specific endpoint:

```
s3.eu-west-1.amazonaws.com
```

In this case, you can use the client to create a bucket only in the eu-west-1 Region. Amazon S3 returns an error if you specify any other Region in your request to create a bucket.

- If you create a client to access a dual-stack endpoint, you must specify an AWS Region. For more information, see [Dual-Stack Endpoints](#) (p. 14).

For a list of available AWS Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.

For examples, see [Examples of Creating a Bucket](#) (p. 59).

About Permissions

You can use your AWS account root credentials to create a bucket and perform any other Amazon S3 operation. However, AWS recommends not using the root credentials of your AWS account to make requests such as to create a bucket. Instead, create an IAM user, and grant that user full access (users by default have no permissions). We refer to these users as administrator users. You can use the administrator user credentials, instead of the root credentials of your account, to interact with AWS and perform tasks, such as create a bucket, create users, and grant them permissions.

For more information, see [Root Account Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in the *IAM User Guide*.

The AWS account that creates a resource owns that resource. For example, if you create an IAM user in your AWS account and grant the user permission to create a bucket, the user can create a bucket. But the user does not own the bucket; the AWS account to which the user belongs owns the bucket. The user will need additional permission from the resource owner to perform any other bucket operations. For more information about managing permissions for your Amazon S3 resources, see [Identity and Access Management in Amazon S3](#) (p. 301).

Managing Public Access to Buckets

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, or both. To help you manage public access to Amazon S3 resources, Amazon S3 provides *block public access* settings. Amazon S3 block public access settings can override ACLs and bucket policies so that you can enforce uniform limits on public access to these resources. You can apply block public access settings to individual buckets or to all buckets in your account.

To help ensure that all of your Amazon S3 buckets and objects have their public access blocked, we recommend that you turn on all four settings for block public access for your account. These settings block public access for all current and future buckets.

Before applying these settings, verify that your applications will work correctly without public access. If you require some level of public access to your buckets or objects, for example to host a static website as described at [Hosting a Static Website on Amazon S3](#) (p. 503), you can customize the individual settings to suit your storage use cases. For more information, see [Using Amazon S3 Block Public Access](#) (p. 414).

Accessing a Bucket

You can access your bucket using the Amazon S3 console. Using the console UI, you can perform almost all bucket operations without having to write any code.

If you access a bucket programmatically, note that Amazon S3 supports RESTful architecture in which your buckets and objects are resources, each with a resource URI that uniquely identifies the resource.

Amazon S3 supports both virtual-hosted-style and path-style URLs to access a bucket.

- In a virtual-hosted-style URL, the bucket name is part of the domain name in the URL. For example:
 - `http://bucket.s3-aws-region.amazonaws.com`

- `http://bucket.s3.amazonaws.com`

Note

Buckets created in Regions launched after March 20, 2019 are not reachable via the `https://bucket.s3.amazonaws.com` naming scheme.

In a virtual-hosted-style URL, you can use either of these endpoints. If you make a request to the `http://bucket.s3.amazonaws.com` endpoint, the DNS has sufficient information to route your request directly to the Region where your bucket resides.

For more information, see [Virtual Hosting of Buckets \(p. 45\)](#).

- In a path-style URL, the bucket name is not part of the domain. For example:
 - Region-specific endpoint, `http://s3-aws-region.amazonaws.com/bucket`
 - US East (N. Virginia) Region endpoint, `http://s3.amazonaws.com/bucket`

In a path-style URL, the endpoint you use must match the Region in which the bucket resides. For example, if your bucket is in the South America (São Paulo) Region, you must use the `http://s3.sa-east-1.amazonaws.com/bucket` endpoint. If your bucket is in the US East (N. Virginia) Region, you must use the `http://s3.amazonaws.com/bucket` endpoint.

Important

Because buckets can be accessed using path-style and virtual-hosted-style URLs, we recommend that you create buckets with DNS-compliant bucket names. For more information, see [Bucket Restrictions and Limitations \(p. 58\)](#).

Accessing an S3 Bucket over IPv6

Amazon S3 has a set of dual-stack endpoints, which support requests to S3 buckets over both Internet Protocol version 6 (IPv6) and IPv4. For more information, see [Making Requests over IPv6 \(p. 12\)](#).

Bucket Configuration Options

Amazon S3 supports various options for you to configure your bucket. For example, you can configure your bucket for website hosting, add configuration to manage lifecycle of objects in the bucket, and configure the bucket to log all access to the bucket. Amazon S3 supports subresources for you to store, and manage the bucket configuration information. That is, using the Amazon S3 API, you can create and manage these subresources. You can also use the console or the AWS SDKs.

Note

There are also object-level configurations. For example, you can configure object-level permissions by configuring an access control list (ACL) specific to that object.

These are referred to as subresources because they exist in the context of a specific bucket or object. The following table lists subresources that enable you to manage bucket-specific configurations.

Subresource	Description
<i>cors</i> (cross-origin resource sharing)	You can configure your bucket to allow cross-origin requests. For more information, see Enabling Cross-Origin Resource Sharing .
<i>event notification</i>	You can enable your bucket to send you notifications of specified bucket events. For more information, see Configuring Amazon S3 Event Notifications (p. 530) .

Subresource	Description
<i>lifecycle</i>	<p>You can define lifecycle rules for objects in your bucket that have a well-defined lifecycle. For example, you can define a rule to archive objects one year after creation, or delete an object 10 years after creation.</p> <p>For more information, see Object Lifecycle Management.</p>
<i>location</i>	<p>When you create a bucket, you specify the AWS Region where you want Amazon S3 to create the bucket. Amazon S3 stores this information in the location subresource and provides an API for you to retrieve this information.</p>
<i>logging</i>	<p>Logging enables you to track requests for access to your bucket. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. Access log information can be useful in security and access audits. It can also help you learn about your customer base and understand your Amazon S3 bill.</p> <p>For more information, see Amazon S3 Server Access Logging (p. 647).</p>
<i>object locking</i>	<p>To use Amazon S3 object lock, you must enable it for a bucket. You can also optionally configure a default retention mode and period that applies to new objects that are placed in the bucket.</p> <p>For more information, see Bucket Configuration (p. 456).</p>
<i>policy and ACL</i> (access control list)	<p>All your resources (such as buckets and objects) are private by default. Amazon S3 supports both bucket policy and access control list (ACL) options for you to grant and manage bucket-level permissions. Amazon S3 stores the permission information in the <i>policy</i> and <i>acl</i> subresources.</p> <p>For more information, see Identity and Access Management in Amazon S3 (p. 301).</p>
<i>replication</i>	<p>Replication is the automatic, asynchronous copying of objects across buckets in different or the same AWS Regions. For more information, see Replication (p. 551).</p>
<i>requestPayment</i>	<p>By default, the AWS account that creates the bucket (the bucket owner) pays for downloads from the bucket. Using this subresource, the bucket owner can specify that the person requesting the download will be charged for the download. Amazon S3 provides an API for you to manage this subresource.</p> <p>For more information, see Requester Pays Buckets (p. 80).</p>
<i>tagging</i>	<p>You can add cost allocation tags to your bucket to categorize and track your AWS costs. Amazon S3 provides the <i>tagging</i> subresource to store and manage tags on a bucket. Using tags you apply to your bucket, AWS generates a cost allocation report with usage and costs aggregated by your tags.</p> <p>For more information, see Billing and Usage Reporting for S3 Buckets (p. 84).</p>
<i>transfer acceleration</i>	<p>Transfer Acceleration enables fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration takes advantage of Amazon CloudFront's globally distributed edge locations.</p> <p>For more information, see Amazon S3 Transfer Acceleration (p. 73).</p>

Subresource	Description
<i>versioning</i>	<p>Versioning helps you recover accidental overwrites and deletes.</p> <p>We recommend versioning as a best practice to recover objects from being deleted or overwritten by mistake.</p> <p>For more information, see Using Versioning (p. 432).</p>
<i>website</i>	<p>You can configure your bucket for static website hosting. Amazon S3 stores this configuration by creating a <i>website</i> subresource.</p> <p>For more information, see Hosting a Static Website on Amazon S3.</p>

Bucket Restrictions and Limitations

A bucket is owned by the AWS account that created it. By default, you can create up to 100 buckets in each of your AWS accounts. If you need additional buckets, you can increase your account bucket limit to a maximum of 1,000 buckets by submitting a service limit increase. For information about how to increase your bucket limit, see [AWS Service Limits](#) in the *AWS General Reference*.

Bucket ownership is not transferable; however, if a bucket is empty, you can delete it. After a bucket is deleted, the name becomes available to reuse, but the name might not be available for you to reuse for various reasons. For example, some other account could create a bucket with that name. Note, too, that it might take some time before the name can be reused. So if you want to use the same bucket name, don't delete the bucket.

There is no limit to the number of objects that can be stored in a bucket and no difference in performance whether you use many buckets or just a few. You can store all of your objects in a single bucket, or you can organize them across several buckets.

Important

After you have created a bucket, you can't change its Region.

You cannot create a bucket within another bucket.

The high-availability engineering of Amazon S3 is focused on get, put, list, and delete operations. Because bucket operations work against a centralized, global resource space, it is not appropriate to create or delete buckets on the high-availability code path of your application. It is better to create or delete buckets in a separate initialization or setup routine that you run less often.

Note

If your application automatically creates buckets, choose a bucket naming scheme that is unlikely to cause naming conflicts. Ensure that your application logic will choose a different bucket name if a bucket name is already taken.

Rules for Bucket Naming

After you create an S3 bucket, you can't change the bucket name, so choose the name wisely.

Important

On March 1, 2018, we updated our naming conventions for S3 buckets in the US East (N. Virginia) Region to match the naming conventions that we use in all other worldwide AWS Regions. Amazon S3 no longer supports creating bucket names that contain uppercase letters or underscores. This change ensures that each bucket can be addressed using virtual host style addressing, such as `https://myawsbucket.s3.amazonaws.com`. We highly recommend that you review your existing bucket-creation processes to ensure that you follow these DNS-compliant naming conventions.

The following are the rules for naming S3 buckets in all AWS Regions:

- Bucket names must be unique across all existing bucket names in Amazon S3.
- Bucket names must comply with DNS naming conventions.
- Bucket names must be at least 3 and no more than 63 characters long.
- Bucket names must not contain uppercase characters or underscores.
- Bucket names must start with a lowercase letter or number.
- Bucket names must be a series of one or more labels. Adjacent labels are separated by a single period (.). Bucket names can contain lowercase letters, numbers, and hyphens. Each label must start and end with a lowercase letter or a number.
- Bucket names must not be formatted as an IP address (for example, 192.168.5.4).
- When you use virtual hosted-style buckets with Secure Sockets Layer (SSL), the SSL wildcard certificate only matches buckets that don't contain periods. To work around this, use HTTP or write your own certificate verification logic. We recommend that you do not use periods (".") in bucket names when using virtual hosted-style buckets.

Legacy Non-DNS-Compliant Bucket Names

Beginning on March 1, 2018, we updated our naming conventions for S3 buckets in the US East (N. Virginia) Region to require DNS-compliant names.

The US East (N. Virginia) Region previously allowed more relaxed standards for bucket naming, which could have resulted in a bucket name that is not DNS-compliant. For example, `MyAWSbucket` was a valid bucket name, even though it contains uppercase letters. If you try to access this bucket by using a virtual-hosted-style request (`http://MyAWSbucket.s3.amazonaws.com/yourobject`), the URL resolves to the bucket `myawsbucket` and not the bucket `MyAWSbucket`. In response, Amazon S3 returns a "bucket not found" error. For more information about virtual-hosted-style access to your buckets, see [Virtual Hosting of Buckets \(p. 45\)](#).

The legacy rules for bucket names in the US East (N. Virginia) Region allowed bucket names to be as long as 255 characters, and bucket names could contain any combination of uppercase letters, lowercase letters, numbers, periods (.), hyphens (-), and underscores (_).

The name of the bucket used for Amazon S3 Transfer Acceleration must be DNS-compliant and must not contain periods ("."). For more information about transfer acceleration, see [Amazon S3 Transfer Acceleration \(p. 73\)](#).

Examples of Creating a Bucket

Topics

- [Using the Amazon S3 Console \(p. 60\)](#)
- [Using the AWS SDK for Java \(p. 60\)](#)
- [Using the AWS SDK for .NET \(p. 61\)](#)
- [Using the AWS SDK for Ruby Version 3 \(p. 62\)](#)
- [Using Other AWS SDKs \(p. 62\)](#)

The following code examples create a bucket programmatically using the AWS SDKs for Java, .NET, and Ruby. The code examples perform the following tasks:

- Create a bucket, if the bucket doesn't already exist—The examples create a bucket by performing the following tasks:

- Create a client by explicitly specifying an AWS Region (the example uses the `s3-eu-west-1` Region). Accordingly, the client communicates with Amazon S3 using the `s3-eu-west-1.amazonaws.com` endpoint. You can specify any other AWS Region. For a list of AWS Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.
- Send a create bucket request by specifying only a bucket name. The create bucket request doesn't specify another AWS Region. The client sends a request to Amazon S3 to create the bucket in the Region you specified when creating the client. Once you have created a bucket, you can't change its Region.

Note

If you explicitly specify an AWS Region in your create bucket request that is different from the Region you specified when you created the client, you might get an error. For more information, see [Creating a Bucket](#) (p. 53).

The SDK libraries send the PUT bucket request to Amazon S3 to create the bucket. For more information, see [PUT Bucket](#).

- Retrieve information about the location of the bucket—Amazon S3 stores bucket location information in the *location* subresource that is associated with the bucket. The SDK libraries send the GET Bucket location request (see [GET Bucket location](#)) to retrieve this information.

Using the Amazon S3 Console

To create a bucket using the Amazon S3 console, see [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Using the AWS SDK for Java

Example

This example shows how to create an Amazon S3 bucket using the AWS SDK for Java. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.GetBucketLocationRequest;

import java.io.IOException;

public class CreateBucket {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            if (!s3Client.doesBucketExistV2(bucketName)) {
                // Because the CreateBucketRequest object doesn't specify a region, the
                // bucket is created in the region specified in the client.
            }
        }
    }
}
```

```
s3Client.createBucket(new CreateBucketRequest(bucketName));

// Verify that the bucket was created by retrieving it and checking its
location.
String bucketLocation = s3Client.getBucketLocation(new
GetBucketLocationRequest(bucketName));
System.out.println("Bucket location: " + bucketLocation);
}
} catch (AmazonServiceException e) {
// The call was transmitted successfully, but Amazon S3 couldn't process
// it and returned an error response.
e.printStackTrace();
} catch (SdkClientException e) {
// Amazon S3 couldn't be contacted for a response, or the client
// couldn't parse the response from Amazon S3.
e.printStackTrace();
}
}
}
```

Using the AWS SDK for .NET

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Example

```
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CreateBucketTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            CreateBucketAsync().Wait();
        }

        static async Task CreateBucketAsync()
        {
            try
            {
                if (!(await AmazonS3Util.DoesS3BucketExistAsync(s3Client, bucketName)))
                {
                    var putBucketRequest = new PutBucketRequest
                    {
                        BucketName = bucketName,
                        UseClientRegion = true
                    };

                    PutBucketResponse putBucketResponse = await
s3Client.PutBucketAsync(putBucketRequest);
                }
            }
        }
    }
}
```

```
        }
        // Retrieve the bucket location.
        string bucketLocation = await FindBucketLocationAsync(s3Client);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}
static async Task<string> FindBucketLocationAsync(IAmazonS3 client)
{
    string bucketLocation;
    var request = new GetBucketLocationRequest()
    {
        BucketName = bucketName
    };
    GetBucketLocationResponse response = await
client.GetBucketLocationAsync(request);
    bucketLocation = response.Location.ToString();
    return bucketLocation;
}
}
```

Using the AWS SDK for Ruby Version 3

For information about how to create and test a working sample, see [Using the AWS SDK for Ruby - Version 3 \(p. 679\)](#).

Example

```
require 'aws-sdk-s3'

s3 = Aws::S3::Client.new(region: 'us-west-2')
s3.create_bucket(bucket: 'bucket-name')
```

Using Other AWS SDKs

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Deleting or Emptying a Bucket

It is easy to delete an empty bucket. However, in some situations, you may need to delete or empty a bucket that contains objects. In this section, we'll explain how to delete objects in an unversioned bucket, and how to delete object versions and delete markers in a bucket that has versioning enabled. For more information about versioning, see [Using Versioning \(p. 432\)](#). In some situations, you may choose to empty a bucket instead of deleting it. This section explains various options you can use to delete or empty a bucket that contains objects.

Topics

- [Delete a Bucket \(p. 63\)](#)
- [Empty a Bucket \(p. 65\)](#)

Delete a Bucket

You can delete a bucket and its content programmatically using the AWS SDKs. You can also use lifecycle configuration on a bucket to empty its content and then delete the bucket. There are additional options, such as using Amazon S3 console and AWS CLI, but there are limitations on these methods based on the number of objects in your bucket and the bucket's versioning status.

Delete a Bucket: Using the Amazon S3 Console

The Amazon S3 console supports deleting a bucket that may or may not be empty. For information about using the Amazon S3 console to delete a bucket, see [How Do I Delete an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Delete a Bucket: Using the AWS CLI

You can delete a bucket that contains objects using the AWS CLI only if the bucket does not have versioning enabled. If your bucket does not have versioning enabled, you can use the `rb` (remove bucket) AWS CLI command with `--force` parameter to remove a non-empty bucket. This command deletes all objects first and then deletes the bucket.

```
$ aws s3 rb s3://bucket-name --force
```

For more information, see [Using High-Level S3 Commands with the AWS Command Line Interface](#) in the AWS Command Line Interface User Guide.

To delete a non-empty bucket that does not have versioning enabled, you have the following options:

- Delete the bucket programmatically using the AWS SDK.
- Delete all of the objects using the bucket's lifecycle configuration and then delete the empty bucket using the Amazon S3 console.

Delete a Bucket: Using Lifecycle Configuration

You can configure lifecycle on your bucket to expire objects, Amazon S3 then deletes expired objects. You can add lifecycle configuration rules to expire all or a subset of objects with a specific key name prefix. For example, to remove all objects in a bucket, you can set a lifecycle rule to expire objects one day after creation.

If your bucket has versioning enabled, you can also configure the rule to expire noncurrent objects.

After Amazon S3 deletes all of the objects in your bucket, you can delete the bucket or keep it.

Important

If you just want to empty the bucket and not delete it, make sure you remove the lifecycle configuration rule you added to empty the bucket so that any new objects you create in the bucket will remain in the bucket.

For more information, see [Object Lifecycle Management \(p. 119\)](#) and [Configuring Object Expiration \(p. 126\)](#).

Delete a Bucket: Using the AWS SDKs

You can use the AWS SDKs to delete a bucket. The following sections provide examples of how to delete a bucket using the AWS SDK for Java and .NET. First, the code deletes objects in the bucket and then it deletes the bucket. For information about other AWS SDKs, see [Tools for Amazon Web Services](#).

Delete a Bucket Using the AWS SDK for Java

The following Java example deletes a bucket that contains objects. The example deletes all objects, and then it deletes the bucket. The example works for buckets with or without versioning enabled.

Note

For buckets without versioning enabled, you can delete all objects directly and then delete the bucket. For buckets with versioning enabled, you must delete all object versions before deleting the bucket.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;

public class DeleteBucket {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Delete all objects from the bucket. This is sufficient
            // for unversioned buckets. For versioned buckets, when you attempt to delete
            // objects, Amazon S3 inserts
            // delete markers for all objects, but doesn't delete the object versions.
            // To delete objects from versioned buckets, delete all of the object versions
            // before deleting
            // the bucket (see below for an example).
            ObjectListing objectListing = s3Client.listObjects(bucketName);
            while (true) {
                Iterator<S3ObjectSummary> objIter =
                    objectListing.getObjectSummaries().iterator();
                while (objIter.hasNext()) {
                    s3Client.deleteObject(bucketName, objIter.next().getKey());
                }

                // If the bucket contains many objects, the listObjects() call
                // might not return all of the objects in the first listing. Check to
                // see whether the listing was truncated. If so, retrieve the next page of
                // objects
                // and delete them.
                if (objectListing.isTruncated()) {
                    objectListing = s3Client.listNextBatchOfObjects(objectListing);
                } else {
                    break;
                }
            }

            // Delete all object versions (required for versioned buckets).
```

```
VersionListing versionList = s3Client.listVersions(new
ListVersionsRequest().withBucketName(bucketName));
while (true) {
    Iterator<S3VersionSummary> versionIter =
versionList.getVersionSummaries().iterator();
    while (versionIter.hasNext()) {
        S3VersionSummary vs = versionIter.next();
        s3Client.deleteVersion(bucketName, vs.getKey(), vs.getVersionId());
    }

    if (versionList.isTruncated()) {
        versionList = s3Client.listNextBatchOfVersions(versionList);
    } else {
        break;
    }
}

// After all objects and object versions are deleted, delete the bucket.
s3Client.deleteBucket(bucketName);
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client couldn't
    // parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Empty a Bucket

You can empty a bucket's content (that is, delete all content, but keep the bucket) programmatically using the AWS SDK. You can also specify lifecycle configuration on a bucket to expire objects so that Amazon S3 can delete them. There are additional options, such as using Amazon S3 console and AWS CLI, but there are limitations on this method based on the number of objects in your bucket and the bucket's versioning status.

Topics

- [Empty a Bucket: Using the Amazon S3 console \(p. 65\)](#)
- [Empty a Bucket: Using the AWS CLI \(p. 65\)](#)
- [Empty a Bucket: Using Lifecycle Configuration \(p. 66\)](#)
- [Empty a Bucket: Using the AWS SDKs \(p. 66\)](#)

Empty a Bucket: Using the Amazon S3 console

For information about using the Amazon S3 console to empty a bucket, see [How Do I Empty an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*

Empty a Bucket: Using the AWS CLI

You can empty a bucket using the AWS CLI only if the bucket does not have versioning enabled. If your bucket does not have versioning enabled, you can use the `rm` (remove) AWS CLI command with the `--recursive` parameter to empty a bucket (or remove a subset of objects with a specific key name prefix).

The following `rm` command removes objects with key name prefix `doc`, for example, `doc/doc1` and `doc/doc2`.

```
$ aws s3 rm s3://bucket-name/doc --recursive
```

Use the following command to remove all objects without specifying a prefix.

```
$ aws s3 rm s3://bucket-name --recursive
```

For more information, see [Using High-Level S3 Commands with the AWS Command Line Interface](#) in the AWS Command Line Interface User Guide.

Note

You cannot remove objects from a bucket with versioning enabled. Amazon S3 adds a delete marker when you delete an object, which is what this command will do. For more information about versioning, see [Using Versioning \(p. 432\)](#).

To empty a bucket with versioning enabled, you have the following options:

- Delete the bucket programmatically using the AWS SDK.
- Use the bucket's lifecycle configuration to request that Amazon S3 delete the objects.
- Use the Amazon S3 console. For more information, see [How Do I Empty an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Empty a Bucket: Using Lifecycle Configuration

You can configure lifecycle on your bucket to expire objects and request that Amazon S3 delete expired objects. You can add lifecycle configuration rules to expire all or a subset of objects with a specific key name prefix. For example, to remove all objects in a bucket, you can set lifecycle rule to expire objects one day after creation.

If your bucket has versioning enabled, you can also configure the rule to expire noncurrent objects.

Warning

After your objects expire, Amazon S3 deletes the expired objects. If you just want to empty the bucket and not delete it, make sure you remove the lifecycle configuration rule you added to empty the bucket so that any new objects you create in the bucket will remain in the bucket.

For more information, see [Object Lifecycle Management \(p. 119\)](#) and [Configuring Object Expiration \(p. 126\)](#).

Empty a Bucket: Using the AWS SDKs

You can use the AWS SDKs to empty a bucket or remove a subset of objects with a specific key name prefix.

For an example of how to empty a bucket using AWS SDK for Java, see [Delete a Bucket Using the AWS SDK for Java \(p. 64\)](#). The code deletes all objects, regardless of whether the bucket has versioning enabled or not, and then it deletes the bucket. To just empty the bucket, make sure you remove the statement that deletes the bucket.

For more information about using other AWS SDKs, see [Tools for Amazon Web Services](#).

Amazon S3 Default Encryption for S3 Buckets

Amazon S3 default encryption provides a way to set the default encryption behavior for an S3 bucket. You can set default encryption on a bucket so that all objects are encrypted when they are stored in the bucket. The objects are encrypted using server-side encryption with either Amazon S3-managed keys (SSE-S3) or AWS KMS-managed keys (SSE-KMS).

When you use server-side encryption, Amazon S3 encrypts an object before saving it to disk in its data centers and decrypts it when you download the objects. For more information about protecting data using server-side encryption and encryption key management, see [Protecting Data Using Server-Side Encryption](#) (p. 265).

Default encryption works with all existing and new S3 buckets. Without default encryption, to encrypt all objects stored in a bucket, you must include encryption information with every object storage request. You must also set up an S3 bucket policy to reject storage requests that don't include encryption information.

There are no new charges for using default encryption for S3 buckets. Requests to configure the default encryption feature incur standard Amazon S3 request charges. For information about pricing, see [Amazon S3 Pricing](#). For SSE-KMS encryption key storage, AWS Key Management Service charges apply and are listed at [AWS KMS Pricing](#).

Topics

- [How Do I Set Up Amazon S3 Default Encryption for an S3 Bucket?](#) (p. 67)
- [Moving to Default Encryption from Using Bucket Policies for Encryption Enforcement](#) (p. 68)
- [Using Default Encryption with Replication](#) (p. 68)
- [Monitoring Default Encryption with CloudTrail and CloudWatch](#) (p. 69)
- [More Info](#) (p. 69)

How Do I Set Up Amazon S3 Default Encryption for an S3 Bucket?

This section describes how to set up Amazon S3 default encryption. You can use the AWS SDKs, the Amazon S3 REST API, the AWS Command Line Interface (AWS CLI), or the Amazon S3 console to enable the default encryption. The easiest way to set up default encryption for an S3 bucket is by using the AWS Management Console.

You can set up default encryption on a bucket using any of the following ways:

- Use the Amazon S3 console. For more information, see [How Do I Enable Default Encryption for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.
- Use the following REST APIs:
 - Use the REST API [PUT Bucket encryption](#) operation to enable default encryption and to set the type of server-side encryption to use—SSE-S3 or SSE-KMS.
 - Use the REST API [DELETE Bucket encryption](#) to disable the default encryption of objects. After you disable default encryption, Amazon S3 encrypts objects only if `PUT` requests include the encryption information. For more information, see [PUT Object](#) and [PUT Object - Copy](#).
 - Use the REST API [GET Bucket encryption](#) to check the current default encryption configuration.
- Use the AWS CLI and AWS SDKs. For more information, see [Using the AWS SDKs, CLI, and Explorers](#) (p. 669).

After you enable default encryption for a bucket, the following encryption behavior applies:

- There is no change to the encryption of the objects that existed in the bucket before default encryption was enabled.
- When you upload objects after enabling default encryption:
 - If your `PUT` request headers don't include encryption information, Amazon S3 uses the bucket's default encryption settings to encrypt the objects.

- If your `PUT` request headers include encryption information, Amazon S3 uses the encryption information from the `PUT` request to encrypt objects before storing them in Amazon S3. If the `PUT` succeeds, the response is an `HTTP/1.1 200 OK` with the encryption information in the response headers. For more information, see [PUT Object](#).
- If you use the SSE-KMS option for your default encryption configuration, you are subject to the RPS (requests per second) limits of AWS KMS. For more information about AWS KMS limits and how to request a limit increase, see [AWS KMS limits](#).

Moving to Default Encryption from Using Bucket Policies for Encryption Enforcement

If you currently enforce object encryption for an S3 bucket by using a bucket policy to reject `PUT` requests without encryption headers, we recommend that you use the following procedure to start using default encryption.

To change from using a bucket policy to reject `PUT` requests without encryption headers to using default encryption

1. If you plan to specify that default encryption use SSE-KMS, make sure that all `PUT` and `GET` object requests are signed using Signature Version 4 and sent over an SSL connection to Amazon S3. For information about using AWS KMS, see [Protecting Data Using Server-Side Encryption with keys stored in AWS KMS\(SSE-KMS\)](#) (p. 265).
- Note**
By default, the Amazon S3 console, the AWS CLI version 1.11.108 and later, and all AWS SDKs released after May 2016 use Signature Version 4 signed requests sent to Amazon S3 over an SSL connection.
2. Delete the bucket policy statements that reject `PUT` requests without encryption headers. (We recommend that you save a backup copy of the bucket policy that is being replaced.)
 3. To ensure that the encryption behavior is set as you want, test multiple `PUT` requests to closely simulate your actual workload.
 4. If you are using default encryption with SSE-KMS, monitor your clients for failing `PUT` and `GET` requests that weren't failing before your changes. Most likely these are the requests that you didn't update according to Step 1. Change the failing `PUT` or `GET` requests to be signed with AWS Signature Version 4 and sent over SSL.

After you enable default encryption for your S3 bucket, objects stored in Amazon S3 through any `PUT` requests without encryption headers are encrypted using the bucket-level default encryption settings.

Using Default Encryption with Replication

After you enable default encryption for a replication destination bucket, the following encryption behavior applies:

- If objects in the source bucket are not encrypted, the replica objects in the destination bucket are encrypted using the default encryption settings of the destination bucket. This results in the `ETag` of the source object being different from the `ETag` of the replica object. You must update applications that use the `ETag` to accommodate for this difference.
- If objects in the source bucket are encrypted using SSE-S3 or SSE-KMS, the replica objects in the destination bucket use the same encryption as the source object encryption. The default encryption settings of the destination bucket are not used.

Monitoring Default Encryption with CloudTrail and CloudWatch

You can track default encryption configuration requests through AWS CloudTrail events. The API event names used in CloudTrail logs are `PutBucketEncryption`, `GetBucketEncryption`, and `DeleteBucketEncryption`. You can also create Amazon CloudWatch Events with S3 bucket-level operations as the event type. For more information about CloudTrail events, see [How Do I Enable Object-Level Logging for an S3 Bucket with CloudWatch Data Events?](#)

You can use CloudTrail logs for object-level Amazon S3 actions to track `PUT` and `POST` requests to Amazon S3 to verify whether default encryption is being used to encrypt objects when incoming `PUT` requests don't have encryption headers.

When Amazon S3 encrypts an object using the default encryption settings, the log includes the following field as the name/value pair: `"SSEApplied": "Default_SSE_S3"` or `"SSEApplied": "Default_SSE_KMS"`.

When Amazon S3 encrypts an object using the `PUT` encryption headers, the log includes the following field as the name/value pair: `"SSEApplied": "SSE_S3"`, `"SSEApplied": "SSE_KMS"`, or `"SSEApplied": "SSE_C"`. For multipart uploads, this information is included in the `InitiateMultipartUpload` API requests. For more information about using CloudTrail and CloudWatch, see [Monitoring Amazon S3 \(p. 610\)](#).

More Info

- [PUT Bucket encryption](#)
- [DELETE Bucket encryption](#)
- [GET Bucket encryption](#)

Managing Bucket Website Configuration

Topics

- [Managing Websites with the AWS Management Console \(p. 69\)](#)
- [Managing Websites with the AWS SDK for Java \(p. 70\)](#)
- [Managing Websites with the AWS SDK for .NET \(p. 71\)](#)
- [Managing Websites with the AWS SDK for PHP \(p. 72\)](#)
- [Managing Websites with the REST API \(p. 73\)](#)

You can host static websites in Amazon S3 by configuring your bucket for website hosting. For more information, see [Hosting a Static Website on Amazon S3 \(p. 503\)](#). There are several ways you can manage your bucket's website configuration. You can use the AWS Management Console to manage configuration without writing any code. You can programmatically create, update, and delete the website configuration by using the AWS SDKs. The SDKs provide wrapper classes around the Amazon S3 REST API. If your application requires it, you can send REST API requests directly from your application.

Managing Websites with the AWS Management Console

For more information, see [Configuring a Bucket for Website Hosting \(p. 505\)](#).

Managing Websites with the AWS SDK for Java

The following example shows how to use the AWS SDK for Java to manage website configuration for a bucket. To add a website configuration to a bucket, you provide a bucket name and a website configuration. The website configuration must include an index document and can include an optional error document. These documents must already exist in the bucket. For more information, see [PUT Bucket website](#). For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3](#) (p. 503).

Example

The following example uses the AWS SDK for Java to add a website configuration to a bucket, retrieve and print the configuration, and then delete the configuration and verify the deletion. For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;

import java.io.IOException;

public class WebsiteConfiguration {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String indexDocName = "**** Index document name ****";
        String errorDocName = "**** Error document name ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Print the existing website configuration, if it exists.
            printWebsiteConfig(s3Client, bucketName);

            // Set the new website configuration.
            s3Client.setBucketWebsiteConfiguration(bucketName, new
            BucketWebsiteConfiguration(indexDocName, errorDocName));

            // Verify that the configuration was set properly by printing it.
            printWebsiteConfig(s3Client, bucketName);

            // Delete the website configuration.
            s3Client.deleteBucketWebsiteConfiguration(bucketName);

            // Verify that the website configuration was deleted by printing it.
            printWebsiteConfig(s3Client, bucketName);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```



```
    }  
}  
  
private static void printWebsiteConfig(AmazonS3 s3Client, String bucketName) {  
    System.out.println("Website configuration: ");  
    BucketWebsiteConfiguration bucketWebsiteConfig =  
s3Client.GetBucketWebsiteConfiguration(bucketName);  
    if (bucketWebsiteConfig == null) {  
        System.out.println("No website config.");  
    } else {  
        System.out.println("Index doc: " +  
bucketWebsiteConfig.GetIndexDocumentSuffix());  
        System.out.println("Error doc: " + bucketWebsiteConfig.GetErrorDocument());  
    }  
}
```

Managing Websites with the AWS SDK for .NET

The following example shows how to use the AWS SDK for .NET to manage website configuration for a bucket. To add a website configuration to a bucket, you provide a bucket name and a website configuration. The website configuration must include an index document and can contain an optional error document. These documents must be stored in the bucket. For more information, see [PUT Bucket website](#). For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3](#) (p. 503).

Example

The following C# code example adds a website configuration to the specified bucket. The configuration specifies both the index document and the error document names. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples](#) (p. 678).

```
using Amazon.S3;  
using Amazon.S3.Model;  
using System;  
using System.Threading.Tasks;  
  
namespace Amazon.DocSamples.S3  
{  
    class WebsiteConfigTest  
    {  
        private const string bucketName = "*** bucket name ***";  
        private const string indexDocumentSuffix = "*** index object key ***"; // For  
example, index.html.  
        private const string errorDocument = "*** error object key ***"; // For example,  
error.html.  
        // Specify your bucket region (an example region is shown).  
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;  
        private static IAmazonS3 client;  
        public static void Main()  
        {  
            client = new AmazonS3Client(bucketRegion);  
            AddWebsiteConfigurationAsync(bucketName, indexDocumentSuffix,  
errorDocument).Wait();  
        }  
  
        static async Task AddWebsiteConfigurationAsync(string bucketName,  
string indexDocumentSuffix,  
string errorDocument)  
        {  

```

```
try
{
    // 1. Put the website configuration.
    PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
    {
        BucketName = bucketName,
        WebsiteConfiguration = new WebsiteConfiguration()
        {
            IndexDocumentSuffix = indexDocumentSuffix,
            ErrorDocument = errorDocument
        }
    };
    PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);

    // 2. Get the website configuration.
    GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
    {
        BucketName = bucketName
    };
    GetBucketWebsiteResponse getResponse = await
client.GetBucketWebsiteAsync(getRequest);
    Console.WriteLine("Index document: {0}",
getResponse.WebsiteConfiguration.IndexDocumentSuffix);
    Console.WriteLine("Error document: {0}",
getResponse.WebsiteConfiguration.ErrorDocument);
}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
}
}
```

Managing Websites with the AWS SDK for PHP

This topic explains how to use classes from the AWS SDK for PHP to configure and manage an Amazon S3 bucket for website hosting. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed. For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3 \(p. 503\)](#).

The following PHP example adds a website configuration to the specified bucket. The `create_website_config` method explicitly provides the index document and error document names. The example also retrieves the website configuration and prints the response. For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3 \(p. 503\)](#).

For instructions on creating and testing a working sample, see [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
```

```
$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Add the website configuration.
$s3->putBucketWebsite([
    'Bucket' => $bucket,
    'WebsiteConfiguration' => [
        'IndexDocument' => ['Suffix' => 'index.html'],
        'ErrorDocument' => ['Key' => 'error.html']
    ]
]);

// Retrieve the website configuration.
$result = $s3->getBucketWebsite([
    'Bucket' => $bucket
]);
echo $result->getPath('IndexDocument/Suffix');

// Delete the website configuration.
$s3->deleteBucketWebsite([
    'Bucket' => $bucket
]);
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Managing Websites with the REST API

You can use the AWS Management Console or the AWS SDK to configure a bucket as a website. However, if your application requires it, you can send REST requests directly. For more information, see the following sections in the Amazon Simple Storage Service API Reference.

- [PUT Bucket website](#)
- [GET Bucket website](#)
- [DELETE Bucket website](#)

Amazon S3 Transfer Acceleration

Amazon S3 Transfer Acceleration enables fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration takes advantage of Amazon CloudFront's globally distributed edge locations. As the data arrives at an edge location, data is routed to Amazon S3 over an optimized network path.

When using Transfer Acceleration, additional data transfer charges may apply. For more information about pricing, see [Amazon S3 Pricing](#).

Topics

- [Why Use Amazon S3 Transfer Acceleration? \(p. 74\)](#)
- [Getting Started with Amazon S3 Transfer Acceleration \(p. 74\)](#)

- [Requirements for Using Amazon S3 Transfer Acceleration](#) (p. 75)
- [Amazon S3 Transfer Acceleration Examples](#) (p. 76)

Why Use Amazon S3 Transfer Acceleration?

You might want to use Transfer Acceleration on a bucket for various reasons, including the following:

- You have customers that upload to a centralized bucket from all over the world.
- You transfer gigabytes to terabytes of data on a regular basis across continents.
- You are unable to utilize all of your available bandwidth over the Internet when uploading to Amazon S3.

For more information about when to use Transfer Acceleration, see [Amazon S3 FAQs](#).

Using the Amazon S3 Transfer Acceleration Speed Comparison Tool

You can use the [Amazon S3 Transfer Acceleration Speed Comparison tool](#) to compare accelerated and non-accelerated upload speeds across Amazon S3 regions. The Speed Comparison tool uses multipart uploads to transfer a file from your browser to various Amazon S3 regions with and without using Transfer Acceleration.

You can access the Speed Comparison tool using either of the following methods:

- Copy the following URL into your browser window, replacing *region* with the region that you are using (for example, us-west-2) and *yourBucketName* with the name of the bucket that you want to evaluate:

```
http://s3-accelerate-speedtest.s3-accelerate.amazonaws.com/en/accelerate-speed-comparison.html?region=region&origBucketName=yourBucketName
```

For a list of the regions supported by Amazon S3, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

- Use the Amazon S3 console. For details, see [Enabling Transfer Acceleration](#) in the *Amazon Simple Storage Service Console User Guide*.

Getting Started with Amazon S3 Transfer Acceleration

To get started using Amazon S3 Transfer Acceleration, perform the following steps:

1. **Enable Transfer Acceleration on a bucket** – For your bucket to work with transfer acceleration, the bucket name must conform to DNS naming requirements and must not contain periods (".").

You can enable Transfer Acceleration on a bucket any of the following ways:

- Use the Amazon S3 console. For more information, see [Enabling Transfer Acceleration](#) in the *Amazon Simple Storage Service Console User Guide*.
- Use the REST API [PUT Bucket accelerate](#) operation.
- Use the AWS CLI and AWS SDKs. For more information, see [Using the AWS SDKs, CLI, and Explorers](#) (p. 669).

2. **Transfer data to and from the acceleration-enabled bucket by using one of the following s3-accelerate endpoint domain names:**

- `bucketname.s3-accelerate.amazonaws.com` – to access an acceleration-enabled bucket.
- `bucketname.s3-accelerate.dualstack.amazonaws.com` – to access an acceleration-enabled bucket over IPv6. Amazon S3 dual-stack endpoints support requests to S3 buckets over IPv6 and IPv4. The Transfer Acceleration dual-stack endpoint only uses the virtual hosted-style type of endpoint name. For more information, see [Getting Started Making Requests over IPv6 \(p. 12\)](#) and [Using Amazon S3 Dual-Stack Endpoints \(p. 14\)](#).

Important

Support for the dual-stack accelerated endpoint currently is only available from the AWS Java SDK. Support for the AWS CLI and other AWS SDKs is coming soon.

Note

You can continue to use the regular endpoint in addition to the accelerate endpoints.

You can point your Amazon S3 PUT object and GET object requests to the s3-accelerate endpoint domain name after you enable Transfer Acceleration. For example, let's say you currently have a REST API application using [PUT Object](#) that uses the host name `mybucket.s3.amazonaws.com` in the PUT request. To accelerate the PUT you simply change the host name in your request to `mybucket.s3-accelerate.amazonaws.com`. To go back to using the standard upload speed, simply change the name back to `mybucket.s3.amazonaws.com`.

After Transfer Acceleration is enabled, it can take up to 20 minutes for you to realize the performance benefit. However, the accelerate endpoint will be available as soon as you enable Transfer Acceleration.

You can use the accelerate endpoint in the AWS CLI, AWS SDKs, and other tools that transfer data to and from Amazon S3. If you are using the AWS SDKs, some of the supported languages use an accelerate endpoint client configuration flag so you don't need to explicitly set the endpoint for Transfer Acceleration to `bucketname.s3-accelerate.amazonaws.com`. For examples of how to use an accelerate endpoint client configuration flag, see [Amazon S3 Transfer Acceleration Examples \(p. 76\)](#).

You can use all of the Amazon S3 operations through the transfer acceleration endpoints, except for the following the operations: [GET Service \(list buckets\)](#), [PUT Bucket \(create bucket\)](#), and [DELETE Bucket](#). Also, Amazon S3 Transfer Acceleration does not support cross region copies using [PUT Object - Copy](#).

Requirements for Using Amazon S3 Transfer Acceleration

The following are the requirements for using Transfer Acceleration on an S3 bucket:

- Transfer Acceleration is only supported on virtual style requests. For more information about virtual style requests, see [Making Requests Using the REST API \(p. 44\)](#).
- The name of the bucket used for Transfer Acceleration must be DNS-compliant and must not contain periods (".").
- Transfer Acceleration must be enabled on the bucket. After enabling Transfer Acceleration on a bucket it might take up to 20 minutes before the data transfer speed to the bucket increases.
- To access the bucket that is enabled for Transfer Acceleration, you must use the endpoint `bucketname.s3-accelerate.amazonaws.com` or the dual-stack endpoint `bucketname.s3-accelerate.dualstack.amazonaws.com` to connect to the enabled bucket over IPv6.
- You must be the bucket owner to set the transfer acceleration state. The bucket owner can assign permissions to other users to allow them to set the acceleration state on a bucket. The `s3:PutAccelerateConfiguration` permission permits users to enable or disable Transfer

Acceleration on a bucket. The `s3:GetAccelerateConfiguration` permission permits users to return the Transfer Acceleration state of a bucket, which is either `Enabled` or `Suspended`. For more information about these permissions, see [Permissions Related to Bucket Subresource Operations](#) (p. 347) and [Identity and Access Management in Amazon S3](#) (p. 301).

More Info

- [GET Bucket accelerate](#)
- [PUT Bucket accelerate](#)

Amazon S3 Transfer Acceleration Examples

This section provides examples of how to enable Amazon S3 Transfer Acceleration on a bucket and use the acceleration endpoint for the enabled bucket. Some of the AWS SDK supported languages (for example, Java and .NET) use an accelerate endpoint client configuration flag so you don't need to explicitly set the endpoint for Transfer Acceleration to `bucketname.s3-accelerate.amazonaws.com`. For more information about Transfer Acceleration, see [Amazon S3 Transfer Acceleration](#) (p. 73).

Topics

- [Using the Amazon S3 Console](#) (p. 76)
- [Using Transfer Acceleration from the AWS Command Line Interface \(AWS CLI\)](#) (p. 76)
- [Using Transfer Acceleration from the AWS SDK for Java](#) (p. 77)
- [Using Transfer Acceleration from the AWS SDK for .NET](#) (p. 79)
- [Using Transfer Acceleration from the AWS SDK for JavaScript](#) (p. 80)
- [Using Transfer Acceleration from the AWS SDK for Python \(Boto\)](#) (p. 80)
- [Using Other AWS SDKs](#) (p. 80)

Using the Amazon S3 Console

For information about enabling Transfer Acceleration on a bucket using the Amazon S3 console, see [Enabling Transfer Acceleration](#) in the *Amazon Simple Storage Service Console User Guide*.

Using Transfer Acceleration from the AWS Command Line Interface (AWS CLI)

This section provides examples of AWS CLI commands used for Transfer Acceleration. For instructions on setting up the AWS CLI, see [Setting Up the AWS CLI](#) (p. 675).

Enabling Transfer Acceleration on a Bucket Using the AWS CLI

Use the AWS CLI `put-bucket-accelerate-configuration` command to enable or suspend Transfer Acceleration on a bucket. The following example sets `Status=Enabled` to enable Transfer Acceleration on a bucket. You use `Status=Suspended` to suspend Transfer Acceleration.

Example

```
$ aws s3api put-bucket-accelerate-configuration --bucket bucketname --accelerate-configuration Status=Enabled
```

Using the Transfer Acceleration from the AWS CLI

Setting the configuration value `use_accelerate_endpoint` to `true` in a profile in your AWS Config File will direct all Amazon S3 requests made by `s3` and `s3api` AWS CLI commands to the accelerate endpoint: `s3-accelerate.amazonaws.com`. Transfer Acceleration must be enabled on your bucket to use the accelerate endpoint.

All request are sent using the virtual style of bucket addressing: `my-bucket.s3-accelerate.amazonaws.com`. Any `ListBuckets`, `CreateBucket`, and `DeleteBucket` requests will not be sent to the accelerate endpoint as the endpoint does not support those operations. For more information about `use_accelerate_endpoint`, see [AWS CLI S3 Configuration](#).

The following example sets `use_accelerate_endpoint` to `true` in the default profile.

Example

```
$ aws configure set default.s3.use_accelerate_endpoint true
```

If you want to use the accelerate endpoint for some AWS CLI commands but not others, you can use either one of the following two methods:

- You can use the accelerate endpoint per command by setting the `--endpoint-url` parameter to `https://s3-accelerate.amazonaws.com` or `http://s3-accelerate.amazonaws.com` for any `s3` or `s3api` command.
- You can setup separate profiles in your AWS Config File. For example, create one profile that sets `use_accelerate_endpoint` to `true` and a profile that does not set `use_accelerate_endpoint`. When you execute a command specify which profile you want to use, depending upon whether or not you want to use the accelerate endpoint.

AWS CLI Examples of Uploading an Object to a Bucket Enabled for Transfer Acceleration

The following example uploads a file to a bucket enabled for Transfer Acceleration by using the default profile that has been configured to use the accelerate endpoint.

Example

```
$ aws s3 cp file.txt s3://bucketname/keyname --region region
```

The following example uploads a file to a bucket enabled for Transfer Acceleration by using the `--endpoint-url` parameter to specify the accelerate endpoint.

Example

```
$ aws configure set s3.addressing_style virtual  
$ aws s3 cp file.txt s3://bucketname/keyname --region region --endpoint-url http://s3-accelerate.amazonaws.com
```

Using Transfer Acceleration from the AWS SDK for Java

Example

The following example shows how to use an accelerate endpoint to upload an object to Amazon S3. The example does the following:

- Creates an `AmazonS3Client` that is configured to use accelerate endpoints. All buckets that the client accesses must have transfer acceleration enabled.

- Enables transfer acceleration on a specified bucket. This step is necessary only if the bucket you specify doesn't already have transfer acceleration enabled.
- Verifies that transfer acceleration is enabled for the specified bucket.
- Uploads a new object to the specified bucket using the bucket's accelerate endpoint.

For more information about using Transfer Acceleration, see [Getting Started with Amazon S3 Transfer Acceleration \(p. 74\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketAccelerateConfiguration;
import com.amazonaws.services.s3.model.BucketAccelerateStatus;
import com.amazonaws.services.s3.model.GetBucketAccelerateConfigurationRequest;
import com.amazonaws.services.s3.model.SetBucketAccelerateConfigurationRequest;

public class TransferAcceleration {
    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";

        try {
            // Create an Amazon S3 client that is configured to use the accelerate
            // endpoint.
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .enableAccelerateMode()
                .build();

            // Enable Transfer Acceleration for the specified bucket.
            s3Client.setBucketAccelerateConfiguration(
                new SetBucketAccelerateConfigurationRequest(bucketName,
                    new BucketAccelerateConfiguration(
                        BucketAccelerateStatus.Enabled)));

            // Verify that transfer acceleration is enabled for the bucket.
            String accelerateStatus = s3Client.getBucketAccelerateConfiguration(
                new GetBucketAccelerateConfigurationRequest(bucketName))
                .getStatus();
            System.out.println("Bucket accelerate status: " + accelerateStatus);

            // Upload a new object using the accelerate endpoint.
            s3Client.putObject(bucketName, keyName, "Test object for transfer
            acceleration");
            System.out.println("Object \"" + keyName + "\" uploaded with transfer
            acceleration.");
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```



```
}
```

Using Transfer Acceleration from the AWS SDK for .NET

The following example shows how to use the AWS SDK for .NET to enable Transfer Acceleration on a bucket. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Example

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class TransferAccelerationTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            EnableAccelerationAsync().Wait();
        }

        static async Task EnableAccelerationAsync()
        {
            try
            {
                var putRequest = new PutBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName,
                    AccelerateConfiguration = new AccelerateConfiguration
                    {
                        Status = BucketAccelerateStatus.Enabled
                    }
                };
                await s3Client.PutBucketAccelerateConfigurationAsync(putRequest);

                var getRequest = new GetBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName
                };
                var response = await
s3Client.GetBucketAccelerateConfigurationAsync(getRequest);

                Console.WriteLine("Acceleration state = '{0}' ", response.Status);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine(
                    "Error occurred. Message:'{0}' when setting transfer acceleration",
                    amazonS3Exception.Message);
            }
        }
    }
}
```

When uploading an object to a bucket that has Transfer Acceleration enabled, you specify using acceleration endpoint at the time of creating a client as shown:

```
var client = new AmazonS3Client(new AmazonS3Config
{
    RegionEndpoint = TestRegionEndpoint,
    UseAccelerateEndpoint = true
})
```

Using Transfer Acceleration from the AWS SDK for JavaScript

For an example of enabling Transfer Acceleration by using the AWS SDK for JavaScript, see [Calling the putBucketAccelerateConfiguration operation](#) in the *AWS SDK for JavaScript API Reference*.

Using Transfer Acceleration from the AWS SDK for Python (Boto)

For an example of enabling Transfer Acceleration by using the SDK for Python, see [put_bucket_accelerate_configuration](#) in the *AWS SDK for Python (Boto 3) API Reference*.

Using Other AWS SDKs

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Requester Pays Buckets

Topics

- [Configure Requester Pays by Using the Amazon S3 Console \(p. 81\)](#)
- [Configure Requester Pays with the REST API \(p. 81\)](#)
- [Charge Details \(p. 83\)](#)

In general, bucket owners pay for all Amazon S3 storage and data transfer costs associated with their bucket. A bucket owner, however, can configure a bucket to be a Requester Pays bucket. With Requester Pays buckets, the requester instead of the bucket owner pays the cost of the request and the data download from the bucket. The bucket owner always pays the cost of storing data.

Typically, you configure buckets to be Requester Pays when you want to share data but not incur charges associated with others accessing the data. You might, for example, use Requester Pays buckets when making available large datasets, such as zip code directories, reference data, geospatial information, or web crawling data.

Important

If you enable Requester Pays on a bucket, anonymous access to that bucket is not allowed.

You must authenticate all requests involving Requester Pays buckets. The request authentication enables Amazon S3 to identify and charge the requester for their use of the Requester Pays bucket.

When the requester assumes an AWS Identity and Access Management (IAM) role prior to making their request, the account to which the role belongs is charged for the request. For more information about IAM roles, see [IAM Roles](#) in the *IAM User Guide*.

After you configure a bucket to be a Requester Pays bucket, requesters must include `x-amz-request-payer` in their requests either in the header, for POST, GET and HEAD requests, or as a parameter in a REST request to show that they understand that they will be charged for the request and the data download.

Requester Pays buckets do not support the following.

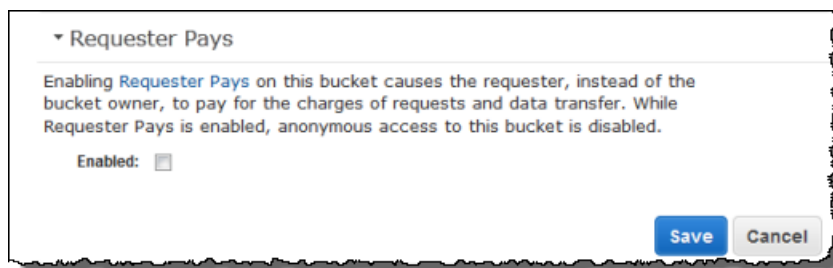
- Anonymous requests
- BitTorrent
- SOAP requests
- You cannot use a Requester Pays bucket as the target bucket for end user logging, or vice versa; however, you can turn on end user logging on a Requester Pays bucket where the target bucket is not a Requester Pays bucket.

Configure Requester Pays by Using the Amazon S3 Console

You can configure a bucket for Requester Pays by using the Amazon S3 console.

To configure a bucket for Requester Pays

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, click the details icon on the left of the bucket name and then click **Properties** to display bucket properties.
3. In the **Properties** pane, click **Requester Pays**.
4. Select the **Enabled** check box.



Configure Requester Pays with the REST API

Topics

- [Setting the requestPayment Bucket Configuration \(p. 81\)](#)
- [Retrieving the requestPayment Configuration \(p. 82\)](#)
- [Downloading Objects in Requester Pays Buckets \(p. 83\)](#)

Setting the requestPayment Bucket Configuration

Only the bucket owner can set the `RequestPaymentConfiguration.payer` configuration value of a bucket to `BucketOwner`, the default, or `Requester`. Setting the `requestPayment` resource is optional. By default, the bucket is not a Requester Pays bucket.

To revert a Requester Pays bucket to a regular bucket, you use the value `BucketOwner`. Typically, you would use `BucketOwner` when uploading data to the Amazon S3 bucket, and then you would set the value to `Requester` before publishing the objects in the bucket.

To set requestPayment

- Use a PUT request to set the Payer value to Requester on a specified bucket.

```
PUT ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Content-Length: 173
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]

<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

If the request succeeds, Amazon S3 returns a response similar to the following.

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Length: 0
Connection: close
Server: AmazonS3
x-amz-request-charged:requester
```

You can set Requester Pays only at the bucket level; you cannot set Requester Pays for specific objects within the bucket.

You can configure a bucket to be BucketOwner or Requester at any time. Realize, however, that there might be a small delay, on the order of minutes, before the new configuration value takes effect.

Note

Bucket owners who give out presigned URLs should think twice before configuring a bucket to be Requester Pays, especially if the URL has a very long lifetime. The bucket owner is charged each time the requester uses a presigned URL that uses the bucket owner's credentials.

Retrieving the requestPayment Configuration

You can determine the Payer value that is set on a bucket by requesting the resource `requestPayment`.

To return the requestPayment resource

- Use a GET request to obtain the `requestPayment` resource, as shown in the following request.

```
GET ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

If the request succeeds, Amazon S3 returns a response similar to the following.

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Type: [type]
Content-Length: [length]
Connection: close
```

```
Server: AmazonS3
```

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

This response shows that the `payer` value is set to `Requester`.

Downloading Objects in Requester Pays Buckets

Because requesters are charged for downloading data from Requester Pays buckets, the requests must contain a special parameter, `x-amz-request-payer`, which confirms that the requester knows he or she will be charged for the download. To access objects in Requester Pays buckets, requests must include one of the following.

- For GET, HEAD, and POST requests, include `x-amz-request-payer : requester` in the header
- For signed URLs, include `x-amz-request-payer=requester` in the request

If the request succeeds and the requester is charged, the response includes the header `x-amz-request-charged:requester`. If `x-amz-request-payer` is not in the request, Amazon S3 returns a 403 error and charges the bucket owner for the request.

Note

Bucket owners do not need to add `x-amz-request-payer` to their requests. Ensure that you have included `x-amz-request-payer` and its value in your signature calculation. For more information, see [Constructing the CanonicalizedAmzHeaders Element](#) (p. 691).

To download objects from a Requester Pays bucket

- Use a GET request to download an object from a Requester Pays bucket, as shown in the following request.

```
GET / [destinationObject] HTTP/1.1
Host: [BucketName].s3.amazonaws.com
x-amz-request-payer : requester
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

If the GET request succeeds and the requester is charged, the response includes `x-amz-request-charged:requester`.

Amazon S3 can return an `Access Denied` error for requests that try to get objects from a Requester Pays bucket. For more information, see [Error Responses](#).

Charge Details

The charge for successful Requester Pays requests is straightforward: the requester pays for the data transfer and the request; the bucket owner pays for the data storage. However, the bucket owner is charged for the request under the following conditions:

- The requester doesn't include the parameter `x-amz-request-payer` in the header (GET, HEAD, or POST) or as a parameter (REST) in the request (HTTP code 403).
- Request authentication fails (HTTP code 403).
- The request is anonymous (HTTP code 403).

- The request is a SOAP request.

Buckets and Access Control

Each bucket has an associated access control policy. This policy governs the creation, deletion and enumeration of objects within the bucket. For more information, see [Identity and Access Management in Amazon S3](#) (p. 301).

Billing and Usage Reporting for S3 Buckets

When using Amazon Simple Storage Service (Amazon S3), you don't have to pay any upfront fees or commit to how much content you'll store. As with the other Amazon Web Services (AWS) services, you pay as you go and pay only for what you use.

AWS provides the following reports for Amazon S3:

- **Billing reports** – Multiple reports that provide high-level views of all of the activity for the AWS services that you're using, including Amazon S3. AWS always bills the owner of the S3 bucket for Amazon S3 fees, unless the bucket was created as a Requester Pays bucket. For more information about Requester Pays, see [Requester Pays Buckets](#) (p. 80). For more information about billing reports, see [AWS Billing Reports for Amazon S3](#) (p. 84).
- **Usage report** – A summary of activity for a specific service, aggregated by hour, day, or month. You can choose which usage type and operation to include. You can also choose how the data is aggregated. For more information, see [AWS Usage Report for Amazon S3](#) (p. 86).

The following topics provide information about billing and usage reporting for Amazon S3.

Topics

- [AWS Billing Reports for Amazon S3](#) (p. 84)
- [AWS Usage Report for Amazon S3](#) (p. 86)
- [Understanding Your AWS Billing and Usage Reports for Amazon S3](#) (p. 87)
- [Using Cost Allocation S3 Bucket Tags](#) (p. 95)

AWS Billing Reports for Amazon S3

Your monthly bill from AWS separates your usage information and cost by AWS service and function. There are several AWS billing reports available, the monthly report, the cost allocation report, and detailed billing reports. For information about how to see your billing reports, see [Viewing Your Bill](#) in the *AWS Billing and Cost Management User Guide*.

You can also download a usage report that gives more detail about your Amazon S3 storage usage than the billing reports. For more information, see [AWS Usage Report for Amazon S3](#) (p. 86).

The following table lists the charges associated with Amazon S3 usage.

Amazon S3 Usage Charges

Charge	Comments
Storage	You pay for storing objects in your S3 buckets. The rate you're charged depends on your objects' size, how long you stored the objects during

Charge	Comments
	the month, and the storage class—STANDARD, INTELLIGENT_TIERING, STANDARD_IA (IA for infrequent access), ONEZONE_IA, GLACIER, DEEP_ARCHIVE or Reduced Redundancy Storage (RRS). For more information about storage classes, see Amazon S3 Storage Classes (p. 103) .
Monitoring and Automation	You pay a monthly monitoring and automation fee per object stored in the INTELLIGENT_TIERING storage class to monitor access patterns and move objects between access tiers in INTELLIGENT_TIERING.
Requests	You pay for requests, for example, GET requests, made against your S3 buckets and objects. This includes lifecycle requests. The rates for requests depend on what kind of request you're making. For information about request pricing, see Amazon S3 Pricing .
Retrievals	You pay for retrieving objects that are stored in STANDARD_IA, ONEZONE_IA, GLACIER and DEEP_ARCHIVE storage.
Early Deletes	If you delete an object stored in INTELLIGENT_TIERING, STANDARD_IA, ONEZONE_IA, GLACIER, or DEEP_ARCHIVE storage before the minimum storage commitment has passed, you pay an early deletion fee for that object.
Storage Management	You pay for the storage management features (Amazon S3 inventory, analytics, and object tagging) that are enabled on your account's buckets.
Bandwidth	<p>You pay for all bandwidth into and out of Amazon S3, except for the following:</p> <ul style="list-style-type: none"> • Data transferred in from the internet • Data transferred out to an Amazon Elastic Compute Cloud (Amazon EC2) instance, when the instance is in the same AWS Region as the S3 bucket • Data transferred out to Amazon CloudFront (CloudFront) <p>You also pay a fee for any data transferred using Amazon S3 Transfer Acceleration.</p>

For detailed information on Amazon S3 usage charges for storage, data transfer, and services, see [Amazon S3 Pricing](#) and the [Amazon S3 FAQ](#).

For information on understanding codes and abbreviations used in the billing and usage reports for Amazon S3, see [Understanding Your AWS Billing and Usage Reports for Amazon S3 \(p. 87\)](#).

More Info

- [AWS Usage Report for Amazon S3 \(p. 86\)](#)
- [Using Cost Allocation S3 Bucket Tags \(p. 95\)](#)
- [AWS Billing and Cost Management](#)
- [Amazon S3 Pricing](#)
- [Amazon S3 FAQ](#)
- [Glacier Pricing](#)

AWS Usage Report for Amazon S3

For more detail about your Amazon S3 storage usage, download dynamically generated AWS usage reports. You can choose which usage type, operation, and time period to include. You can also choose how the data is aggregated.

When you download a usage report, you can choose to aggregate usage data by hour, day, or month. The Amazon S3 usage report lists operations by usage type and AWS Region, for example, the amount of data transferred out of the Asia Pacific (Sydney) Region.

The Amazon S3 usage report includes the following information:

- **Service** – Amazon Simple Storage Service
- **Operation** – The operation performed on your bucket or object. For a detailed explanation of Amazon S3 operations, see [Tracking Operations in Your Usage Reports \(p. 95\)](#).
- **UsageType** – One of the following values:
 - A code that identifies the type of storage
 - A code that identifies the type of request
 - A code that identifies the type of retrieval
 - A code that identifies the type of data transfer
 - A code that identifies early deletions from INTELLIGENT_TIERING, STANDARD_IA, ONEZONE_IA, GLACIER, or DEEP_ARCHIVE storage
 - `StorageObjectCount` – The count of objects stored within a given bucket

For a detailed explanation of Amazon S3 usage types, see [Understanding Your AWS Billing and Usage Reports for Amazon S3 \(p. 87\)](#).

- **Resource** – The name of the bucket associated with the listed usage.
- **StartTime** – Start time of the day that the usage applies to, in Coordinated Universal Time (UTC).
- **EndTime** – End time of the day that the usage applies to, in Coordinated Universal Time (UTC).
- **UsageValue** – One of the following volume values:
 - The number of requests during the specified time period
 - The amount of data transferred, in bytes
 - The amount of data stored, in byte-hours, which is the number of bytes stored in a given hour
 - The amount of data associated with restorations from DEEP_ARCHIVE, GLACIER, STANDARD_IA, or ONEZONE_IA storage, in bytes

Tip

For detailed information about every request that Amazon S3 receives for your objects, turn on server access logging for your buckets. For more information, see [Amazon S3 Server Access Logging \(p. 647\)](#).

You can download a usage report as an XML or a comma-separated values (CSV) file. The following is an example CSV usage report opened in a spreadsheet application.

Service	Operation	UsageType	Resource	StartTime	EndTime	UsageValue
AmazonS3	HeadBucket	USW2-C3DataTransfer-Out-Bytes	admin-created3	6/1/2017 0:00	7/1/2017 0:00	15309
AmazonS3	PutObject	USW2-C3DataTransfer-In-Bytes	admin-created3	6/1/2017 0:00	7/1/2017 0:00	19062
AmazonS3	HeadBucket	USW2-Requests-Tier2	admin-created3	6/1/2017 0:00	7/1/2017 0:00	68
AmazonS3	PutObjectForRepl	USW1-Requests-SIA-Tier1	ca-example-bucket	6/1/2017 0:00	7/1/2017 0:00	178294
AmazonS3	PutObjectForRepl	USW1-USW2-AWS-In-Bytes	ca-example-bucket	6/1/2017 0:00	7/1/2017 0:00	387929083
AmazonS3	GetObjectForRepl	USW2-Requests-NoCharge	admin-created3	6/1/2017 0:00	7/1/2017 0:00	108
AmazonS3	GetObjectForRepl	USW2-USW1-AWS-Out-Bytes	my-test-bucket-bash	6/1/2017 0:00	7/1/2017 0:00	387910021

For information on understanding the usage report, see [Understanding Your AWS Billing and Usage Reports for Amazon S3](#) (p. 87).

Downloading the AWS Usage Report

You can download a usage report as an .xml or a .csv file.

To download the usage report

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the title bar, choose your AWS Identity and Access Management (IAM) user name, and then choose **My Billing Dashboard**.
3. In the navigation pane, choose **Reports**.
4. In the **Other Reports** section, choose **AWS Usage Report**.
5. For **Services**, choose **Amazon Simple Storage Service**.
6. For **Download Usage Report**, choose the following settings:
 - **Usage Types** – For a detailed explanation of Amazon S3 usage types, see [Understanding Your AWS Billing and Usage Reports for Amazon S3](#) (p. 87).
 - **Operation** – For a detailed explanation of Amazon S3 operations, see [Tracking Operations in Your Usage Reports](#) (p. 95).
 - **Time Period** – The time period that you want the report to cover.
 - **Report Granularity** – Whether you want the report to include subtotals by the hour, by the day, or by the month.
7. To choose the format for the report, choose the **Download** for that format, and then follow the prompts to see or save the report.

More Info

- [Understanding Your AWS Billing and Usage Reports for Amazon S3](#) (p. 87)
- [AWS Billing Reports for Amazon S3](#) (p. 84)

Understanding Your AWS Billing and Usage Reports for Amazon S3

Amazon S3 billing and usage reports use codes and abbreviations. For example, for usage type, which is defined in the following table, *region* is replaced with one of the following abbreviations:

- **APE1**: Asia Pacific (Hong Kong)

- **APN1:** Asia Pacific (Tokyo)
- **APN2:** Asia Pacific (Seoul)
- **APS1:** Asia Pacific (Singapore)
- **APS2:** Asia Pacific (Sydney)
- **APS3:** Asia Pacific (Mumbai)
- **CAN1:** Canada (Central)
- **EUN1:** EU (Stockholm)
- **EUC1:** EU (Frankfurt)
- **EU:** EU (Ireland)
- **EUW2:** EU (London)
- **EUW3:** EU (Paris)
- **SAE1:** South America (São Paulo)
- **UGW1:** AWS GovCloud (US-West)
- **UGE1:** AWS GovCloud (US-East)
- **USE1 (or no prefix):** US East (N. Virginia)
- **USE2:** US East (Ohio)
- **USW1:** US West (N. California)
- **USW2:** US West (Oregon)

For information about pricing by AWS Region, see [Amazon S3 Pricing](#).

The first column in the following table lists usage types that appear in your billing and usage reports.

Usage Types

Usage Type	Units	Granularity	Description
<i>region1-region2</i> -AWS-In-Bytes	Bytes	Hourly	The amount of data transferred in to AWS Region1 from AWS Region2
<i>region1-region2</i> -AWS-Out-Bytes	Bytes	Hourly	The amount of data transferred from AWS Region1 to AWS Region2
<i>region</i> -BatchOperations-Jobs	Count	Hourly	The number of Amazon S3 batch operations jobs performed.
<i>region</i> -BatchOperations-Objects	Count	Hourly	The number of object operations performed by Amazon S3 batch operations.
<i>region</i> -DataTransfer-In-Bytes	Bytes	Hourly	The amount of data transferred into Amazon S3 from the internet
<i>region</i> -DataTransfer-Out-Bytes	Bytes	Hourly	The amount of data transferred from Amazon S3 to the internet ¹
<i>region</i> -C3DataTransfer-In-Bytes	Bytes	Hourly	The amount of data transferred into Amazon S3

Usage Type	Units	Granularity	Description
			from Amazon EC2 within the same AWS Region
<i>region</i> -C3DataTransfer-Out-Bytes	Bytes	Hourly	The amount of data transferred from Amazon S3 to Amazon EC2 within the same AWS Region
<i>region</i> -S3G-DataTransfer-In-Bytes	Bytes	Hourly	The amount of data transferred into Amazon S3 to restore objects from GLACIER or DEEP_ARCHIVE storage
<i>region</i> -S3G-DataTransfer-Out-Bytes	Bytes	Hourly	The amount of data transferred from Amazon S3 to transition objects to GLACIER or DEEP_ARCHIVE storage
<i>region</i> -DataTransfer-Regional-Bytes	Bytes	Hourly	The amount of data transferred from Amazon S3 to AWS resources within the same AWS Region
StorageObjectCount	Count	Daily	The number of objects stored within a given bucket
<i>region</i> -CloudFront-In-Bytes	Bytes	Hourly	The amount of data transferred into an AWS Region from a CloudFront distribution
<i>region</i> -CloudFront-Out-Bytes	Bytes	Hourly	The amount of data transferred from an AWS Region to a CloudFront distribution
<i>region</i> -EarlyDelete-ByteHrs	Byte-Hours ²	Hourly	Prorated storage usage for objects deleted from, GLACIER storage before the 90-day minimum commitment ended ³
<i>region</i> -EarlyDelete-GDA	Byte-Hours ²	Hourly	Prorated storage usage for objects deleted from DEEP_ARCHIVE storage before the 180-day minimum commitment ended ³
<i>region</i> -EarlyDelete-SIA	Byte-Hours	Hourly	Prorated storage usage for objects deleted from STANDARD_IA before the 30-day minimum commitment ended ⁴

Usage Type	Units	Granularity	Description
<i>region</i> -EarlyDelete-ZIA	Byte-Hours	Hourly	Prorated storage usage for objects deleted from ONEZONE_IA before the 30-day minimum commitment ended ⁴
<i>region</i> -EarlyDelete-SIA-SmObjects	Byte-Hours	Hourly	Prorated storage usage for small objects (smaller than 128 KB) that were deleted from STANDARD_IA before the 30-day minimum commitment ended ⁴
<i>region</i> -EarlyDelete-ZIA-SmObjects	Byte-Hours	Hourly	Prorated storage usage for small objects (smaller than 128 KB) that were deleted from ONEZONE_IA before the 30-day minimum commitment ended ⁴
<i>region</i> -Inventory-ObjectsListed	Objects	Hourly	The number of objects listed for an object group (objects are grouped by bucket or prefix) with an inventory list
<i>region</i> -Requests-GLACIER-Tier1	Count	Hourly	The number of PUT, COPY, POST, InitiateMultipartUpload, UploadPart, or CompleteMultipartUpload requests on GLACIER objects
<i>region</i> -Requests-GLACIER-Tier2	Count	Hourly	The number of GET and all other requests not listed on GLACIER objects
<i>region</i> -Requests-SIA-Tier1	Count	Hourly	The number of PUT, COPY, POST, or LIST requests on STANDARD_IA objects
<i>region</i> -Requests-ZIA-Tier1	Count	Hourly	The number of PUT, COPY, POST, or LIST requests on ONEZONE_IA objects
<i>region</i> -Requests-SIA-Tier2	Count	Hourly	The number of GET and all other non-SIA-Tier1 requests on STANDARD_IA objects
<i>region</i> -Requests-ZIA-Tier2	Count	Hourly	The number of GET and all other non-ZIA-Tier1 requests on ONEZONE_IA objects

Usage Type	Units	Granularity	Description
<i>region</i> -Requests-Tier1	Count	Hourly	The number of PUT, COPY, POST, or LIST requests for STANDARD, RRS, and tags
<i>region</i> -Requests-Tier2	Count	Hourly	The number of GET and all other non-Tier1 requests
<i>region</i> -Requests-Tier3	Count	Hourly	The number of lifecycle requests to GLACIER or DEEP_ARCHIVE and standard GLACIER restore requests
<i>region</i> -Requests-Tier4	Count	Hourly	The number of lifecycle transitions to INTELLIGENT_TIERING, STANDARD_IA, or ONEZONE_IA storage
<i>region</i> -Requests-Tier5	Count	Hourly	The number of Bulk GLACIER restore requests
<i>region</i> -Requests-GDA-Tier1	Count	Hourly	The number of PUT, COPY, POST, InitiateMultipartUpload, UploadPart, or CompleteMultipartUpload requests on DEEP Archive objects
<i>region</i> -Requests-GDA-Tier2	Count	Hourly	The number of GET, HEAD, and LIST requests
<i>region</i> -Requests-GDA-Tier3	Count	Hourly	The number of DEEP_ARCHIVE standard restore requests
<i>region</i> -Requests-GDA-Tier5	Count	Hourly	The number of Bulk DEEP_ARCHIVE restore requests
<i>region</i> -Requests-Tier6	Count	Hourly	The number of Expedited GLACIER restore requests
<i>region</i> -Bulk-Retrieval-Bytes	Bytes	Hourly	The number of bytes of data retrieved with Bulk GLACIER or DEEP_ARCHIVE requests
<i>region</i> -Requests-INT-Tier1	Count	Hourly	The number of PUT, COPY, POST, or LIST requests on INTELLIGENT_TIERING objects

Usage Type	Units	Granularity	Description
<i>region</i> -Requests-INT-Tier2	Count	Hourly	The number of GET and all other non-Tier1 requests for INTELLIGENT_TIERING objects
<i>region</i> -Select-Returned-INT-Bytes	Bytes	Hourly	The number of bytes of data returned with Select requests from INTELLIGENT_TIERING storage
<i>region</i> -Select-Scanned-INT-Bytes	Bytes	Hourly	The number of bytes of data scanned with Select requests from INTELLIGENT_TIERING storage
<i>region</i> -EarlyDelete-INT	Byte-Hours	Hourly	Prorated storage usage for objects deleted from INTELLIGENT_TIERING before the 30-day minimum commitment ended
<i>region</i> -Monitoring-Automation-INT	Objects	Hourly	The number of unique objects monitored and auto-tiered in the INTELLIGENT_TIERING storage class
<i>region</i> -Expedited-Retrieval-Bytes	Bytes	Hourly	The number of bytes of data retrieved with Expedited GLACIER requests
<i>region</i> -Standard-Retrieval-Bytes	Bytes	Hourly	The number of bytes of data retrieved with standard GLACIER or DEEP_ARCHIVE requests
<i>region</i> -Retrieval-SIA	Bytes	Hourly	The number of bytes of data retrieved from STANDARD_IA storage
<i>region</i> -Retrieval-ZIA	Bytes	Hourly	The number of bytes of data retrieved from ONEZONE_IA storage
<i>region</i> -StorageAnalytics-ObjCount	Objects	Hourly	The number of unique objects in each object group (where objects are grouped by bucket or prefix) tracked by storage analytics

Usage Type	Units	Granularity	Description
<i>region</i> -Select-Scanned-Bytes	Bytes	Hourly	The number of bytes of data scanned with Select requests from STANDARD storage
<i>region</i> -Select-Scanned-SIA-Bytes	Bytes	Hourly	The number of bytes of data scanned with Select requests from STANDARD_IA storage
<i>region</i> -Select-Scanned-ZIA-Bytes	Bytes	Hourly	The number of bytes of data scanned with Select requests from ONEZONE_IA storage
<i>region</i> -Select-Returned-Bytes	Bytes	Hourly	The number of bytes of data returned with Select requests from STANDARD storage
<i>region</i> -Select-Returned-SIA-Bytes	Bytes	Hourly	The number of bytes of data returned with Select requests from STANDARD_IA storage
<i>region</i> -Select-Returned-ZIA-Bytes	Bytes	Hourly	The number of bytes of data returned with Select requests from ONEZONE_IA storage
<i>region</i> -TagStorage-TagHrs	Tag-Hours	Daily	The total of tags on all objects in the bucket reported by hour
<i>region</i> -TimedStorage-ByteHrs	Byte-Hours	Daily	The number of byte-hours that data was stored in STANDARD storage
<i>region</i> -TimedStorage-GLACIERByteHrs	Byte-Hours	Daily	The number of byte-hours that data was stored in GLACIER storage
<i>region</i> -TimedStorage-GlacierStaging	Byte-Hours	Daily	The number of byte-hours that data was stored in GLACIER staging storage
<i>region</i> -TimedStorage-GDA-ByteHrs	Byte-Hours	Daily	The number of byte-hours that data was stored in DEEP_ARCHIVE storage
<i>region</i> -TimedStorage-GDA-Staging	Byte-Hours	Daily	The number of byte-hours that data was stored in DEEP_ARCHIVE staging storage

Usage Type	Units	Granularity	Description
<i>region</i> -TimedStorage-INT-FA-ByteHrs	Byte-Hours	Daily	The number of byte-hours that data was stored in the frequent access tier of INTELLIGENT_TIERING storage
<i>region</i> -TimedStorage-INT-IA-ByteHrs	Byte-Hours	Daily	The number of byte-hours that data was stored in the infrequent access tier of INTELLIGENT_TIERING storage
<i>region</i> -TimedStorage-RRS-ByteHrsb	Byte-Hours	Daily	The number of byte-hours that data was stored in Reduced Redundancy Storage (RRS) storage
<i>region</i> -TimedStorage-SIA-ByteHrs	Byte-Hours	Daily	The number of byte-hours that data was stored in STANDARD_IA storage
<i>region</i> -TimedStorage-ZIA-ByteHrs	Byte-Hours	Daily	The number of byte-hours that data was stored in ONEZONE_IA storage
<i>region</i> -TimedStorage-SIA-SmObjects	Byte-Hours	Daily	The number of byte-hours that small objects (smaller than 128 KB) were stored in STANDARD_IA storage
<i>region</i> -TimedStorage-ZIA-SmObjects	Byte-Hours	Daily	The number of byte-hours that small objects (smaller than 128 KB) were stored in ONEZONE_IA storage

Notes:

1. If you terminate a transfer before completion, the amount of data that is transferred might exceed the amount of data that your application receives. This discrepancy can occur because a transfer termination request cannot be executed instantaneously, and some amount of data might be in transit pending execution of the termination request. This data in transit is billed as data transferred "out."
2. For more information on the byte-hours unit, see [Converting Usage Byte-Hours to Billed GB-Months \(p. 95\)](#).
3. When objects that are archived to the GLACIER or DEEP_ARCHIVE storage class are deleted, overwritten, or transitioned to a different storage class before the minimum storage commitment has passed, which is 90 days for GLACIER or 180-days for DEEP_ARCHIVE, there is a prorated charge per gigabyte for the remaining days.
4. For objects that are in INTELLIGENT_TIERING, STANDARD_IA, or ONEZONE_IA storage, when they are deleted, overwritten, or transitioned to a different storage class prior to 30 days, there is a prorated charge per gigabyte for the remaining days.
5. For small objects (smaller than 128 KB) that are in STANDARD_IA or ONEZONE_IA storage, when they are deleted, overwritten, or transitioned to a different storage class prior to 30 days, there is a prorated charge per gigabyte for the remaining days.

6. There is no minimum billable object size for objects in the `INTELLIGENT_TIERING` storage class, but objects that are smaller than 128 KB are not eligible for auto-tiering and are always charged at the rate for the `INTELLIGENT_TIERING` frequent access tier.

Tracking Operations in Your Usage Reports

Operations describe the action taken on your AWS object or bucket by the specified usage type. Operations are indicated by self-explanatory codes, such as `PutObject` or `ListBucket`. To see which actions on your bucket generated a specific type of usage, use these codes. When you create a usage report, you can choose to include **All Operations**, or a specific operation, for example, **GetObject**, to report on.

Converting Usage Byte-Hours to Billed GB-Months

The volume of storage that we bill you for each month is based on the average amount of storage you used throughout the month. You are billed for all of the object data and metadata stored in buckets that you created under your AWS account. For more information about metadata, see [Object Key and Metadata](#) (p. 99).

We measure your storage usage in `TimedStorage-ByteHrs`, which are totaled up at the end of the month to generate your monthly charges. The usage report reports your storage usage in byte-hours and the billing reports report storage usage in GB-months. To correlate your usage report to your billing reports, you need to convert byte-hours into GB-months.

For example, if you store 100 GB (107,374,182,400 bytes) of `STANDARD` Amazon S3 storage data in your bucket for the first 15 days in March, and 100 TB (109,951,162,777,600 bytes) of `STANDARD` Amazon S3 storage data for the final 16 days in March, you will have used 42,259,901,212,262,400 byte-hours.

First, calculate the total byte-hour usage:

```
[107,374,182,400 bytes x 15 days x (24 hours/day)]  
+ [109,951,162,777,600 bytes x 16 days x (24 hours/day)]  
= 42,259,901,212,262,400 byte-hours
```

Then convert the byte-hours to GB-Months:

```
42,259,901,212,262,400 byte-hours / 1,073,741,824 bytes per GB / 24 hours per day  
/ 31 days in March  
= 52,900 GB-Months
```

More Info

- [AWS Usage Report for Amazon S3](#) (p. 86)
- [AWS Billing Reports for Amazon S3](#) (p. 84)
- [Amazon S3 Pricing](#)
- [Amazon S3 FAQ](#)
- [Glacier Pricing](#)
- [Glacier FAQs](#)

Using Cost Allocation S3 Bucket Tags

To track the storage cost or other criteria for individual projects or groups of projects, label your Amazon S3 buckets using cost allocation tags. A *cost allocation tag* is a key-value pair that you associate with an

S3 bucket. After you activate cost allocation tags, AWS uses the tags to organize your resource costs on your cost allocation report. Cost allocation tags can only be used to label buckets. For information about tags used for labeling objects, see [Object Tagging \(p. 110\)](#).

The *cost allocation report* lists the AWS usage for your account by product category and AWS Identity and Access Management (IAM) user. The report contains the same line items as the detailed billing report (see [Understanding Your AWS Billing and Usage Reports for Amazon S3 \(p. 87\)](#)) and additional columns for your tag keys.

AWS provides two types of cost allocation tags, an AWS-generated tag and user-defined tags. AWS defines, creates, and applies the AWS-generated `createdBy` tag for you after an Amazon S3 CreateBucket event. You define, create, and apply *user-defined* tags to your S3 bucket.

You must activate both types of tags separately in the Billing and Cost Management console before they can appear in your billing reports. For more information about AWS-generated tags, see [AWS-Generated Cost Allocation Tags](#). For more information about activating tags, see [Using Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*.

A user-defined cost allocation tag has the following components:

- The tag key. The tag key is the name of the tag. For example, in the tag `project/Trinity`, `project` is the key. The tag key is a case-sensitive string that can contain 1 to 128 Unicode characters.
- The tag value. The tag value is a required string. For example, in the tag `project/Trinity`, `Trinity` is the value. The tag value is a case-sensitive string that can contain from 0 to 256 Unicode characters.

For details on the allowed characters for user-defined tags and other restrictions, see [User-Defined Tag Restrictions](#) in the *AWS Billing and Cost Management User Guide*.

Each S3 bucket has a tag set. A *tag set* contains all of the tags that are assigned to that bucket. A tag set can contain as many as 50 tags, or it can be empty. Keys must be unique within a tag set, but values in a tag set don't have to be unique. For example, you can have the same value in tag sets named `project/Trinity` and `cost-center/Trinity`.

Within a bucket, if you add a tag that has the same key as an existing tag, the new value overwrites the old value.

AWS doesn't apply any semantic meaning to your tags. We interpret tags strictly as character strings.

To add, list, edit, or delete tags, you can use the Amazon S3 console, the AWS Command Line Interface (AWS CLI), or the Amazon S3 API.

For more information about creating tags, see the appropriate topic:

- To create tags in the console, see [How Do I View the Properties for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.
- To create tags using the Amazon S3 API, see [PUT Bucket tagging](#) in the *Amazon Simple Storage Service API Reference*.
- To create tags using the AWS CLI, see [put-bucket-tagging](#) in the AWS CLI Command Reference.

For more information about user-defined tags, see [User-Defined Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*.

More Info

- [Using Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*

- [Understanding Your AWS Billing and Usage Reports for Amazon S3 \(p. 87\)](#)
- [AWS Billing Reports for Amazon S3 \(p. 84\)](#)

Working with Amazon S3 Objects

Amazon S3 is a simple key, value store designed to store as many objects as you want. You store these objects in one or more buckets. An object consists of the following:

- **Key** – The name that you assign to an object. You use the object key to retrieve the object.

For more information, see [Object Key and Metadata \(p. 99\)](#).

- **Version ID** – Within a bucket, a key and version ID uniquely identify an object.

The version ID is a string that Amazon S3 generates when you add an object to a bucket. For more information, see [Object Versioning \(p. 108\)](#).

- **Value** – The content that you are storing.

An object value can be any sequence of bytes. Objects can range in size from zero to 5 TB. For more information, see [Uploading Objects \(p. 169\)](#).

- **Metadata** – A set of name-value pairs with which you can store information regarding the object.

You can assign metadata, referred to as user-defined metadata, to your objects in Amazon S3. Amazon S3 also assigns system-metadata to these objects, which it uses for managing objects. For more information, see [Object Key and Metadata \(p. 99\)](#).

- **Subresources** – Amazon S3 uses the subresource mechanism to store object-specific additional information.

Because subresources are subordinates to objects, they are always associated with some other entity such as an object or a bucket. For more information, see [Object Subresources \(p. 108\)](#).

- **Access Control Information** – You can control access to the objects you store in Amazon S3.

Amazon S3 supports both the resource-based access control, such as an access control list (ACL) and bucket policies, and user-based access control. For more information, see [Identity and Access Management in Amazon S3 \(p. 301\)](#).

For more information about working with objects, see the following sections. Your Amazon S3 resources (for example buckets and objects) are private by default. You need to explicitly grant permission for others to access these resources. For example, you might want to share a video or a photo stored in your Amazon S3 bucket on your website. That works only if you either make the object public or use a presigned URL on your website. For more information about sharing objects, see [Share an Object with Others \(p. 167\)](#).

Topics

- [Object Key and Metadata \(p. 99\)](#)
- [Amazon S3 Storage Classes \(p. 103\)](#)
- [Object Subresources \(p. 108\)](#)
- [Object Versioning \(p. 108\)](#)
- [Object Tagging \(p. 110\)](#)
- [Object Lifecycle Management \(p. 119\)](#)
- [Cross-Origin Resource Sharing \(CORS\) \(p. 151\)](#)

- [Operations on Objects \(p. 160\)](#)

Object Key and Metadata

Each Amazon S3 object has data, a key, and metadata. Object key (or key name) uniquely identifies the object in a bucket. Object metadata is a set of name-value pairs. You can set object metadata at the time you upload it. After you upload the object, you cannot modify object metadata. The only way to modify object metadata is to make a copy of the object and set the metadata.

Topics

- [Object Keys \(p. 99\)](#)
- [Object Metadata \(p. 101\)](#)

Object Keys

When you create an object, you specify the key name, which uniquely identifies the object in the bucket. For example, in the Amazon S3 console (see [AWS Management Console](#)), when you highlight a bucket, a list of objects in your bucket appears. These names are the object keys. The name for a key is a sequence of Unicode characters whose UTF-8 encoding is at most 1024 bytes long.

The Amazon S3 data model is a flat structure: you create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders. However, you can infer logical hierarchy using key name prefixes and delimiters as the Amazon S3 console does. The Amazon S3 console supports a concept of folders. Suppose that your bucket (admin-created) has four objects with the following object keys:

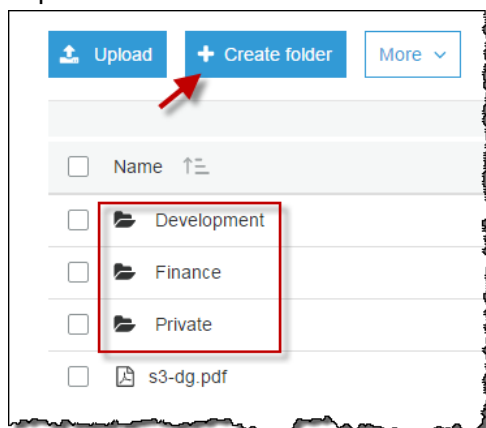
Development/Projects.xls

Finance/statement1.pdf

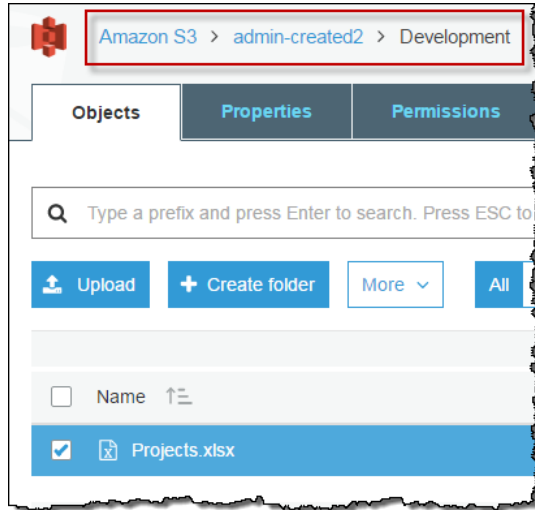
Private/taxdocument.pdf

s3-dg.pdf

The console uses the key name prefixes (Development/, Finance/, and Private/) and delimiter ('/') to present a folder structure as shown:



The s3-dg.pdf key does not have a prefix, so its object appears directly at the root level of the bucket. If you open the Development/ folder, you see the Projects.xlsx object in it.



Note

Amazon S3 supports buckets and objects, and there is no hierarchy in Amazon S3. However, the prefixes and delimiters in an object key name enable the Amazon S3 console and the AWS SDKs to infer hierarchy and introduce the concept of folders.

Object Key Naming Guidelines

You can use any UTF-8 character in an object key name. However, using certain characters in key names may cause problems with some applications and protocols. The following guidelines help you maximize compliance with DNS, web-safe characters, XML parsers, and other APIs.

Safe Characters

The following character sets are generally safe for use in key names:

Alphanumeric characters

- 0-9
- a-z
- A-Z

Special characters

- !
- -
- _
- .
- *
- ' (
-)

The following are examples of valid object key names:

- 4my-organization
- my.great_photos-2014/jan/myvacation.jpg
- videos/2014/birthday/video1.wmv

Important

If an object key name consists of a single period (.), or two periods (..), you can't download the object using the Amazon S3 console. To download an object with a key name of "." or "..", you must use the AWS CLI, AWS SDKs, or REST API.

Characters That Might Require Special Handling

The following characters in a key name might require additional code handling and likely need to be URL encoded or referenced as HEX. Some of these are non-printable characters and your browser might not handle them, which also requires special handling:

- Ampersand ("&")
- Dollar ("\$")
- ASCII character ranges 00–1F hex (0–31 decimal) and 7F (127 decimal)
- 'At' symbol ("@")
- Equals ("=")
- Semicolon (";")
- Colon (":")
- Plus ("+")
- Space – Significant sequences of spaces may be lost in some uses (especially multiple spaces)
- Comma (",")
- Question mark ("?")

Characters to Avoid

Avoid the following characters in a key name because of significant special handling for consistency across all applications.

- Backslash ("\")
- Left curly brace ("{"
- Non-printable ASCII characters (128–255 decimal characters)
- Caret ("^")
- Right curly brace ("}")
- Percent character ("%")
- Grave accent / back tick ("`")
- Right square bracket ("]")
- Quotation marks
- 'Greater Than' symbol (">")
- Left square bracket ("["
- Tilde ("~")
- 'Less Than' symbol ("<")
- 'Pound' character ("#")
- Vertical bar / pipe ("|")

Object Metadata

There are two kinds of metadata: system metadata and user-defined metadata.

System-Defined Object Metadata

For each object stored in a bucket, Amazon S3 maintains a set of system metadata. Amazon S3 processes this system metadata as needed. For example, Amazon S3 maintains object creation date and size metadata and uses this information as part of object management.

There are two categories of system metadata:

1. Metadata such as object creation date is system controlled where only Amazon S3 can modify the value.
2. Other system metadata, such as the storage class configured for the object and whether the object has server-side encryption enabled, are examples of system metadata whose values you control. If your bucket is configured as a website, sometimes you might want to redirect a page request to another page or an external URL. In this case, a webpage is an object in your bucket. Amazon S3 stores the page redirect value as system metadata whose value you control.

When you create objects, you can configure values of these system metadata items or update the values when you need to. For more information about storage classes, see [Amazon S3 Storage Classes \(p. 103\)](#). For more information about server-side encryption, see [Protecting Data Using Encryption \(p. 264\)](#).

The following table provides a list of system-defined metadata and whether you can update it.

Name	Description	Can User Modify the Value?
Date	Current date and time.	No
Content-Length	Object size in bytes.	No
Last-Modified	Object creation date or the last modified date, whichever is the latest.	No
Content-MD5	The base64-encoded 128-bit MD5 digest of the object.	No
x-amz-server-side-encryption	Indicates whether server-side encryption is enabled for the object, and whether that encryption is from the AWS Key Management Service (SSE-KMS) or from AWS managed encryption (SSE-S3). For more information, see Protecting Data Using Server-Side Encryption (p. 265) .	Yes
x-amz-version-id	Object version. When you enable versioning on a bucket, Amazon S3 assigns a version number to objects added to the bucket. For more information, see Using Versioning (p. 432) .	No
x-amz-delete-marker	In a bucket that has versioning enabled, this Boolean marker indicates whether the object is a delete marker.	No
x-amz-storage-class	Storage class used for storing the object. For more information, see Amazon S3 Storage Classes (p. 103) .	Yes
x-amz-website-redirect-location	Redirects requests for the associated object to another object in the same bucket or an external URL. For more information, see (Optional) Configuring a Webpage Redirect (p. 510) .	Yes

Name	Description	Can User Modify the Value?
x-amz-server-side-encryption-aws-kms-key-id	If <code>x-amz-server-side-encryption</code> is present and has the value of <code>aws:kms</code> , this indicates the ID of the AWS Key Management Service (AWS KMS) master encryption key that was used for the object.	Yes
x-amz-server-side-encryption-customer-algorithm	Indicates whether server-side encryption with customer-provided encryption keys (SSE-C) is enabled. For more information, see Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys (SSE-C) (p. 279).	Yes

User-Defined Object Metadata

When uploading an object, you can also assign metadata to the object. You provide this optional information as a name-value (key-value) pair when you send a PUT or POST request to create the object. When you upload objects using the REST API, the optional user-defined metadata names must begin with "x-amz-meta-" to distinguish them from other HTTP headers. When you retrieve the object using the REST API, this prefix is returned. When you upload objects using the SOAP API, the prefix is not required. When you retrieve the object using the SOAP API, the prefix is removed, regardless of which API you used to upload the object.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

When metadata is retrieved through the REST API, Amazon S3 combines headers that have the same name (ignoring case) into a comma-delimited list. If some metadata contains unprintable characters, it is not returned. Instead, the `x-amz-missing-meta` header is returned with a value of the number of unprintable metadata entries.

User-defined metadata is a set of key-value pairs. Amazon S3 stores user-defined metadata keys in lowercase. Each key-value pair must conform to US-ASCII when you are using REST and to UTF-8 when you are using SOAP or browser-based uploads via POST.

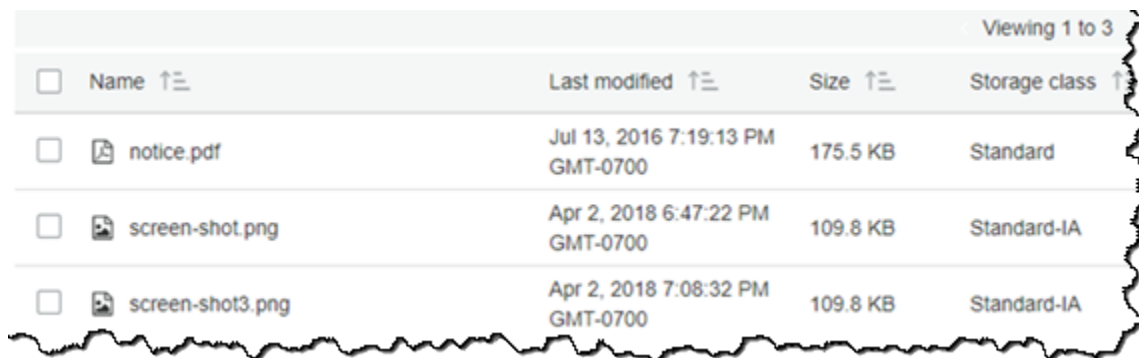
Note

The PUT request header is limited to 8 KB in size. Within the PUT request header, the user-defined metadata is limited to 2 KB in size. The size of user-defined metadata is measured by taking the sum of the number of bytes in the UTF-8 encoding of each key and value.




For information about adding metadata to your object after it's been uploaded, see [How Do I Add Metadata to an S3 Object?](#) in the *Amazon Simple Storage Service Console User Guide*.

Amazon S3 Storage Classes

Each object in Amazon S3 has a storage class associated with it. For example, if you list the objects in an S3 bucket, the console shows the storage class for all the objects in the list.



The screenshot shows the Amazon S3 console interface. At the top right, it says 'Viewing 1 to 3'. Below this is a table with four columns: 'Name', 'Last modified', 'Size', and 'Storage class'. Each row has a checkbox on the left. The first row is for 'notice.pdf' with a last modified date of 'Jul 13, 2016 7:19:13 PM GMT-0700', a size of '175.5 KB', and a storage class of 'Standard'. The second row is for 'screen-shot.png' with a last modified date of 'Apr 2, 2018 6:47:22 PM GMT-0700', a size of '109.8 KB', and a storage class of 'Standard-IA'. The third row is for 'screen-shot3.png' with a last modified date of 'Apr 2, 2018 7:08:32 PM GMT-0700', a size of '109.8 KB', and a storage class of 'Standard-IA'.

<input type="checkbox"/>	Name ↑	Last modified ↑	Size ↑	Storage class ↑
<input type="checkbox"/>	 notice.pdf	Jul 13, 2016 7:19:13 PM GMT-0700	175.5 KB	Standard
<input type="checkbox"/>	 screen-shot.png	Apr 2, 2018 6:47:22 PM GMT-0700	109.8 KB	Standard-IA
<input type="checkbox"/>	 screen-shot3.png	Apr 2, 2018 7:08:32 PM GMT-0700	109.8 KB	Standard-IA

Amazon S3 offers a range of storage classes for the objects that you store. You choose a class depending on your use case scenario and performance access requirements. All of these storage classes offer high durability.

Topics

- [Storage Classes for Frequently Accessed Objects \(p. 104\)](#)
- [Storage Class That Automatically Optimizes Frequently and Infrequently Accessed Objects \(p. 104\)](#)
- [Storage Classes for Infrequently Accessed Objects \(p. 105\)](#)
- [Storage Classes for Archiving Objects \(p. 106\)](#)
- [Comparing the Amazon S3 Storage Classes \(p. 106\)](#)
- [Setting the Storage Class of an Object \(p. 107\)](#)

Storage Classes for Frequently Accessed Objects

For performance-sensitive use cases (those that require millisecond access time) and frequently accessed data, Amazon S3 provides the following storage classes:

- **STANDARD**—The default storage class. If you don't specify the storage class when you upload an object, Amazon S3 assigns the STANDARD storage class.
- **REDUCED_REDUNDANCY**—The Reduced Redundancy Storage (RRS) storage class is designed for noncritical, reproducible data that can be stored with less redundancy than the STANDARD storage class.

Important

We recommend that you not use this storage class. The STANDARD storage class is more cost effective.

For durability, RRS objects have an average annual expected loss of 0.01% of objects. If an RRS object is lost, when requests are made to that object, Amazon S3 returns a 405 error.

Storage Class That Automatically Optimizes Frequently and Infrequently Accessed Objects

The **INTELLIGENT_TIERING** storage class is designed to optimize storage costs by automatically moving data to the most cost-effective storage access tier, without performance impact or operational overhead. INTELLIGENT_TIERING delivers automatic cost savings by moving data on a granular object level

between two access tiers, a frequent access tier and a lower-cost infrequent access tier, when access patterns change. The `INTELLIGENT_TIERING` storage class is ideal if you want to optimize storage costs automatically for long-lived data when access patterns are unknown or unpredictable.

The `INTELLIGENT_TIERING` storage class stores objects in two access tiers: one tier that is optimized for frequent access and another lower-cost tier that is optimized for infrequently accessed data. For a small monthly monitoring and automation fee per object, Amazon S3 monitors access patterns of the objects in the `INTELLIGENT_TIERING` storage class and moves objects that have not been accessed for 30 consecutive days to the infrequent access tier. There are no retrieval fees when using the `INTELLIGENT_TIERING` storage class. If an object in the infrequent access tier is accessed, it is automatically moved back to the frequent access tier. No additional tiering fees apply when objects are moved between access tiers within the `INTELLIGENT_TIERING` storage class.

Note

The `INTELLIGENT_TIERING` storage class is suitable for objects larger than 128 KB that you plan to store for at least 30 days. If the size of an object is less than 128 KB, it is not eligible for auto-tiering. Smaller objects can be stored, but they are always charged at the frequent access tier rates in the `INTELLIGENT_TIERING` storage class. If you delete an object before the end of the 30-day minimum storage duration period, you are charged for 30 days. For pricing information, see [Amazon S3 Pricing](#).

Storage Classes for Infrequently Accessed Objects

The `STANDARD_IA` and `ONEZONE_IA` storage classes are designed for long-lived and infrequently accessed data. (IA stands for infrequent access.) `STANDARD_IA` and `ONEZONE_IA` objects are available for millisecond access (similar to the `STANDARD` storage class). Amazon S3 charges a retrieval fee for these objects, so they are most suitable for infrequently accessed data. For pricing information, see [Amazon S3 Pricing](#).

For example, you might choose the `STANDARD_IA` and `ONEZONE_IA` storage classes:

- For storing backups.
- For older data that is accessed infrequently, but that still requires millisecond access. For example, when you upload data, you might choose the `STANDARD` storage class, and use lifecycle configuration to tell Amazon S3 to transition the objects to the `STANDARD_IA` or `ONEZONE_IA` class. For more information about lifecycle management, see [Object Lifecycle Management \(p. 119\)](#).

Note

The `STANDARD_IA` and `ONEZONE_IA` storage classes are suitable for objects larger than 128 KB that you plan to store for at least 30 days. If an object is less than 128 KB, Amazon S3 charges you for 128 KB. If you delete an object before the end of the 30-day minimum storage duration period, you are charged for 30 days. For pricing information, see [Amazon S3 Pricing](#).

These storage classes differ as follows:

- `STANDARD_IA`—Amazon S3 stores the object data redundantly across multiple geographically separated Availability Zones (similar to the `STANDARD` storage class). `STANDARD_IA` objects are resilient to the loss of an Availability Zone. This storage class offers greater availability and resiliency than the `ONEZONE_IA` class.
- `ONEZONE_IA`—Amazon S3 stores the object data in only one Availability Zone, which makes it less expensive than `STANDARD_IA`. However, the data is not resilient to the physical loss of the Availability Zone resulting from disasters, such as earthquakes and floods. The `ONEZONE_IA` storage class is

as durable as STANDARD_IA, but it is less available and less resilient. For a comparison of storage class durability and availability, see the Durability and Availability table at the end of this section. For pricing, see [Amazon S3 Pricing](#).

We recommend the following:

- STANDARD_IA—Use for your primary or only copy of data that can't be recreated.
- ONEZONE_IA—Use if you can recreate the data if the Availability Zone fails, and for object replicas when setting cross-region replication (CRR).

Storage Classes for Archiving Objects

The **GLACIER** and **DEEP_ARCHIVE** storage classes are designed for low-cost data archiving. These storage classes offer the same durability and resiliency as the STANDARD storage class. For a comparison of storage class durability and availability, see the Durability and Availability table at the end of this section.

These storage classes differ as follows:

- GLACIER—Use for archives where portions of the data might need to be retrieved in minutes. Data stored in the GLACIER storage class has a minimum storage duration period of 90 days and can be accessed in as little as 1-5 minutes using expedited retrieval. If you have deleted, overwritten, or transitioned to a different storage class an object before the 90-day minimum, you are charged for 90 days. For pricing information, see [Amazon S3 Pricing](#).
- DEEP_ARCHIVE—Use for archiving data that rarely needs to be accessed. Data stored in the DEEP_ARCHIVE storage class has a minimum storage duration period of 180 days and a default retrieval time of 12 hours. If you have deleted, overwritten, or transitioned to a different storage class an object before the 180-day minimum, you are charged for 180 days. For pricing information, see [Amazon S3 Pricing](#).

DEEP_ARCHIVE is the lowest cost storage option in AWS. Storage costs for DEEP_ARCHIVE are less expensive than using the GLACIER storage class. You can reduce DEEP_ARCHIVE retrieval costs by using bulk retrieval, which returns data within 48 hours.

Retrieving Archived Objects

You can set the storage class of an object to GLACIER or DEEP_ARCHIVE in the same ways that you do for the other storage classes as described in the section [Setting the Storage Class of an Object \(p. 107\)](#). However, the GLACIER and DEEP_ARCHIVE objects are not available for real-time access. You must first restore the GLACIER and DEEP_ARCHIVE objects before you can access them (STANDARD, RRS, STANDARD_IA, ONEZONE_IA, and INTELLIGENT_TIERING objects are available for anytime access). For more information about retrieving archived objects, see [Restoring Archived Objects \(p. 248\)](#).

Important

When you choose the GLACIER or DEEP_ARCHIVE storage class, your objects remain in Amazon S3. You cannot access them directly through the separate Amazon S3 Glacier service.

To learn more about the Amazon S3 Glacier service, see the [Amazon S3 Glacier Developer Guide](#).

Comparing the Amazon S3 Storage Classes

The following table compares the storage classes.

Storage Class	Designed for	Durability (designed for)	Availability (designed for)	Availability Zones	Min storage duration	Min billable object size	Other Considerations
STANDARD	Frequently accessed data	99.99999999%	99.99%	>= 3	None	None	None
STANDARD_IA	Long-lived, infrequently accessed data	99.99999999%	99.9%	>= 3	30 days	128 KB	Per GB retrieval fees apply.
INTELLIGENT_TIERING	Long-lived data with changing or unknown access patterns	99.99999999%	99.9%	>= 3	30 days	None	Monitoring and automation fees per object apply. No retrieval fees.
ONEZONE_IA	Long-lived, infrequently accessed, non-critical data	99.99999999%	99.5%	1	30 days	128 KB	Per GB retrieval fees apply. Not resilient to the loss of the Availability Zone.
GLACIER	Long-term data archiving with retrieval times ranging from minutes to hours	99.99999999%	99.99% (after you restore objects)	>= 3	90 days	None	Per GB retrieval fees apply. You must first restore archived objects before you can access them. For more information, see Restoring Archived Objects .
DEEP_ARCHIVE	Archiving rarely accessed data with a default retrieval time of 12 hours	99.99999999%	99.99% (after you restore objects)	>= 3	180 days	None	Per GB retrieval fees apply. You must first restore archived objects before you can access them. For more information, see Restoring Archived Objects .
RRS (Not recommended)	Frequently accessed, non-critical data	99.99%	99.99%	>= 3	None	None	None

All of the storage classes except for ONEZONE_IA are designed to be resilient to simultaneous complete data loss in a single Availability Zone and partial loss in another Availability Zone.

In addition to the performance requirements of your application scenario, consider price. For storage class pricing, see [Amazon S3 Pricing](#).

Setting the Storage Class of an Object

Amazon S3 APIs support setting (or updating) the storage class of objects as follows:

- When creating a new object, you can specify its storage class. For example, when creating objects using the [PUT Object](#), [POST Object](#), and [Initiate Multipart Upload](#) APIs, you add the `x-amz-storage-class` request header to specify a storage class. If you don't add this header, Amazon S3 uses STANDARD, the default storage class.
- You can also change the storage class of an object that is already stored in Amazon S3 to any other storage class by making a copy of the object using the [PUT Object - Copy](#) API. However, you cannot use [PUT Object - Copy](#) to copy objects that are stored in the GLACIER or DEEP_ARCHIVE storage classes.

You copy the object in the same bucket using the same key name and specify request headers as follows:

- Set the `x-amz-metadata-directive` header to COPY.
- Set the `x-amz-storage-class` to the storage class that you want to use.

In a versioning-enabled bucket, you cannot change the storage class of a specific version of an object. When you copy it, Amazon S3 gives it a new version ID.

- You can direct Amazon S3 to change the storage class of objects by adding a lifecycle configuration to a bucket. For more information, see [Object Lifecycle Management \(p. 119\)](#).
- When setting up a replication configuration, you can set the storage class for replicated objects to any other storage class. However, you cannot replicate objects that are stored in the GLACIER or DEEP_ARCHIVE storage classes. For more information, see [Replication Configuration Overview \(p. 556\)](#).

To create and update object storage classes, you can use the Amazon S3 console, AWS SDKs, or the AWS Command Line Interface (AWS CLI). Each uses the Amazon S3 APIs to send requests to Amazon S3.

Object Subresources

Amazon S3 defines a set of subresources associated with buckets and objects. Subresources are subordinates to objects; that is, subresources do not exist on their own, they are always associated with some other entity, such as an object or a bucket.

The following table lists the subresources associated with Amazon S3 objects.

Subresource	Description
acl	Contains a list of grants identifying the grantees and the permissions granted. When you create an object, the <code>acl</code> identifies the object owner as having full control over the object. You can retrieve an object ACL or replace it with an updated list of grants. Any update to an ACL requires you to replace the existing ACL. For more information about ACLs, see Managing Access with ACLs (p. 403) .
torrent	<p>Amazon S3 supports the BitTorrent protocol. Amazon S3 uses the <code>torrent</code> subresource to return the torrent file associated with the specific object. To retrieve a torrent file, you specify the <code>torrent</code> subresource in your GET request. Amazon S3 creates a torrent file and returns it. You can only retrieve the <code>torrent</code> subresource, you cannot create, update, or delete the <code>torrent</code> subresource. For more information, see Using BitTorrent with Amazon S3 (p. 636).</p> <p>Note Amazon S3 does not support the BitTorrent protocol in AWS Regions launched after May 30, 2016.</p>

Object Versioning

Use versioning to keep multiple versions of an object in one bucket. For example, you could store `my-image.jpg` (version 111111) and `my-image.jpg` (version 222222) in a single bucket. Versioning protects you from the consequences of unintended overwrites and deletions. You can also use versioning to archive objects so you have access to previous versions.

Note

The SOAP API does not support versioning. SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features are not supported for SOAP.

To customize your data retention approach and control storage costs, use object versioning with [Object Lifecycle Management \(p. 119\)](#). For information about creating lifecycle policies using the AWS Management Console, see [How Do I Create a Lifecycle Policy for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

If you have an object expiration lifecycle policy in your non-versioned bucket and you want to maintain the same permanent delete behavior when you enable versioning, you must add a noncurrent expiration policy. The noncurrent expiration lifecycle policy will manage the deletes of the noncurrent object versions in the version-enabled bucket. (A version-enabled bucket maintains one current and zero or more noncurrent object versions.)

You must explicitly enable versioning on your bucket. By default, versioning is disabled. Regardless of whether you have enabled versioning, each object in your bucket has a version ID. If you have not enabled versioning, Amazon S3 sets the value of the version ID to null. If you have enabled versioning, Amazon S3 assigns a unique version ID value for the object. When you enable versioning on a bucket,

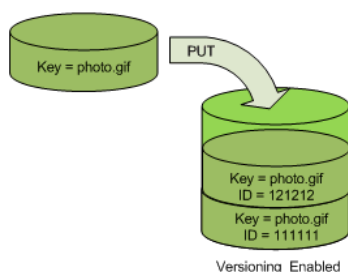
objects already stored in the bucket are unchanged. The version IDs (null), contents, and permissions remain the same.

Enabling and suspending versioning is done at the bucket level. When you enable versioning for a bucket, all objects added to it will have a unique version ID. Unique version IDs are randomly generated, Unicode, UTF-8 encoded, URL-ready, opaque strings that are at most 1024 bytes long. An example version ID is `3/L4kqtJlcpXroDTdmJ+rmSpXd3dIbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo`. Only Amazon S3 generates version IDs. They cannot be edited.

Note

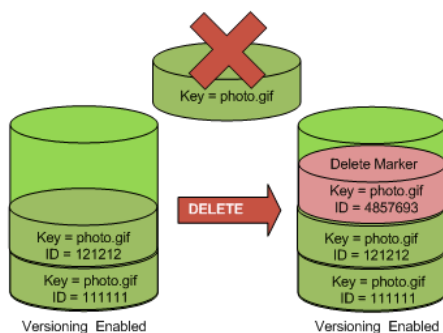
For simplicity, we will use much shorter IDs in all our examples.

When you `PUT` an object in a versioning-enabled bucket, the noncurrent version is not overwritten. The following figure shows that when a new version of `photo.gif` is `PUT` into a bucket that already contains an object with the same name, the original object (ID = 111111) remains in the bucket, Amazon S3 generates a new version ID (121212), and adds the newer version to the bucket.



This functionality prevents you from accidentally overwriting or deleting objects and affords you the opportunity to retrieve a previous version of an object.

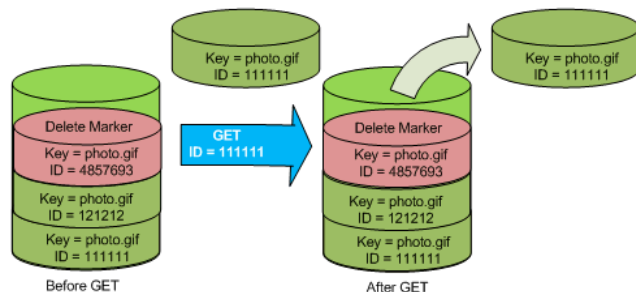
When you `DELETE` an object, all versions remain in the bucket and Amazon S3 inserts a delete marker, as shown in the following figure.



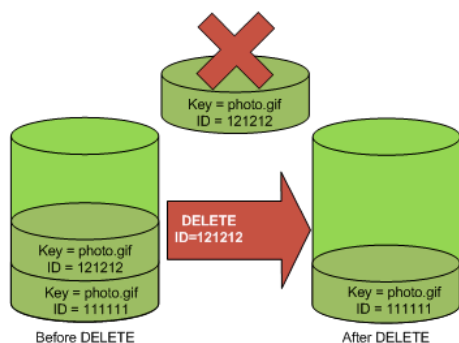
The delete marker becomes the current version of the object. By default, `GET` requests retrieve the most recently stored version. Performing a simple `GET` `Object` request when the current version is a delete marker returns a `403 Forbidden` error, as shown in the following figure.



You can, however, `GET` a noncurrent version of an object by specifying its version ID. In the following figure, we `GET` a specific object version, 111111. Amazon S3 returns that object version even though it's not the current version.



You can permanently delete an object by specifying the version you want to delete. Only the owner of an Amazon S3 bucket can permanently delete a version. The following figure shows how `DELETE versionId` permanently deletes an object from a bucket and that Amazon S3 doesn't insert a delete marker.



You can add additional security by configuring a bucket to enable MFA (multi-factor authentication) Delete. When you do, the bucket owner must include two forms of authentication in any request to delete a version or change the versioning state of the bucket. For more information, see [MFA Delete](#) (p. 433).

Important

If you notice a significant increase in the number of HTTP 503-slow down responses received for Amazon S3 PUT or DELETE object requests to a bucket that has versioning enabled, you might have one or more objects in the bucket for which there are millions of versions. For more information, see [Troubleshooting Amazon S3](#) (p. 643).

For more information, see [Using Versioning](#) (p. 432).

Object Tagging

Use object tagging to categorize storage. Each tag is a key-value pair. Consider the following tagging examples:

- Suppose that an object contains protected health information (PHI) data. You might tag the object using the following key-value pair.

```
PHI=True
```

or


```
Classification=PHI
```

- Suppose that you store project files in your S3 bucket. You might tag these objects with a key named `Project` and a value, as shown following.

```
Project=Blue
```

- You can add multiple tags to an object, as shown following.

```
Project=x  
Classification=confidential
```

You can add tags to new objects when you upload them, or you can add them to existing objects. Note the following:

- You can associate up to 10 tags with an object. Tags that are associated with an object must have unique tag keys.
- A tag key can be up to 128 Unicode characters in length, and tag values can be up to 256 Unicode characters in length.
- The key and values are case sensitive.
- For more information about tag restrictions, see [User-Defined Tag Restrictions](#).

Object key name prefixes also enable you to categorize storage. However, prefix-based categorization is one-dimensional. Consider the following object key names:

```
photos/photo1.jpg  
project/projectx/document.pdf  
project/projecty/document2.pdf
```

These key names have the prefixes `photos/`, `project/projectx/`, and `project/projecty/`. These prefixes enable one-dimensional categorization. That is, everything under a prefix is one category. For example, the prefix `project/projectx` identifies all documents related to project x.

With tagging, you now have another dimension. If you want photo1 in project x category, you can tag the object accordingly. In addition to data classification, tagging offers benefits such as the following:

- Object tags enable fine-grained access control of permissions. For example, you could grant an IAM user permissions to read-only objects with specific tags.
- Object tags enable fine-grained object lifecycle management in which you can specify a tag-based filter, in addition to a key name prefix, in a lifecycle rule.
- When using Amazon S3 analytics, you can configure filters to group objects together for analysis by object tags, by key name prefix, or by both prefix and tags.
- You can also customize Amazon CloudWatch metrics to display information by specific tag filters. The following sections provide details.

Important

It is acceptable to use tags to label objects containing confidential data, such as personally identifiable information (PII) or protected health information (PHI). However, the tags themselves shouldn't contain any confidential information.

To add object tag sets to more than one Amazon S3 object with a single request, you can use Amazon S3 batch operations. You provide Amazon S3 batch operations with a list of objects to operate on. Amazon

S3 batch operations call the respective API to perform the specified operation. A single Amazon S3 batch operations job can perform the specified operation on billions of objects containing exabytes of data.

Amazon S3 batch operations track progress, send notifications, and store a detailed completion report of all actions, providing a fully managed, auditable, serverless experience. You can use Amazon S3 batch operations through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see [the section called "The Basics: Jobs" \(p. 468\)](#).

API Operations Related to Object Tagging

Amazon S3 supports the following API operations that are specifically for object tagging:

Object API Operations

- [PUT Object tagging](#) – Replaces tags on an object. You specify tags in the request body. There are two distinct scenarios of object tag management using this API.
 - Object has no tags – Using this API you can add a set of tags to an object (the object has no prior tags).
 - Object has a set of existing tags – To modify the existing tag set, you must first retrieve the existing tag set, modify it on the client side, and then use this API to replace the tag set.

Note

If you send this request with an empty tag set, Amazon S3 deletes the existing tag set on the object. If you use this method, you will be charged for a Tier 1 Request (PUT). For more information, see [Amazon S3 Pricing](#).

The [DELETE Object tagging](#) request is preferred because it achieves the same result without incurring charges.

- [GET Object tagging](#) – Returns the tag set associated with an object. Amazon S3 returns object tags in the response body.
- [DELETE Object tagging](#) – Deletes the tag set associated with an object.

Other API Operations That Support Tagging

- [PUT Object](#) and [Initiate Multipart Upload](#)– You can specify tags when you create objects. You specify tags using the `x-amz-tagging` request header.
- [GET Object](#) – Instead of returning the tag set, Amazon S3 returns the object tag count in the `x-amz-tag-count` header (only if the requester has permissions to read tags) because the header response size is limited to 8 K bytes. If you want to view the tags, you make another request for the [GET Object tagging](#) API operation.
- [POST Object](#) – You can specify tags in your POST request.

As long as the tags in your request don't exceed the 8 K byte HTTP request header size limit, you can use the `PUT Object` API to create objects with tags. If the tags you specify exceed the header size limit, you can use this POST method in which you include the tags in the body.

PUT Object - Copy – You can specify the `x-amz-tagging-directive` in your request to direct Amazon S3 to either copy (default behavior) the tags or replace tags by a new set of tags provided in the request.

Note the following:

- Tagging follows the eventual consistency model. That is, soon after adding tags to an object, if you try to retrieve the tags, you might get old tags, if any, on the objects. However, a subsequent call will likely provide the updated tags.

Object Tagging and Additional Information

This section explains how object tagging relates to other configurations.

Object Tagging and Lifecycle Management

In bucket lifecycle configuration, you can specify a filter to select a subset of objects to which the rule applies. You can specify a filter based on the key name prefixes, object tags, or both.

Suppose that you store photos (raw and the finished format) in your Amazon S3 bucket. You might tag these objects as shown following.

```
phototype=raw  
or  
phototype=finished
```

You might consider archiving the raw photos to Glacier sometime after they are created. You can configure a lifecycle rule with a filter that identifies the subset of objects with the key name prefix (photos/) that have a specific tag (phototype=raw).

For more information, see [Object Lifecycle Management \(p. 119\)](#).

Object Tagging and Replication

If you configured Replication on your bucket, Amazon S3 replicates tags, provided you grant Amazon S3 permission to read the tags. For more information, see [Overview of Setting Up Replication \(p. 555\)](#).

Object Tagging and Access Control Policies

You can also use permissions policies (bucket and user policies) to manage permissions related to object tagging. For policy actions see the following topics:

- [Permissions for Object Operations \(p. 345\)](#)
- [Permissions Related to Bucket Operations \(p. 346\)](#)

Object tags enable fine-grained access control for managing permissions. You can grant conditional permissions based on object tags. Amazon S3 supports the following condition keys that you can use to grant conditional permissions based on object tags:

- `s3:ExistingObjectTag/<tag-key>` – Use this condition key to verify that an existing object tag has the specific tag key and value.

Note

When granting permissions for the `PUT Object` and `DELETE Object` operations, this condition key is not supported. That is, you cannot create a policy to grant or deny a user permissions to delete or overwrite an object based on its existing tags.

- `s3:RequestObjectTagKeys` – Use this condition key to restrict the tag keys that you want to allow on objects. This is useful when adding tags to objects using the `PutObjectTagging` and `PutObject`, and `POST` object requests.
- `s3:RequestObjectTag/<tag-key>` – Use this condition key to restrict the tag keys and values that you want to allow on objects. This is useful when adding tags to objects using the `PutObjectTagging` and `PutObject`, and `POST` Bucket requests.

For a complete list of Amazon S3 service-specific condition keys, see [Available Condition Keys \(p. 351\)](#). The following permissions policies illustrate how object tagging enables fine grained access permissions management.

Example 1: Allow a user to read only the Objects that have a specific tag

The following permissions policy grants a user permission to read objects, but the condition limits the read permission to only objects that have the following specific tag key and value.

```
security : public
```

Note that the policy uses the Amazon S3 condition key, `s3:ExistingObjectTag/<tag-key>` to specify the key and value.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/security": "public"
        }
      }
    }
  ]
}
```

Example 2: Allow a user to add object tags with restrictions on the allowed tag keys

The following permissions policy grants a user permissions to perform the `s3:PutObjectTagging` action, which allows user to add tags to an existing object. The condition limits the tag keys that the user is allowed to use. The condition uses the `s3:RequestObjectTagKeys` condition key to specify the set of tag keys.

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObjectTagging"
    ],
    "Resource": [
      "arn:aws:s3:::examplebucket/*"
    ],
    "Condition": {
      "ForAllValues:StringLike": {
        "s3:RequestObjectTagKeys": [
          "Owner",
          "CreationDate"
        ]
      }
    }
  }
]
}

```

The policy ensures that the tag set, if specified in the request, has the specified keys. A user might send an empty tag set in `PutObjectTagging`, which is allowed by this policy (an empty tag set in the request removes any existing tags on the object). If you want to prevent a user from removing the tag set, you can add another condition to ensure that the user provides at least one value. The `ForAnyValue` in the condition ensures at least one of the specified values must be present in the request.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObjectTagging"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/*"
      ],
      "Condition": {
        "ForAllValues:StringLike": {
          "s3:RequestObjectTagKeys": [
            "Owner",
            "CreationDate"
          ]
        },
        "ForAnyValue:StringLike": {
          "s3:RequestObjectTagKeys": [
            "Owner",
            "CreationDate"
          ]
        }
      }
    }
  ]
}

```

For more information, see [Creating a Condition That Tests Multiple Key Values \(Set Operations\)](#) in the *IAM User Guide*.

Example 3: Allow a user to add object tags that include a specific tag key and value

The following user policy grants a user permissions to perform the `s3:PutObjectTagging` action, which allows user to add tags on an existing object. The condition requires the user to include a specific tag (`Project`) with value set to `x`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObjectTagging"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:RequestObjectTag/Project": "x"
        }
      }
    }
  ]
}
```

Related Topics

[Managing Object Tags \(p. 116\)](#)

Managing Object Tags

This section explains how you can add object tags programmatically using the AWS SDK for Java or the Amazon S3 console.

Topics

- [Managing Object Tags Using the Console \(p. 116\)](#)
- [Managing Tags Using the AWS SDK for Java \(p. 116\)](#)
- [Managing Tags Using the AWS SDK for .NET \(p. 117\)](#)

Managing Object Tags Using the Console

You can use the Amazon S3 console to add tags to new objects when you upload them or you can add them to existing objects. For instructions on how to add tags to objects using the Amazon S3 console, see [Adding Object Tags](#) in the Amazon Simple Storage Service Console User Guide.

Managing Tags Using the AWS SDK for Java

The following example shows how to use the AWS SDK for Java to set tags for a new object and retrieve or replace tags for an existing object. For more information about object tagging, see [Object Tagging \(p. 110\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class ManagingObjectTags {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String keyName = "**** Object key ****";
        String filePath = "**** File path ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Create an object, add two new tags, and upload the object to Amazon S3.
            PutObjectRequest putRequest = new PutObjectRequest(bucketName, keyName, new
File(filePath));
            List<Tag> tags = new ArrayList<Tag>();
            tags.add(new Tag("Tag 1", "This is tag 1"));
            tags.add(new Tag("Tag 2", "This is tag 2"));
            putRequest.setTagging(new ObjectTagging(tags));
            PutObjectResult putResult = s3Client.putObject(putRequest);

            // Retrieve the object's tags.
            GetObjectTaggingRequest getTaggingRequest = new
GetObjectTaggingRequest(bucketName, keyName);
            GetObjectTaggingResult getTagsResult =
s3Client.getObjectTagging(getTaggingRequest);

            // Replace the object's tags with two new tags.
            List<Tag> newTags = new ArrayList<Tag>();
            newTags.add(new Tag("Tag 3", "This is tag 3"));
            newTags.add(new Tag("Tag 4", "This is tag 4"));
            s3Client.setObjectTagging(new SetObjectTaggingRequest(bucketName, keyName, new
ObjectTagging(newTags)));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Managing Tags Using the AWS SDK for .NET

The following example shows how to use the AWS SDK for .NET to set the tags for a new object and retrieve or replace the tags for an existing object. For more information about object tagging, see [Object Tagging](#) (p. 110).

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    public class ObjectTagsTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** key name for the new object ***";
        private const string filePath = @ "*** file path ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            PutObjectWithTagsTestAsync().Wait();
        }

        static async Task PutObjectWithTagsTestAsync()
        {
            try
            {
                // 1. Put an object with tags.
                var putRequest = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                    FilePath = filePath,
                    TagSet = new List<Tag>{
                        new Tag { Key = "Keyx1", Value = "Value1"},
                        new Tag { Key = "Keyx2", Value = "Value2" }
                    }
                };

                PutObjectResponse response = await client.PutObjectAsync(putRequest);
                // 2. Retrieve the object's tags.
                GetObjectTaggingRequest getTagsRequest = new GetObjectTaggingRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                };

                GetObjectTaggingResponse objectTags = await
                client.GetObjectTaggingAsync(getTagsRequest);
                for (int i = 0; i < objectTags.Tagging.Count; i++)
                    Console.WriteLine("Key: {0}, Value: {1}", objectTags.Tagging[i].Key,
                    objectTags.Tagging[i].Value);

                // 3. Replace the tagset.

                Tagging newTagSet = new Tagging();
                newTagSet.TagSet = new List<Tag>{
                    new Tag { Key = "Key3", Value = "Value3"},
                    new Tag { Key = "Key4", Value = "Value4" }
                };
            }
            catch { }
        }
    }
}
```



```

        PutObjectTaggingRequest putObjTagsRequest = new PutObjectTaggingRequest()
        {
            BucketName = bucketName,
            Key = keyName,
            Tagging = newTagSet
        };
        PutObjectTaggingResponse response2 = await
client.PutObjectTaggingAsync(putObjTagsRequest);

        // 4. Retrieve the object's tags.
        GetObjectTaggingRequest getTagsRequest2 = new GetObjectTaggingRequest();
        getTagsRequest2.BucketName = bucketName;
        getTagsRequest2.Key = keyName;
        GetObjectTaggingResponse objectTags2 = await
client.GetObjectTaggingAsync(getTagsRequest2);
        for (int i = 0; i < objectTags2.Tagging.Count; i++)
            Console.WriteLine("Key: {0}, Value: {1}", objectTags2.Tagging[i].Key,
objectTags2.Tagging[i].Value);

    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine(
            "Error encountered ***. Message:'{0}' when writing an object"
            , e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine(
            "Encountered an error. Message:'{0}' when writing an object"
            , e.Message);
    }
}
}
}
}

```

Object Lifecycle Management

To manage your objects so that they are stored cost effectively throughout their lifecycle, configure their lifecycle. A *lifecycle configuration* is a set of rules that define actions that Amazon S3 applies to a group of objects. There are two types of actions:

- **Transition actions**—Define when objects transition to another [storage class](#). For example, you might choose to transition objects to the STANDARD_IA storage class 30 days after you created them, or archive objects to the GLACIER storage class one year after creating them.

There are costs associated with the lifecycle transition requests. For pricing information, see [Amazon S3 Pricing](#).

- **Expiration actions**—Define when objects expire. Amazon S3 deletes expired objects on your behalf.

The lifecycle expiration costs depend on when you choose to expire objects. For more information, see [Configuring Object Expiration \(p. 126\)](#).

For more information about lifecycle rules, see [Lifecycle Configuration Elements \(p. 127\)](#).

When Should I Use Lifecycle Configuration?

Define lifecycle configuration rules for objects that have a well-defined lifecycle. For example:

- If you upload periodic logs to a bucket, your application might need them for a week or a month. After that, you might want to delete them.
- Some documents are frequently accessed for a limited period of time. After that, they are infrequently accessed. At some point, you might not need real-time access to them, but your organization or regulations might require you to archive them for a specific period. After that, you can delete them.
- You might upload some types of data to Amazon S3 primarily for archival purposes. For example, you might archive digital media, financial and healthcare records, raw genomics sequence data, long-term database backups, and data that must be retained for regulatory compliance.

With lifecycle configuration rules, you can tell Amazon S3 to transition objects to less expensive storage classes, or archive or delete them.

How Do I Configure a Lifecycle?

A lifecycle configuration, an XML file, comprises a set of rules with predefined actions that you want Amazon S3 to perform on objects during their lifetime.

Amazon S3 provides a set of API operations for managing lifecycle configuration on a bucket. Amazon S3 stores the configuration as a *lifecycle subresource* that is attached to your bucket. For details, see the following:

[PUT Bucket lifecycle](#)

[GET Bucket lifecycle](#)

[DELETE Bucket lifecycle](#)

You can also configure the lifecycle by using the Amazon S3 console or programmatically by using the AWS SDK wrapper libraries. If you need to, you can also make the REST API calls directly. For more information, see [Setting Lifecycle Configuration on a Bucket \(p. 143\)](#).

For more information, see the following topics:

- [Additional Considerations for Lifecycle Configuration \(p. 120\)](#)
- [Lifecycle Configuration Elements \(p. 127\)](#)
- [Examples of Lifecycle Configuration \(p. 133\)](#)
- [Setting Lifecycle Configuration on a Bucket \(p. 143\)](#)

Additional Considerations for Lifecycle Configuration

When configuring the lifecycle of objects, you need to understand the following guidelines for transitioning objects, setting expiration dates, and other object configurations.

Topics

- [Transitioning Objects Using Amazon S3 Lifecycle \(p. 121\)](#)
- [Configuring Object Expiration \(p. 126\)](#)
- [Lifecycle and Other Bucket Configurations \(p. 126\)](#)

Transitioning Objects Using Amazon S3 Lifecycle

You can add rules in a lifecycle configuration to tell Amazon S3 to transition objects to another Amazon S3 [storage class](#). For example:

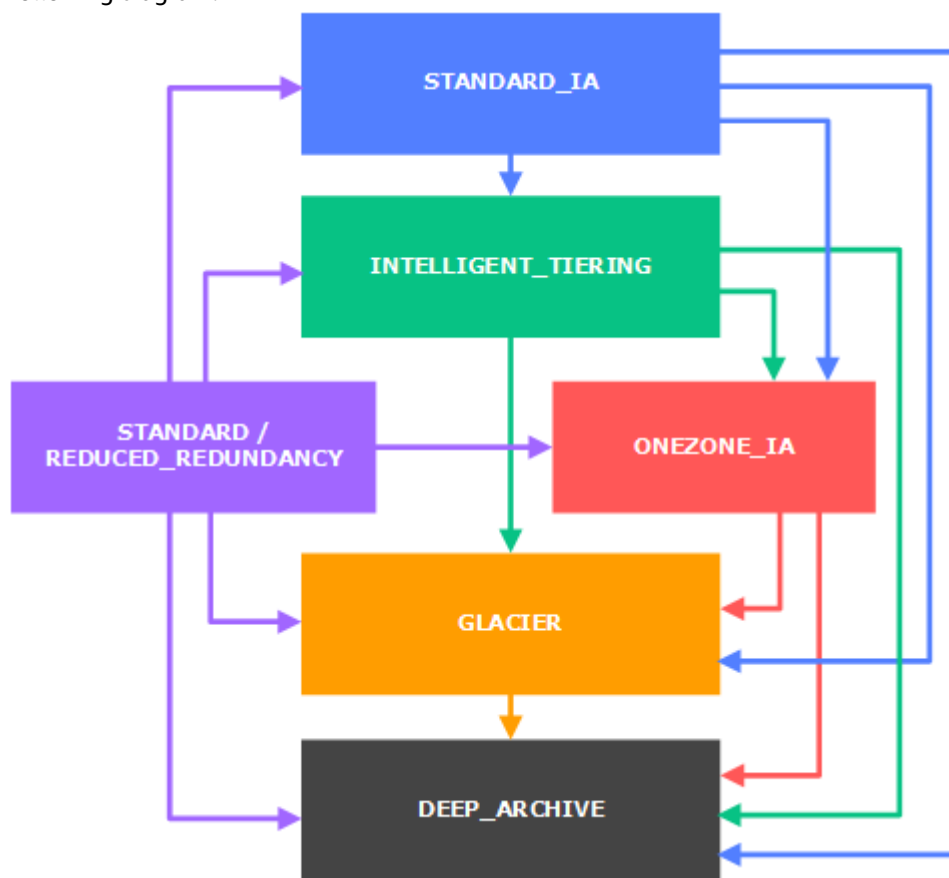
- When you know that objects are infrequently accessed, you might transition them to the STANDARD_IA storage class.
- You might want to archive objects that you don't need to access in real time to the GLACIER storage class.

The following sections describe supported transitions, related constraints, and transitioning to the GLACIER storage class.

Supported Transitions and Related Constraints

In a lifecycle configuration, you can define rules to transition objects from one storage class to another to save on storage costs. When you don't know the access patterns of your objects, or your access patterns are changing over time, you can transition the objects to the INTELLIGENT_TIERING storage class for automatic cost savings. For information about storage classes, see [Amazon S3 Storage Classes](#) (p. 103).

Amazon S3 supports a waterfall model for transitioning between storage classes, as shown in the following diagram.



Amazon S3 supports the following lifecycle transitions between storage classes using a lifecycle configuration:

- You can transition from the STANDARD storage class to any other storage class.
- You can transition from any storage class to the GLACIER or DEEP_ARCHIVE storage classes.
- You can transition from the STANDARD_IA storage class to the INTELLIGENT_TIERING or ONEZONE_IA storage classes.
- You can transition from the INTELLIGENT_TIERING storage class to the ONEZONE_IA storage class.
- You can transition from the GLACIER storage class to the DEEP_ARCHIVE storage class.

The following lifecycle transitions are not supported:

- You can't transition from any storage class to the STANDARD storage class.
- You can't transition from any storage class to the REDUCED_REDUNDANCY storage class.
- You can't transition from the INTELLIGENT_TIERING storage class to the STANDARD_IA storage class.
- You can't transition from the ONEZONE_IA storage class to the STANDARD_IA or INTELLIGENT_TIERING storage classes.
- You can transition from the GLACIER storage class to the DEEP_ARCHIVE storage class only.
- You can't transition from the DEEP_ARCHIVE storage class to any other storage class.

The lifecycle storage class transitions have the following constraints:

- From the STANDARD or STANDARD_IA storage class to INTELLIGENT_TIERING. The following constraints apply:
 - For larger objects, there is a cost benefit for transitioning to INTELLIGENT_TIERING. Amazon S3 does not transition objects that are smaller than 128 KB to the INTELLIGENT_TIERING storage class because it's not cost effective.
- From the STANDARD storage classes to STANDARD_IA or ONEZONE_IA. The following constraints apply:
 - For larger objects, there is a cost benefit for transitioning to STANDARD_IA or ONEZONE_IA. Amazon S3 does not transition objects that are smaller than 128 KB to the STANDARD_IA or ONEZONE_IA storage classes because it's not cost effective.
 - Objects must be stored at least 30 days in the current storage class before you can transition them to STANDARD_IA or ONEZONE_IA. For example, you cannot create a lifecycle rule to transition objects to the STANDARD_IA storage class one day after you create them.

Amazon S3 doesn't transition objects within the first 30 days because newer objects are often accessed more frequently or deleted sooner than is suitable for STANDARD_IA or ONEZONE_IA storage.

- If you are transitioning noncurrent objects (in versioned buckets), you can transition only objects that are at least 30 days noncurrent to STANDARD_IA or ONEZONE_IA storage.
- From the STANDARD_IA storage class to ONEZONE_IA. The following constraints apply:
 - Objects must be stored at least 30 days in the STANDARD_IA storage class before you can transition them to the ONEZONE_IA class.

You can combine these lifecycle actions to manage an object's complete lifecycle. For example, suppose that the objects you create have a well-defined lifecycle. Initially, the objects are frequently accessed for a period of 30 days. Then, objects are infrequently accessed for up to 90 days. After that, the objects are no longer needed, so you might choose to archive or delete them.

In this scenario, you can create a lifecycle rule in which you specify the initial transition action to INTELLIGENT_TIERING, STANDARD_IA, or ONEZONE_IA storage, another transition action to GLACIER storage for archiving, and an expiration action. As you move the objects from one storage class to another, you save on storage cost. For more information about cost considerations, see [Amazon S3 Pricing](#).

Important

You can't specify a single lifecycle rule for both INTELLIGENT_TIERING (or STANDARD_IA or ONEZONE_IA) and GLACIER or DEEP_ARCHIVE transitions when the GLACIER or DEEP_ARCHIVE transition occurs less than 30 days after the INTELLIGENT_TIERING, STANDARD_IA, or ONEZONE_IA transition. That's because there is a minimum 30-day storage charge associated with the INTELLIGENT_TIERING, STANDARD_IA, and ONEZONE_IA storage classes.

The same 30-day minimum applies when you specify a transition from STANDARD_IA storage to ONEZONE_IA or INTELLIGENT_TIERING storage. You can specify two rules to accomplish this, but you pay minimum storage charges. For more information about cost considerations, see [Amazon S3 Pricing](#).

Transitioning to the GLACIER and DEEP ARCHIVE Storage Classes (Object Archival)

Using lifecycle configuration, you can transition objects to the GLACIER or DEEP_ARCHIVE storage classes for archiving. When you choose the GLACIER or DEEP_ARCHIVE storage class, your objects remain in Amazon S3. You cannot access them directly through the separate Amazon S3 Glacier service.

Before you archive objects, review the following sections for relevant considerations.

General Considerations

The following are the general considerations for you to consider before you archive objects:

- Encrypted objects remain encrypted throughout the storage class transition process.
- Objects that are stored in the GLACIER or DEEP_ARCHIVE storage classes are not available in real time.

Archived objects are Amazon S3 objects, but before you can access an archived object, you must first restore a temporary copy of it. The restored object copy is available only for the duration you specify in the restore request. After that, Amazon S3 deletes the temporary copy, and the object remains archived in Amazon S3 Glacier.

You can restore an object by using the Amazon S3 console or programmatically by using the AWS SDKs wrapper libraries or the Amazon S3 REST API in your code. For more information, see [Restoring Archived Objects \(p. 248\)](#).

- Objects that are stored in the GLACIER storage class can only be transitioned to the DEEP_ARCHIVE storage class.

You can use a lifecycle configuration rule to convert the storage class of an object from GLACIER to the DEEP_ARCHIVE storage class only. If you want to change the storage class of an object that is stored in GLACIER to a storage class other than DEEP_ARCHIVE, you must use the restore operation to make a temporary copy of the object first. Then use the copy operation to overwrite the object specifying STANDARD, INTELLIGENT_TIERING, STANDARD_IA, ONEZONE_IA, or REDUCED_REDUNDANCY as the storage class.

- The transition of objects to the DEEP_ARCHIVE storage class can go only one way.

You cannot use a lifecycle configuration rule to convert the storage class of an object from DEEP_ARCHIVE to any other storage class. If you want to change the storage class of an archived object to another storage class, you must use the restore operation to make a temporary copy of the object first. Then use the copy operation to overwrite the object specifying STANDARD, INTELLIGENT_TIERING, STANDARD_IA, ONEZONE_IA, GLACIER, or REDUCED_REDUNDANCY as the storage class.

- The objects that are stored in the GLACIER and DEEP_ARCHIVE storage classes are visible and available only through Amazon S3. They are not available through the separate Amazon S3 Glacier service.

These are Amazon S3 objects, and you can access them only by using the Amazon S3 console or the Amazon S3 API. You cannot access the archived objects through the separate Amazon S3 Glacier console or the Amazon S3 Glacier API.

Cost Considerations

If you are planning to archive infrequently accessed data for a period of months or years, the GLACIER and DEEP_ARCHIVE storage classes can reduce your storage costs. However, to ensure that the GLACIER or DEEP_ARCHIVE storage class is appropriate for you, consider the following:

- **Storage overhead charges** – When you transition objects to the GLACIER or DEEP_ARCHIVE storage class, a fixed amount of storage is added to each object to accommodate metadata for managing the object.
- For each object archived to GLACIER or DEEP_ARCHIVE, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. Amazon S3 stores this metadata so that you can get a real-time list of your archived objects by using the Amazon S3 API. For more information, see [Get Bucket \(List Objects\)](#). You are charged Amazon S3 STANDARD rates for this additional storage.
- For each object that is archived to GLACIER or DEEP_ARCHIVE, Amazon S3 adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged GLACIER or DEEP_ARCHIVE rates for this additional storage.

If you are archiving small objects, consider these storage charges. Also consider aggregating many small objects into a smaller number of large objects to reduce overhead costs.

- **Number of days you plan to keep objects archived**—GLACIER and DEEP_ARCHIVE are long-term archival solutions. The minimal storage duration period is 90 days for the GLACIER storage class and 180 days for DEEP_ARCHIVE. Deleting data that is archived to Amazon S3 Glacier is free if the objects you delete are archived for more than the minimal storage duration period. If you delete or overwrite an archived object within the minimal duration period, Amazon S3 charges a prorated early deletion fee.
- **Amazon S3 GLACIER and DEEP_ARCHIVE transition request charges**— Each object that you transition to the GLACIER or DEEP_ARCHIVE storage class constitutes one transition request. There is a cost for each such request. If you plan to transition a large number of objects, consider the request costs. If you are archiving small objects, consider aggregating many small objects into a smaller number of large objects to reduce transition request costs.
- **Amazon S3 GLACIER and DEEP_ARCHIVE data restore charges**—GLACIER and DEEP_ARCHIVE are designed for long-term archival of data that you access infrequently. For information about data restoration charges, see [How much does it cost to retrieve data from Amazon S3 Glacier?](#) in the Amazon S3 FAQ. For information about how to restore data from Amazon S3 Glacier, see [Restoring Archived Objects](#) (p. 248).

When you archive objects to Amazon S3 Glacier by using object lifecycle management, Amazon S3 transitions these objects asynchronously. There might be a delay between the transition date in the lifecycle configuration rule and the date of the physical transition. You are charged Amazon S3 Glacier prices based on the transition date specified in the rule.

The Amazon S3 product detail page provides pricing information and example calculations for archiving Amazon S3 objects. For more information, see the following topics:

- [How is my storage charge calculated for Amazon S3 objects archived to Amazon S3 Glacier?](#)
- [How am I charged for deleting objects from Amazon S3 Glacier that are less than 3 months old?](#)
- [How much does it cost to retrieve data from Amazon S3 Glacier?](#)
- [Amazon S3 Pricing](#) for storage costs for the different storage classes.

Restoring Archived Objects

Archived objects are not accessible in real time. You must first initiate a restore request and then wait until a temporary copy of the object is available for the duration that you specify in the request. After you receive a temporary copy of the restored object, the object's storage class remains GLACIER or DEEP_ARCHIVE. (A [HEAD Object](#) or [GET Object](#) API operation request will return GLACIER or DEEP_ARCHIVE as the storage class.)

Note

When you restore an archive, you are paying for both the archive (GLACIER or DEEP_ARCHIVE rate) and a copy that you restored temporarily (REDUCED_REDUNDANCY storage rate). For information about pricing, see [Amazon S3 Pricing](#).

You can restore an object copy programmatically or by using the Amazon S3 console. Amazon S3 processes only one restore request at a time per object. For more information, see [Restoring Archived Objects](#) (p. 248).

Configuring Object Expiration

When an object reaches the end of its lifetime, Amazon S3 queues it for removal and removes it asynchronously. There may be a delay between the expiration date and the date at which Amazon S3 removes an object. You are not charged for storage time associated with an object that has expired.

To find when an object is scheduled to expire, use the [HEAD Object](#) or the [GET Object](#) API operations. These API operations return response headers that provide this information.

If you create a lifecycle expiration rule that causes objects that have been in INTELLIGENT_TIERING, STANDARD_IA, or ONEZONE_IA storage for less than 30 days to expire, you are charged for 30 days. If you create a lifecycle expiration rule that causes objects that have been in GLACIER storage for less than 90 days to expire, you are charged for 90 days. If you create a lifecycle expiration rule that causes objects that have been in DEEP_ARCHIVE storage for less than 180 days to expire, you are charged for 180 days. For more information, see [Amazon S3 Pricing](#).

Lifecycle and Other Bucket Configurations

In addition to lifecycle configurations, you can associate other configurations with your bucket. This section explains how lifecycle configuration relates to other bucket configurations.

Lifecycle and Versioning

You can add lifecycle configurations to unversioned buckets and versioning-enabled buckets. For more information, see [Object Versioning](#) (p. 108).

A versioning-enabled bucket maintains one current object version, and zero or more noncurrent object versions. You can define separate lifecycle rules for current and noncurrent object versions.

For more information, see [Lifecycle Configuration Elements](#) (p. 127). For information about versioning, see [Object Versioning](#) (p. 108).

Lifecycle Configuration on MFA-enabled Buckets

Lifecycle configuration on MFA-enabled buckets is not supported.

Lifecycle and Logging

Amazon S3 lifecycle actions are not captured by CloudTrail object level logging since CloudTrail captures API requests made to external Amazon S3 endpoints whereas Amazon S3 lifecycle actions are performed using internal Amazon S3 endpoints. Amazon S3 server access logs can be enabled in an S3 bucket to capture Amazon S3 lifecycle related actions such as object transition to another storage class and object expiration resulting in permanent deletion or logical deletion. For more information, see [Server Access Logging](#) (p. 647)

If you have logging enabled on your bucket, Amazon S3 server access logs report the results of the following operations:

Operation log	Description
S3.EXPIRE.OBJECT	Amazon S3 permanently deletes the object due to the lifecycle expiration action.
S3.CREATE.DELETEMARKER	Amazon S3 logically deletes the current version and adds a delete marker in a versioning enabled bucket.

Operation log	Decription
<code>S3.TRANSITION_SIA.OBJECT</code>	Amazon S3 transitions the object to the STANDARD_IA storage class.
<code>S3.TRANSITION_ZIA.OBJECT</code>	Amazon S3 transitions the object to the ONEZONE_IA storage class.
<code>S3.TRANSITION_INT.OBJECT</code>	Amazon S3 transitions the object to the Intelligent-Tiering storage class.
<code>S3.TRANSITION.OBJECT</code>	Amazon S3 initiates the transition of object to the GLACIER storage class.
<code>S3.TRANSITION_GDA.OBJECT</code>	Amazon S3 initiates the transition of object to the GLACIER DEEP_ARCHIVE storage class.
<code>S3.DELETE.UPLOAD</code>	Amazon S3 aborts incomplete multipart upload.

Note

Amazon S3 server access log records are generally delivered on a best effort basis and cannot be used for complete accounting of all Amazon S3 requests.

More Info

- [Lifecycle Configuration Elements \(p. 127\)](#)
- [Transitioning to the GLACIER and DEEP ARCHIVE Storage Classes \(Object Archival\) \(p. 123\)](#)
- [Setting Lifecycle Configuration on a Bucket \(p. 143\)](#)

Lifecycle Configuration Elements

Topics

- [ID Element \(p. 128\)](#)
- [Status Element \(p. 128\)](#)
- [Filter Element \(p. 128\)](#)
- [Elements to Describe Lifecycle Actions \(p. 130\)](#)

You specify a lifecycle configuration as XML, consisting of one or more lifecycle rules.

```
<LifecycleConfiguration>
  <Rule>
    ...
  </Rule>
  <Rule>
    ...
  </Rule>
</LifecycleConfiguration>
```

Each rule consists of the following:

- Rule metadata that include a rule ID, and status indicating whether the rule is enabled or disabled. If a rule is disabled, Amazon S3 doesn't perform any actions specified in the rule.
- Filter identifying objects to which the rule applies. You can specify a filter by using an object key prefix, one or more object tags, or both.

- One or more transition or expiration actions with a date or a time period in the object's lifetime when you want Amazon S3 to perform the specified action.

The following sections describe the XML elements in a lifecycle configuration. For example lifecycle configurations, see [Examples of Lifecycle Configuration](#) (p. 133).

ID Element

A lifecycle configuration can have up to 1,000 rules. The <ID> element uniquely identifies a rule. ID length is limited to 255 characters.

Status Element

The <Status> element value can be either Enabled or Disabled. If a rule is disabled, Amazon S3 doesn't perform any of the actions defined in the rule.

Filter Element

A lifecycle rule can apply to all or a subset of objects in a bucket based on the <Filter> element that you specify in the lifecycle rule.

You can filter objects by key prefix, object tags, or a combination of both (in which case Amazon S3 uses a logical AND to combine the filters). Consider the following examples:

- **Specifying a filter using key prefixes** – This example shows a lifecycle rule that applies to a subset of objects based on the key name prefix (logs/). For example, the lifecycle rule applies to objects logs/mylog.txt, logs/temp1.txt, and logs/test.txt. The rule does not apply to the object example.jpg.

```
<LifecycleConfiguration>
  <Rule>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    transition/expiration actions.
    ...
  </Rule>
  ...
</LifecycleConfiguration>
```

If you want to apply a lifecycle action to a subset of objects based on different key name prefixes, specify separate rules. In each rule, specify a prefix-based filter. For example, to describe a lifecycle action for objects with key prefixes projectA/ and projectB/, you specify two rules as shown following:

```
<LifecycleConfiguration>
  <Rule>
    <Filter>
      <Prefix>projectA/</Prefix>
    </Filter>
    transition/expiration actions.
    ...
  </Rule>
  <Rule>
    <Filter>
      <Prefix>projectB/</Prefix>
    </Filter>
    transition/expiration actions.
```

```
...
</Rule>
</LifecycleConfiguration>
```

For more information about object keys, see [Object Keys \(p. 99\)](#).

- **Specifying a filter based on object tags** – In the following example, the lifecycle rule specifies a filter based on a tag (**key**) and value (**value**). The rule then applies only to a subset of objects with the specific tag.

```
<LifecycleConfiguration>
  <Rule>
    <Filter>
      <Tag>
        <Key>key</Key>
        <Value>value</Value>
      </Tag>
    </Filter>
    transition/expiration actions.
    ...
  </Rule>
</LifecycleConfiguration>
```

You can specify a filter based on multiple tags. You must wrap the tags in the **<AND>** element shown in the following example. The rule directs Amazon S3 to perform lifecycle actions on objects with two tags (with the specific tag key and value).

```
<LifecycleConfiguration>
  <Rule>
    <Filter>
      <And>
        <Tag>
          <Key>key1</Key>
          <Value>value1</Value>
        </Tag>
        <Tag>
          <Key>key2</Key>
          <Value>value2</Value>
        </Tag>
        ...
      </And>
    </Filter>
    transition/expiration actions.
  </Rule>
</LifecycleConfiguration>
```

The lifecycle rule applies to objects that have both of the tags specified. Amazon S3 performs a logical AND. Note the following:

- Each tag must match both key and value exactly.
- The rule applies to a subset of objects that has tags specified in the rule. If an object has additional tags specified, it doesn't matter.

Note

When you specify multiple tags in a filter, each tag key must be unique.

- **Specifying a filter based on both prefix and one or more tags** – In a lifecycle rule, you can specify a filter based on both the key prefix and one or more tags. Again, you must wrap all of these in the **<And>** element as shown following:

```
<LifecycleConfiguration>
  <Rule>
```

```
<Filter>
  <And>
    <Prefix>key-prefix</Prefix>
    <Tag>
      <Key>key1</Key>
      <Value>value1</Value>
    </Tag>
    <Tag>
      <Key>key2</Key>
      <Value>value2</Value>
    </Tag>
    ...
  </And>
</Filter>
<Status>Enabled</Status>
transition/expiration actions.
</Rule>
</LifecycleConfiguration>
```

Amazon S3 combines these filters using a logical AND. That is, the rule applies to subset of objects with a specific key prefix and specific tags. A filter can have only one prefix, and zero or more tags.

- You can specify an empty filter, in which case the rule applies to all objects in the bucket.

```
<LifecycleConfiguration>
  <Rule>
    <Filter>
    </Filter>
    <Status>Enabled</Status>
    transition/expiration actions.
  </Rule>
</LifecycleConfiguration>
```

Elements to Describe Lifecycle Actions

You can direct Amazon S3 to perform specific actions in an object's lifetime by specifying one or more of the following predefined actions in a lifecycle rule. The effect of these actions depends on the versioning state of your bucket.

- **Transition** action element – You specify the `Transition` action to transition objects from one storage class to another. For more information about transitioning objects, see [Supported Transitions and Related Constraints \(p. 121\)](#). When a specified date or time period in the object's lifetime is reached, Amazon S3 performs the transition.

For a versioned bucket (versioning-enabled or versioning-suspended bucket), the `Transition` action applies to the current object version. To manage noncurrent versions, Amazon S3 defines the `NoncurrentVersionTransition` action (described below).

- **Expiration action element** – The `Expiration` action expires objects identified in the rule and applies to eligible objects in any of the Amazon S3 storage classes. For more information about storage classes, see [Amazon S3 Storage Classes \(p. 103\)](#). Amazon S3 makes all expired objects unavailable. Whether the objects are permanently removed depends on the versioning state of the bucket.

Important

Object expiration lifecycle policies do not remove incomplete multipart uploads. To remove incomplete multipart uploads you must use the **AbortIncompleteMultipartUpload** lifecycle configuration action that is described later in this section.

- **Non-versioned bucket** – The `Expiration` action results in Amazon S3 permanently removing the object.

- **Versioned bucket** – For a versioned bucket (that is, versioning-enabled or versioning-suspended), there are several considerations that guide how Amazon S3 handles the `expiration` action. For more information, see [Using Versioning \(p. 432\)](#). Regardless of the versioning state, the following applies:
 - The `Expiration` action applies only to the current version (it has no impact on noncurrent object versions).
 - Amazon S3 doesn't take any action if there are one or more object versions and the delete marker is the current version.
 - If the current object version is the only object version and it is also a delete marker (also referred as an *expired object delete marker*, where all object versions are deleted and you only have a delete marker remaining), Amazon S3 removes the expired object delete marker. You can also use the expiration action to direct Amazon S3 to remove any expired object delete markers. For an example, see [Example 7: Removing Expired Object Delete Markers \(p. 141\)](#).

Also consider the following when setting up Amazon S3 to manage expiration:

- **Versioning-enabled bucket**

If the current object version is not a delete marker, Amazon S3 adds a delete marker with a unique version ID. This makes the current version noncurrent, and the delete marker the current version.

- **Versioning-suspended bucket**

In a versioning-suspended bucket, the expiration action causes Amazon S3 to create a delete marker with null as the version ID. This delete marker replaces any object version with a null version ID in the version hierarchy, which effectively deletes the object.

In addition, Amazon S3 provides the following actions that you can use to manage noncurrent object versions in a versioned bucket (that is, versioning-enabled and versioning-suspended buckets).

- **NoncurrentVersionTransition** action element – Use this action to specify how long (from the time the objects became noncurrent) you want the objects to remain in the current storage class before Amazon S3 transitions them to the specified storage class. For more information about transitioning objects, see [Supported Transitions and Related Constraints \(p. 121\)](#).
- **NoncurrentVersionExpiration** action element – Use this action to specify how long (from the time the objects became noncurrent) you want to retain noncurrent object versions before Amazon S3 permanently removes them. The deleted object can't be recovered.

This delayed removal of noncurrent objects can be helpful when you need to correct any accidental deletes or overwrites. For example, you can configure an expiration rule to delete noncurrent versions five days after they become noncurrent. For example, suppose that on 1/1/2014 10:30 AM UTC, you create an object called `photo.gif` (version ID 111111). On 1/2/2014 11:30 AM UTC, you accidentally delete `photo.gif` (version ID 111111), which creates a delete marker with a new version ID (such as version ID 4857693). You now have five days to recover the original version of `photo.gif` (version ID 111111) before the deletion is permanent. On 1/8/2014 00:00 UTC, the lifecycle rule for expiration executes and permanently deletes `photo.gif` (version ID 111111), five days after it became a noncurrent version.

Important

Object expiration lifecycle policies do not remove incomplete multipart uploads. To remove incomplete multipart uploads, you must use the **AbortIncompleteMultipartUpload** lifecycle configuration action that is described later in this section.

In addition to the transition and expiration actions, you can use the following lifecycle configuration action to direct Amazon S3 to abort incomplete multipart uploads.

- **AbortIncompleteMultipartUpload** action element – Use this element to set a maximum time (in days) that you want to allow multipart uploads to remain in progress. If the applicable multipart uploads (determined by the key name `prefix` specified in the lifecycle rule) are not successfully completed within the predefined time period, Amazon S3 aborts the incomplete multipart uploads. For more information, see [Aborting Incomplete Multipart Uploads Using a Bucket Lifecycle Policy \(p. 177\)](#).

Note

You cannot specify this lifecycle action in a rule that specifies a filter based on object tags.

- **ExpiredObjectDeleteMarker** action element – In a versioning-enabled bucket, a delete marker with zero noncurrent versions is referred to as the expired object delete marker. You can use this lifecycle action to direct S3 to remove the expired object delete markers. For an example, see [Example 7: Removing Expired Object Delete Markers \(p. 141\)](#).

Note

You cannot specify this lifecycle action in a rule that specifies a filter based on object tags.

How Amazon S3 Calculates How Long an Object Has Been Noncurrent

In a versioning-enabled bucket, you can have multiple versions of an object, there is always one current version, and zero or more noncurrent versions. Each time you upload an object, the current version is retained as the noncurrent version and the newly added version, the successor, becomes the current version. To determine the number of days an object is noncurrent, Amazon S3 looks at when its successor was created. Amazon S3 uses the number of days since its successor was created as the number of days an object is noncurrent.

Restoring Previous Versions of an Object When Using Lifecycle Configurations

As explained in detail in the topic [Restoring Previous Versions \(p. 449\)](#), you can use either of the following two methods to retrieve previous versions of an object:

1. By copying a noncurrent version of the object into the same bucket. The copied object becomes the current version of that object, and all object versions are preserved.
2. By permanently deleting the current version of the object. When you delete the current object version, you, in effect, turn the noncurrent version into the current version of that object.

When using lifecycle configuration rules with versioning-enabled buckets, we recommend as a best practice that you use the first method.

Because of Amazon S3's eventual consistency semantics, a current version that you permanently deleted may not disappear until the changes propagate (Amazon S3 may be unaware of this deletion). In the meantime, the lifecycle rule that you configured to expire noncurrent objects may permanently remove noncurrent objects, including the one you want to restore. So, copying the old version, as recommended in the first method, is the safer alternative.

Lifecycle Rules: Based on an Object's Age

You can specify a time period, in number of days from the creation (or modification) of the objects, when Amazon S3 can take the action.

When you specify the number of days in the `Transition` and `Expiration` actions in a lifecycle configuration, note the following:

- It is the number of days since object creation when the action will occur.
- Amazon S3 calculates the time by adding the number of days specified in the rule to the object creation time and rounding the resulting time to the next day midnight UTC. For example, if an object was created at 1/15/2014 10:30 AM UTC and you specify 3 days in a transition rule, then the transition date of the object would be calculated as 1/19/2014 00:00 UTC.

Note

Amazon S3 maintains only the last modified date for each object. For example, the Amazon S3 console shows the **Last Modified** date in the object **Properties** pane. When you initially create a new object, this date reflects the date the object is created. If you replace the object, the date changes accordingly. So when we use the term *creation date*, it is synonymous with the term *last modified date*.

When specifying the number of days in the `NoncurrentVersionTransition` and `NoncurrentVersionExpiration` actions in a lifecycle configuration, note the following:

- It is the number of days from when the version of the object becomes noncurrent (that is, when the object is overwritten or deleted), that Amazon S3 will perform the action on the specified object or objects.
- Amazon S3 calculates the time by adding the number of days specified in the rule to the time when the new successor version of the object is created and rounding the resulting time to the next day midnight UTC. For example, in your bucket, suppose that you have a current version of an object that was created at 1/1/2014 10:30 AM UTC. If the new version of the object that replaces the current version is created at 1/15/2014 10:30 AM UTC, and you specify 3 days in a transition rule, the transition date of the object is calculated as 1/19/2014 00:00 UTC.

Lifecycle Rules: Based on a Specific Date

When specifying an action in a lifecycle rule, you can specify a date when you want Amazon S3 to take the action. When the specific date arrives, S3 applies the action to all qualified objects (based on the filter criteria).

If you specify a lifecycle action with a date that is in the past, all qualified objects become immediately eligible for that lifecycle action.

Important

The date-based action is not a one-time action. S3 continues to apply the date-based action even after the date has passed, as long as the rule status is Enabled.

For example, suppose that you specify a date-based Expiration action to delete all objects (assume no filter specified in the rule). On the specified date, S3 expires all the objects in the bucket. S3 also continues to expire any new objects you create in the bucket. To stop the lifecycle action, you must remove the action from the lifecycle configuration, disable the rule, or delete the rule from the lifecycle configuration.

The date value must conform to the ISO 8601 format. The time is always midnight UTC.

Note

You can't create the date-based lifecycle rules using the Amazon S3 console, but you can view, disable, or delete such rules.

Examples of Lifecycle Configuration

This section provides examples of lifecycle configuration. Each example shows how you can specify the XML in each of the example scenarios.

Topics

- [Example 1: Specifying a Filter \(p. 134\)](#)
- [Example 2: Disabling a Lifecycle Rule \(p. 135\)](#)
- [Example 3: Tiering Down Storage Class over an Object's Lifetime \(p. 136\)](#)
- [Example 4: Specifying Multiple Rules \(p. 137\)](#)
- [Example 5: Overlapping Filters, Conflicting Lifecycle Actions, and What Amazon S3 Does \(p. 137\)](#)

- [Example 6: Specifying a Lifecycle Rule for a Versioning-Enabled Bucket](#) (p. 140)
- [Example 7: Removing Expired Object Delete Markers](#) (p. 141)
- [Example 8: Lifecycle Configuration to Abort Multipart Uploads](#) (p. 142)

Example 1: Specifying a Filter

Each lifecycle rule includes a filter that you can use to identify a subset of objects in your bucket to which the lifecycle rule applies. The following lifecycle configurations show examples of how you can specify a filter.

- In this lifecycle configuration rule, the filter specifies a key prefix (`tax/`). Therefore, the rule applies to objects with key name prefix `tax/`, such as `tax/doc1.txt` and `tax/doc2.txt`

The rule specifies two actions that direct Amazon S3 to do the following:

- Transition objects to the GLACIER storage class 365 days (one year) after creation.
- Delete objects (the `Expiration` action) 3650 days (10 years) after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition and Expiration Rule</ID>
    <Filter>
      <Prefix>tax/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Instead of specifying object age in terms of days after creation, you can specify a date for each action. However, you can't use both `Date` and `Days` in the same rule.

- If you want the lifecycle rule to apply to all objects in the bucket, specify an empty prefix. In the following configuration, the rule specifies a `Transition` action directing Amazon S3 to transition objects to the GLACIER storage class 0 days after creation in which case objects are eligible for archival to Amazon S3 Glacier at midnight UTC following creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Archive all object same-day upon creation</ID>
    <Filter>
      <Prefix></Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

- You can specify zero or one key name prefix and zero or more object tags in a filter. The following example code applies the lifecycle key rule to a subset of objects with the `tax/` key prefix and to objects that have two tags with specific key and value. Note that when you specify more than one filter, you

must include the AND as shown (Amazon S3 applies a logical AND to combine the specified filter conditions).

```
...
<Filter>
  <And>
    <Prefix>tax/</Prefix>
    <Tag>
      <Key>key1</Key>
      <Value>value1</Value>
    </Tag>
    <Tag>
      <Key>key2</Key>
      <Value>value2</Value>
    </Tag>
  </And>
</Filter>
...
```

- You can filter objects based only on tags. For example, the following lifecycle rule applies to objects that have the two specified tags (it does not specify any prefix):

```
...
<Filter>
  <And>
    <Tag>
      <Key>key1</Key>
      <Value>value1</Value>
    </Tag>
    <Tag>
      <Key>key2</Key>
      <Value>value2</Value>
    </Tag>
  </And>
</Filter>
...
```

Important

When you have multiple rules in a lifecycle configuration, an object can become eligible for multiple lifecycle actions. The general rules that Amazon S3 follows in such cases are:

- Permanent deletion takes precedence over transition.
- Transition takes precedence over creation of delete markers.
- When an object is eligible for both a GLACIER and STANDARD_IA (or ONEZONE_IA) transition, Amazon S3 chooses the GLACIER transition.

For examples, see [Example 5: Overlapping Filters, Conflicting Lifecycle Actions, and What Amazon S3 Does](#) (p. 137)

Example 2: Disabling a Lifecycle Rule

You can temporarily disable a lifecycle rule. The following lifecycle configuration specifies two rules:

- Rule 1 directs Amazon S3 to transition objects with the `logs/` prefix to the GLACIER storage class soon after creation.
- Rule 2 directs Amazon S3 to transition objects with the `documents/` prefix to the GLACIER storage class soon after creation.

In the policy Rule 1 is enabled and Rule 2 is disabled. Amazon S3 will not take any action on disabled rules.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule1</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
  </Rule>
  <Rule>
    <ID>Rule2</ID>
    <Prefix>documents/</Prefix>
    <Status>Disabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

Example 3: Tiering Down Storage Class over an Object's Lifetime

In this example, you leverage lifecycle configuration to tier down the storage class of objects over their lifetime. Tiering down can help reduce storage costs. For more information about pricing, see [Amazon S3 Pricing](#).

The following lifecycle configuration specifies a rule that applies to objects with key name prefix logs/. The rule specifies the following actions:

- Two transition actions:
 - Transition objects to the STANDARD_IA storage class 30 days after creation.
 - Transition objects to the GLACIER storage class 90 days after creation.
- One expiration action that directs Amazon S3 to delete objects a year after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>example-id</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>30</Days>
      <StorageClass>STANDARD_IA</StorageClass>
    </Transition>
    <Transition>
      <Days>90</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Note

You can use one rule to describe all lifecycle actions if all actions apply to the same set of objects (identified by the filter). Otherwise, you can add multiple rules with each specifying a different filter.

Example 4: Specifying Multiple Rules

You can specify multiple rules if you want different lifecycle actions of different objects. The following lifecycle configuration has two rules:

- Rule 1 applies to objects with the key name prefix `classA/`. It directs Amazon S3 to transition objects to the GLACIER storage class one year after creation and expire these objects 10 years after creation.
- Rule 2 applies to objects with key name prefix `classB/`. It directs Amazon S3 to transition objects to the STANDARD_IA storage class 90 days after creation and delete them one year after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>ClassADocRule</ID>
    <Filter>
      <Prefix>classA</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>ClassBDocRule</ID>
    <Filter>
      <Prefix>classB</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>90</Days>
      <StorageClass>STANDARD_IA</StorageClass>
    </Transition>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Example 5: Overlapping Filters, Conflicting Lifecycle Actions, and What Amazon S3 Does

You might specify a lifecycle configuration in which you specify overlapping prefixes, or actions.

Generally, lifecycle will optimize for cost. For example, if two expiration policies overlap, the shorter expiration policy will be honored so that data is not stored for longer than expected.

Likewise, if two transition policies overlap, lifecycle will transition your objects to the lower cost storage class. In both cases, lifecycle attempts to choose the path that is least expensive for you. An exception to this general rule is with the INTELLIGENT_TIERING storage class. INTELLIGENT_TIERING will be favored by lifecycle over any storage class, aside from GLACIER and DEEP_ARCHIVE storage classes.

The following examples show how Amazon S3 chooses to resolve potential conflicts.

Example 1: Overlapping Prefixes (No Conflict)

The following example configuration has two rules that specify overlapping prefixes as follows:

- First rule specifies an empty filter, indicating all objects in the bucket.
- Second rule specifies a key name prefix `logs/`, indicating only a subset of objects.

Rule 1 requests Amazon S3 to delete all objects one year after creation, and Rule 2 requests Amazon S3 to transition a subset of objects to the `STANDARD_IA` storage class 30 days after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
    </Filter>
    <Status>Enabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>Rule 2</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <StorageClass>STANDARD_IA</StorageClass>
      <Days>30</Days>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

Example 2: Conflicting Lifecycle Actions

In this example configuration, there are two rules that direct Amazon S3 to perform two different actions on the same set of objects at the same time in object's lifetime:

- Both rules specify the same key name prefix, so both rules apply to the same set of objects.
- Both rules specify the same 365 days after object creation when the rules apply.
- One rule directs Amazon S3 to transition objects to the `STANDARD_IA` storage class and another rule wants Amazon S3 to expire the objects at the same time.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>Rule 2</ID>
    <Filter>
```

```
<Prefix>logs/</Prefix>
</Filter>
<Status>Enabled</Status>
<Transition>
  <StorageClass>STANDARD_IA</StorageClass>
  <Days>365</Days>
</Transition>
</Rule>
</LifecycleConfiguration>
```

In this case, because you want objects to expire (removed), there is no point in changing the storage class, and Amazon S3 simply chooses the expiration action on these objects.

Example 3: Overlapping Prefixes Resulting in Conflicting Lifecycle Actions

In this example, the configuration has two rules, which specify overlapping prefixes as follows:

- Rule 1 specifies an empty prefix (indicating all objects).
- Rule 2 specifies a key name prefix (logs/) that identifies a subset of all objects.

For the subset of objects with the logs/ key name prefix, lifecycle actions in both rules apply. One rule directing Amazon S3 to transition objects 10 days after creation and another rule directing Amazon S3 to transition objects 365 days after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
      <Prefix></Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <StorageClass>STANDARD_IA</StorageClass>
      <Days>10</Days>
    </Transition>
  </Rule>
  <Rule>
    <ID>Rule 2</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <StorageClass>STANDARD_IA</StorageClass>
      <Days>365</Days>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

In this case, Amazon S3 chooses to transition them 10 days after creation.

Example 4: Tag-based Filtering and Resulting Conflicting Lifecycle Actions

Suppose that you have the following lifecycle policy that has two rules, each specifying a tag filter:

- Rule 1 specifies a tag-based filter (tag1/value1). This rule directs Amazon S3 to transition objects to the GLACIER storage class 365 days after creation.
- Rule 2 specifies a tag-based filter (tag2/value2). This rule directs Amazon S3 to expire objects 14 days after creation.

The lifecycle configuration is shown following:

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
      <Tag>
        <Key>tag1</Key>
        <Value>value1</Value>
      </Tag>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <StorageClass>GLACIER</StorageClass>
      <Days>365</Days>
    </Transition>
  </Rule>
  <Rule>
    <ID>Rule 2</ID>
    <Filter>
      <Tag>
        <Key>tag2</Key>
        <Value>value1</Value>
      </Tag>
    </Filter>
    <Status>Enabled</Status>
    <Expiration>
      <Days>14</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

The policy is fine, but if there is an object with both tags, then S3 has to decide what to do. That is, both rules apply to an object and in effect you are directing Amazon S3 to perform conflicting actions. In this case, Amazon S3 expires the object 14 days after creation. The object is removed, and therefore the transition action does not come into play.

Example 6: Specifying a Lifecycle Rule for a Versioning-Enabled Bucket

Suppose that you have a versioning-enabled bucket, which means that for each object you have a current version and zero or more noncurrent versions. You want to maintain one year's worth of history and then delete the noncurrent versions. For more information about versioning, see [Object Versioning \(p. 108\)](#).

Also, you want to save storage costs by moving noncurrent versions to GLACIER 30 days after they become noncurrent (assuming cold data for which you don't need real-time access). In addition, you also expect frequency of access of the current versions to diminish 90 days after creation so you might choose to move these objects to the STANDARD_IA storage class.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Filter>
      <Prefix></Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>90</Days>
      <StorageClass>STANDARD_IA</StorageClass>
    </Transition>
    <NoncurrentVersionTransition>
```

```
<NoncurrentDays>30</NoncurrentDays>
<StorageClass>GLACIER</StorageClass>
</NoncurrentVersionTransition>
<NoncurrentVersionExpiration>
  <NoncurrentDays>365</NoncurrentDays>
</NoncurrentVersionExpiration>
</Rule>
</LifecycleConfiguration>
```

Example 7: Removing Expired Object Delete Markers

A versioning-enabled bucket has one current version and zero or more noncurrent versions for each object. When you delete an object, note the following:

- If you don't specify a version ID in your delete request, Amazon S3 adds a delete marker instead of deleting the object. The current object version becomes noncurrent, and then the delete marker becomes the current version.
- If you specify a version ID in your delete request, Amazon S3 deletes the object version permanently (a delete marker is not created).
- A delete marker with zero noncurrent versions is referred to as the *expired object delete marker*.

This example shows a scenario that can create expired object delete markers in your bucket, and how you can use lifecycle configuration to direct Amazon S3 to remove the expired object delete markers.

Suppose that you write a lifecycle policy that specifies the `NoncurrentVersionExpiration` action to remove the noncurrent versions 30 days after they become noncurrent as shown following:

```
<LifecycleConfiguration>
  <Rule>
    ...
    <NoncurrentVersionExpiration>
      <NoncurrentDays>30</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

The `NoncurrentVersionExpiration` action does not apply to the current object versions. It only removes noncurrent versions.

For current object versions, you have the following options to manage their lifetime depending on whether or not the current object versions follow a well-defined lifecycle:

- **Current object versions follow a well-defined lifecycle.**

In this case you can use lifecycle policy with the `Expiration` action to direct Amazon S3 to remove current versions as shown in the following example:

```
<LifecycleConfiguration>
  <Rule>
    ...
    <Expiration>
      <Days>60</Days>
    </Expiration>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>30</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

Amazon S3 removes current versions 60 days after they are created by adding a delete marker for each of the current object versions. This makes the current version noncurrent and the delete marker becomes the current version. For more information, see [Using Versioning \(p. 432\)](#).

The `NoncurrentVersionExpiration` action in the same lifecycle configuration removes noncurrent objects 30 days after they become noncurrent. Thus, all object versions are removed and you have expired object delete markers, but Amazon S3 detects and removes the expired object delete markers for you.

- **Current object versions don't have a well-defined lifecycle.**

In this case you might remove the objects manually when you don't need them, creating a delete marker with one or more noncurrent versions. If lifecycle configuration with `NoncurrentVersionExpiration` action removes all the noncurrent versions, you now have expired object delete markers.

Specifically for this scenario, Amazon S3 lifecycle configuration provides an `Expiration` action where you can request Amazon S3 to remove the expired object delete markers:

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Expiration>
      <ExpiredObjectDeleteMarker>true</ExpiredObjectDeleteMarker>
    </Expiration>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>30</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

By setting the `ExpiredObjectDeleteMarker` element to true in the `Expiration` action, you direct Amazon S3 to remove expired object delete markers.

Note

When specifying the `ExpiredObjectDeleteMarker` lifecycle action, the rule cannot specify a tag-based filter.

Example 8: Lifecycle Configuration to Abort Multipart Uploads

You can use the multipart upload API to upload large objects in parts. For more information about multipart uploads, see [Multipart Upload Overview \(p. 175\)](#).

Using lifecycle configuration, you can direct Amazon S3 to abort incomplete multipart uploads (identified by the key name prefix specified in the rule) if they don't complete within a specified number of days after initiation. When Amazon S3 aborts a multipart upload, it deletes all parts associated with the multipart upload. This ensures that you don't have incomplete multipart uploads with parts that are stored in Amazon S3 and, therefore, you don't have to pay any storage costs for these parts.

Note

When specifying the `AbortIncompleteMultipartUpload` lifecycle action, the rule cannot specify a tag-based filter.

The following is an example lifecycle configuration that specifies a rule with the `AbortIncompleteMultipartUpload` action. This action requests Amazon S3 to abort incomplete multipart uploads seven days after initiation.


```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Filter>
      <Prefix>SomeKeyPrefix</Prefix>
    </Filter>
    <Status>rule-status</Status>
    <AbortIncompleteMultipartUpload>
      <DaysAfterInitiation>7</DaysAfterInitiation>
    </AbortIncompleteMultipartUpload>
  </Rule>
</LifecycleConfiguration>
```

Setting Lifecycle Configuration on a Bucket

Topics

- [Manage an Object's Lifecycle Using the Amazon S3 Console \(p. 143\)](#)
- [Set Lifecycle Configurations Using the AWS CLI \(p. 144\)](#)
- [Managing Object Lifecycles Using the AWS SDK for Java \(p. 146\)](#)
- [Manage an Object's Lifecycle Using the AWS SDK for .NET \(p. 148\)](#)
- [Manage an Object's Lifecycle Using the AWS SDK for Ruby \(p. 151\)](#)
- [Manage an Object's Lifecycle Using the REST API \(p. 151\)](#)

This section explains how you can set lifecycle configuration on a bucket programmatically using AWS SDKs, or by using the Amazon S3 console, or the AWS CLI. Note the following:

- When you add a lifecycle configuration to a bucket, there is usually some lag before a new or updated lifecycle configuration is fully propagated to all the Amazon S3 systems. Expect a delay of a few minutes before the lifecycle configuration fully takes effect. This delay can also occur when you delete a lifecycle configuration.
- When you disable or delete a lifecycle rule, after a small delay Amazon S3 stops scheduling new objects for deletion or transition. Any objects that were already scheduled will be unscheduled and they won't be deleted or transitioned.
- When you add a lifecycle configuration to a bucket, the configuration rules apply to both existing objects and objects that you add later. For example, if you add a lifecycle configuration rule today with an expiration action that causes objects with a specific prefix to expire 30 days after creation, Amazon S3 will queue for removal any existing objects that are more than 30 days old.
- There may be a lag between when the lifecycle configuration rules are satisfied and when the action triggered by satisfying the rule is taken. However, changes in billing happen as soon as the lifecycle configuration rule is satisfied even if the action is not yet taken. One example is you are not charged for storage after the object expiration time even if the object is not deleted immediately. Another example is you are charged Amazon S3 Glacier storage rates as soon as the object transition time elapses, even if the object is not immediately transitioned to the GLACIER storage class. Lifecycle transitions to the INTELLIGENT_TIERING storage class are the exception and changes in billing do not happen until the object has transitioned into the INTELLIGENT_TIERING storage class.

For information about lifecycle configuration, see [Object Lifecycle Management \(p. 119\)](#).

Manage an Object's Lifecycle Using the Amazon S3 Console

You can specify lifecycle rules on a bucket using the Amazon S3 console.

For instructions on how to setup lifecycle rules using the AWS Management Console, see [How Do I Create a Lifecycle Policy for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Set Lifecycle Configurations Using the AWS CLI

You can use the following AWS CLI commands to manage lifecycle configurations:

- `put-bucket-lifecycle-configuration`
- `get-bucket-lifecycle-configuration`
- `delete-bucket-lifecycle`

For instructions to set up the AWS CLI, see [Setting Up the AWS CLI \(p. 675\)](#).

Note that the Amazon S3 lifecycle configuration is an XML file. But when using CLI, you cannot specify the XML, you must specify JSON instead. The following are examples XML lifecycle configurations and equivalent JSON that you can specify in AWS CLI command:

- Consider the following example lifecycle configuration:

```
<LifecycleConfiguration>
  <Rule>
    <ID>ExampleRule</ID>
    <Filter>
      <Prefix>documents/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

The equivalent JSON is shown:

```
{
  "Rules": [
    {
      "Filter": {
        "Prefix": "documents/"
      },
      "Status": "Enabled",
      "Transitions": [
        {
          "Days": 365,
          "StorageClass": "GLACIER"
        }
      ],
      "Expiration": {
        "Days": 3650
      },
      "ID": "ExampleRule"
    }
  ]
}
```

- Consider the following example lifecycle configuration:

```
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
```

```
<ID>id-1</ID>
<Expiration>
  <Days>1</Days>
</Expiration>
<Filter>
  <And>
    <Prefix>myprefix</Prefix>
    <Tag>
      <Key>mytagkey1</Key>
      <Value>mytagvalue1</Value>
    </Tag>
    <Tag>
      <Key>mytagkey2</Key>
      <Value>mytagvalue2</Value>
    </Tag>
  </And>
</Filter>
<Status>Enabled</Status>
</Rule>
</LifecycleConfiguration>
```

The equivalent JSON is shown:

```
{
  "Rules": [
    {
      "ID": "id-1",
      "Filter": {
        "And": {
          "Prefix": "myprefix",
          "Tags": [
            {
              "Value": "mytagvalue1",
              "Key": "mytagkey1"
            },
            {
              "Value": "mytagvalue2",
              "Key": "mytagkey2"
            }
          ]
        }
      },
      "Status": "Enabled",
      "Expiration": {
        "Days": 1
      }
    }
  ]
}
```

You can test the `put-bucket-lifecycle-configuration` as follows:

1. Save the JSON lifecycle configuration in a file (`lifecycle.json`).
2. Run the following AWS CLI command to set the lifecycle configuration on your bucket:

```
$ aws s3api put-bucket-lifecycle-configuration \
--bucket bucketname \
--lifecycle-configuration file://lifecycle.json
```

3. To verify, retrieve the lifecycle configuration using the `get-bucket-lifecycle-configuration` AWS CLI command as follows:

```
$ aws s3api get-bucket-lifecycle-configuration \
--bucket bucketname
```

4. To delete the lifecycle configuration use the `delete-bucket-lifecycle` AWS CLI command as follows:

```
aws s3api delete-bucket-lifecycle \
--bucket bucketname
```

Managing Object Lifecycles Using the AWS SDK for Java

You can use the AWS SDK for Java to manage the lifecycle configuration of a bucket. For more information about managing lifecycle configuration, see [Object Lifecycle Management \(p. 119\)](#).

Note

When you add a lifecycle configuration to a bucket, Amazon S3 replaces the bucket's current lifecycle configuration, if there is one. To update a configuration, you retrieve it, make the desired changes, and then add the revised lifecycle configuration to the bucket.

Example

The following example shows how to use the AWS SDK for Java to add, update, and delete the lifecycle configuration of a bucket. The example does the following:

- Adds a lifecycle configuration to a bucket.
- Retrieves the lifecycle configuration and updates it by adding another rule.
- Adds the modified lifecycle configuration to the bucket. Amazon S3 replaces the existing configuration.
- Retrieves the configuration again and verifies that it has the right number of rules by the printing number of rules.
- Deletes the lifecycle configuration and verifies that it has been deleted by attempting to retrieve it again.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration.Transition;
import com.amazonaws.services.s3.model.StorageClass;
import com.amazonaws.services.s3.model.Tag;
import com.amazonaws.services.s3.model.lifecycle.LifecycleAndOperator;
import com.amazonaws.services.s3.model.lifecycle.LifecycleFilter;
import com.amazonaws.services.s3.model.lifecycle.LifecyclePrefixPredicate;
import com.amazonaws.services.s3.model.lifecycle.LifecycleTagPredicate;

import java.io.IOException;
import java.util.Arrays;
```

```
public class LifecycleConfiguration {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        // Create a rule to archive objects with the "glacierobjects/" prefix to Glacier
        immediately.
        BucketLifecycleConfiguration.Rule rule1 = new BucketLifecycleConfiguration.Rule()
            .withId("Archive immediately rule")
            .withFilter(new LifecycleFilter(new LifecyclePrefixPredicate("glacierobjects/")))
            .addTransition(new
Transition().withDays(0).withStorageClass(StorageClass.Glacier))
            .withStatus(BucketLifecycleConfiguration.ENABLED);

        // Create a rule to transition objects to the Standard-Infrequent Access storage
        class
        // after 30 days, then to Glacier after 365 days. Amazon S3 will delete the objects
        after 3650 days.
        // The rule applies to all objects with the tag "archive" set to "true".
        BucketLifecycleConfiguration.Rule rule2 = new BucketLifecycleConfiguration.Rule()
            .withId("Archive and then delete rule")
            .withFilter(new LifecycleFilter(new LifecycleTagPredicate(new
Tag("archive", "true"))))
            .addTransition(new
Transition().withDays(30).withStorageClass(StorageClass.StandardInfrequentAccess))
            .addTransition(new
Transition().withDays(365).withStorageClass(StorageClass.Glacier))
            .withExpirationInDays(3650)
            .withStatus(BucketLifecycleConfiguration.ENABLED);

        // Add the rules to a new BucketLifecycleConfiguration.
        BucketLifecycleConfiguration configuration = new BucketLifecycleConfiguration()
            .withRules(Arrays.asList(rule1, rule2));

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Save the configuration.
            s3Client.setBucketLifecycleConfiguration(bucketName, configuration);

            // Retrieve the configuration.
            configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

            // Add a new rule with both a prefix predicate and a tag predicate.
            configuration.getRules().add(new
BucketLifecycleConfiguration.Rule().withId("NewRule")
                .withFilter(new LifecycleFilter(new LifecycleAndOperator(
                    Arrays.asList(new LifecyclePrefixPredicate("YearlyDocuments/"),
                        new LifecycleTagPredicate(new Tag("expire_after",
"ten_years"))))))
                .withExpirationInDays(3650)
                .withStatus(BucketLifecycleConfiguration.ENABLED));

            // Save the configuration.
            s3Client.setBucketLifecycleConfiguration(bucketName, configuration);

            // Retrieve the configuration.
            configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

            // Verify that the configuration now has three rules.
            configuration = s3Client.getBucketLifecycleConfiguration(bucketName);
        }
    }
}
```

```
        System.out.println("Expected # of rules = 3; found: " +
configuration.getRules().size());

        // Delete the configuration.
        s3Client.deleteBucketLifecycleConfiguration(bucketName);

        // Verify that the configuration has been deleted by attempting to retrieve it.
        configuration = s3Client.getBucketLifecycleConfiguration(bucketName);
        String s = (configuration == null) ? "No configuration found." : "Configuration
found.";
        System.out.println(s);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Manage an Object's Lifecycle Using the AWS SDK for .NET

You can use the AWS SDK for .NET to manage the lifecycle configuration on a bucket. For more information about managing lifecycle configuration, see [Object Lifecycle Management \(p. 119\)](#).

Note

When you add a lifecycle configuration, Amazon S3 replaces the existing lifecycle configuration on the specified bucket. To update a configuration, you must first retrieve the lifecycle configuration, make the changes, and then add the revised lifecycle configuration to the bucket.

Example .NET Code Example

The following example shows how to use the AWS SDK for .NET to add, update, and delete a bucket's lifecycle configuration. The code example does the following:

- Adds a lifecycle configuration to a bucket.
- Retrieves the lifecycle configuration and updates it by adding another rule.
- Adds the modified lifecycle configuration to the bucket. Amazon S3 replaces the existing lifecycle configuration.
- Retrieves the configuration again and verifies it by printing the number of rules in the configuration.
- Deletes the lifecycle configuration and verifies the deletion

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
```

```

class LifecycleTest
{
    private const string bucketName = "**** bucket name ****";
    // Specify your bucket region (an example region is shown).
    private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
    private static IAmazonS3 client;
    public static void Main()
    {
        client = new AmazonS3Client(bucketRegion);
        AddUpdateDeleteLifecycleConfigAsync().Wait();
    }

    private static async Task AddUpdateDeleteLifecycleConfigAsync()
    {
        try
        {
            var lifeCycleConfiguration = new LifecycleConfiguration()
            {
                Rules = new List<LifecycleRule>
                {
                    new LifecycleRule
                    {
                        Id = "Archive immediately rule",
                        Filter = new LifecycleFilter()
                        {
                            LifecycleFilterPredicate = new
LifecyclePrefixPredicate()
                            {
                                Prefix = "glacierobjects/"
                            }
                        },
                        Status = LifecycleRuleStatus.Enabled,
                        Transitions = new List<LifecycleTransition>
                        {
                            new LifecycleTransition
                            {
                                Days = 0,
                                StorageClass = S3StorageClass.Glacier
                            }
                        },
                    },
                    new LifecycleRule
                    {
                        Id = "Archive and then delete rule",
                        Filter = new LifecycleFilter()
                        {
                            LifecycleFilterPredicate = new
LifecyclePrefixPredicate()
                            {
                                Prefix = "projectdocs/"
                            }
                        },
                        Status = LifecycleRuleStatus.Enabled,
                        Transitions = new List<LifecycleTransition>
                        {
                            new LifecycleTransition
                            {
                                Days = 30,
                                StorageClass =
S3StorageClass.StandardInfrequentAccess
                            },
                            new LifecycleTransition
                            {
                                Days = 365,
                                StorageClass = S3StorageClass.Glacier
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        },
        Expiration = new LifecycleRuleExpiration()
        {
            Days = 3650
        }
    }
};

// Add the configuration to the bucket.
await AddExampleLifecycleConfigAsync(client, lifeCycleConfiguration);

// Retrieve an existing configuration.
lifeCycleConfiguration = await RetrieveLifecycleConfigAsync(client);

// Add a new rule.
lifeCycleConfiguration.Rules.Add(new LifecycleRule
{
    Id = "NewRule",
    Filter = new LifecycleFilter()
    {
        LifecycleFilterPredicate = new LifecyclePrefixPredicate()
        {
            Prefix = "YearlyDocuments/"
        }
    },
    Expiration = new LifecycleRuleExpiration()
    {
        Days = 3650
    }
});

// Add the configuration to the bucket.
await AddExampleLifecycleConfigAsync(client, lifeCycleConfiguration);

// Verify that there are now three rules.
lifeCycleConfiguration = await RetrieveLifecycleConfigAsync(client);
Console.WriteLine("Expected # of rulest=3; found:{0}",
lifeCycleConfiguration.Rules.Count);

// Delete the configuration.
await RemoveLifecycleConfigAsync(client);

// Retrieve a nonexistent configuration.
lifeCycleConfiguration = await RetrieveLifecycleConfigAsync(client);

}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
}
}

static async Task AddExampleLifecycleConfigAsync(IAmazonS3 client,
LifecycleConfiguration configuration)
{
    PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest
    {
        BucketName = bucketName,

```



```
        Configuration = configuration
    };
    var response = await client.PutLifecycleConfigurationAsync(request);
}

static async Task<LifecycleConfiguration> RetrieveLifecycleConfigAsync(IAmazonS3
client)
{
    GetLifecycleConfigurationRequest request = new GetLifecycleConfigurationRequest
    {
        BucketName = bucketName
    };
    var response = await client.GetLifecycleConfigurationAsync(request);
    var configuration = response.Configuration;
    return configuration;
}

static async Task RemoveLifecycleConfigAsync(IAmazonS3 client)
{
    DeleteLifecycleConfigurationRequest request = new
DeleteLifecycleConfigurationRequest
    {
        BucketName = bucketName
    };
    await client.DeleteLifecycleConfigurationAsync(request);
}
}
```

Manage an Object's Lifecycle Using the AWS SDK for Ruby

You can use the AWS SDK for Ruby to manage lifecycle configuration on a bucket by using the class [AWS::S3::BucketLifecycleConfiguration](#). For more information about using the AWS SDK for Ruby with Amazon S3, see [Using the AWS SDK for Ruby - Version 3](#) (p. 679). For more information about managing lifecycle configuration, see [Object Lifecycle Management](#) (p. 119).

Manage an Object's Lifecycle Using the REST API

You can use the AWS Management Console to set the lifecycle configuration on your bucket. If your application requires it, you can also send REST requests directly. The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API related to the lifecycle configuration.

- [PUT Bucket lifecycle](#)
- [GET Bucket lifecycle](#)
- [DELETE Bucket lifecycle](#)

Cross-Origin Resource Sharing (CORS)

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support, you can build rich client-side web applications with Amazon S3 and selectively allow cross-origin access to your Amazon S3 resources.

This section provides an overview of CORS. The subtopics describe how you can enable CORS using the Amazon S3 console, or programmatically by using the Amazon S3 REST API and the AWS SDKs.

Topics

- [Cross-Origin Resource Sharing: Use-case Scenarios \(p. 152\)](#)
- [How Do I Configure CORS on My Bucket? \(p. 152\)](#)
- [How Does Amazon S3 Evaluate the CORS Configuration on a Bucket? \(p. 154\)](#)
- [Enabling Cross-Origin Resource Sharing \(CORS\) \(p. 154\)](#)
- [Troubleshooting CORS Issues \(p. 160\)](#)

Cross-Origin Resource Sharing: Use-case Scenarios

The following are example scenarios for using CORS:

- Scenario 1: Suppose that you are hosting a website in an Amazon S3 bucket named `website` as described in [Hosting a Static Website on Amazon S3 \(p. 503\)](#). Your users load the website endpoint `http://website.s3-website-us-east-1.amazonaws.com`. Now you want to use JavaScript on the webpages that are stored in this bucket to be able to make authenticated GET and PUT requests against the same bucket by using the Amazon S3 API endpoint for the bucket, `website.s3.amazonaws.com`. A browser would normally block JavaScript from allowing those requests, but with CORS you can configure your bucket to explicitly enable cross-origin requests from `website.s3-website-us-east-1.amazonaws.com`.
- Scenario 2: Suppose that you want to host a web font from your S3 bucket. Again, browsers require a CORS check (also called a preflight check) for loading web fonts. You would configure the bucket that is hosting the web font to allow any origin to make these requests.

How Do I Configure CORS on My Bucket?

To configure your bucket to allow cross-origin requests, you create a CORS configuration, which is an XML document with rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) that will support for each origin, and other operation-specific information.

You can add up to 100 rules to the configuration. You add the XML document as the `cors` subresource to the bucket either programmatically or by using the Amazon S3 console. For more information, see [Enabling Cross-Origin Resource Sharing \(CORS\) \(p. 154\)](#).

Instead of accessing a website by using an Amazon S3 website endpoint, you can use your own domain, such as `example1.com` to serve your content. For information about using your own domain, see [Example: Setting up a Static Website Using a Custom Domain \(p. 519\)](#). The following example `cors` configuration has three rules, which are specified as `CORSRule` elements:

- The first rule allows cross-origin PUT, POST, and DELETE requests from the `http://www.example1.com` origin. The rule also allows all headers in a preflight OPTIONS request through the `Access-Control-Request-Headers` header. In response to preflight OPTIONS requests, Amazon S3 returns requested headers.
- The second rule allows the same cross-origin requests as the first rule, but the rule applies to another origin, `http://www.example2.com`.
- The third rule allows cross-origin GET requests from all origins. The `*` wildcard character refers to all origins.

```
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>http://www.example1.com</AllowedOrigin>

    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
```

```
<AllowedMethod>DELETE</AllowedMethod>

<AllowedHeader>*</AllowedHeader>
</CORSRule>
<CORSRule>
  <AllowedOrigin>http://www.example2.com</AllowedOrigin>

  <AllowedMethod>PUT</AllowedMethod>
  <AllowedMethod>POST</AllowedMethod>
  <AllowedMethod>DELETE</AllowedMethod>

  <AllowedHeader>*</AllowedHeader>
</CORSRule>
<CORSRule>
  <AllowedOrigin>*</AllowedOrigin>
  <AllowedMethod>GET</AllowedMethod>
</CORSRule>
</CORSConfiguration>
```

The CORS configuration also allows optional configuration parameters, as shown in the following CORS configuration. In this example, the CORS configuration allows cross-origin PUT, POST, and DELETE requests from the `http://www.example.com` origin.

```
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>http://www.example.com</AllowedOrigin>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <ExposeHeader>x-amz-server-side-encryption</
ExposeHeader>
    <ExposeHeader>x-amz-request-id</
ExposeHeader>
    <ExposeHeader>x-amz-id-2</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

The `CORSRule` element in the preceding configuration includes the following optional elements:

- **MaxAgeSeconds**—Specifies the amount of time in seconds (in this example, 3000) that the browser caches an Amazon S3 response to a preflight OPTIONS request for the specified resource. By caching the response, the browser does not have to send preflight requests to Amazon S3 if the original request will be repeated.
- **ExposeHeader**—Identifies the response headers (in this example, `x-amz-server-side-encryption`, `x-amz-request-id`, and `x-amz-id-2`) that customers are able to access from their applications (for example, from a JavaScript XMLHttpRequest object).

AllowedMethod Element

In the CORS configuration, you can specify the following values for the `AllowedMethod` element.

- GET
- PUT
- POST
- DELETE
- HEAD

AllowedOrigin Element

In the `AllowedOrigin` element, you specify the origins that you want to allow cross-domain requests from, for example, `http://www.example.com`. The origin string can contain only one `*` wildcard character, such as `http://*.example.com`. You can optionally specify `*` as the origin to enable all the origins to send cross-origin requests. You can also specify `https` to enable only secure origins.

AllowedHeader Element

The `AllowedHeader` element specifies which headers are allowed in a preflight request through the `Access-Control-Request-Headers` header. Each header name in the `Access-Control-Request-Headers` header must match a corresponding entry in the rule. Amazon S3 will send only the allowed headers in a response that were requested. For a sample list of headers that can be used in requests to Amazon S3, go to [Common Request Headers](#) in the *Amazon Simple Storage Service API Reference* guide.

Each `AllowedHeader` string in the rule can contain at most one `*` wildcard character. For example, `<AllowedHeader>x-amz-*</AllowedHeader>` will enable all Amazon-specific headers.

ExposeHeader Element

Each `ExposeHeader` element identifies a header in the response that you want customers to be able to access from their applications (for example, from a JavaScript `XMLHttpRequest` object). For a list of common Amazon S3 response headers, go to [Common Response Headers](#) in the *Amazon Simple Storage Service API Reference* guide.

MaxAgeSeconds Element

The `MaxAgeSeconds` element specifies the time in seconds that your browser can cache the response for a preflight request as identified by the resource, the HTTP method, and the origin.

How Does Amazon S3 Evaluate the CORS Configuration on a Bucket?

When Amazon S3 receives a preflight request from a browser, it evaluates the CORS configuration for the bucket and uses the first `CORSRule` rule that matches the incoming browser request to enable a cross-origin request. For a rule to match, the following conditions must be met:

- The request's `Origin` header must match an `AllowedOrigin` element.
- The request method (for example, GET or PUT) or the `Access-Control-Request-Method` header in case the of a preflight `OPTIONS` request must be one of the `AllowedMethod` elements.
- Every header listed in the request's `Access-Control-Request-Headers` header on the preflight request must match an `AllowedHeader` element.

Note

The ACLs and policies continue to apply when you enable CORS on the bucket.

Enabling Cross-Origin Resource Sharing (CORS)

Enable cross-origin resource sharing by setting a CORS configuration on your bucket using the AWS Management Console, the REST API, or the AWS SDKs.

Topics

- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the AWS Management Console \(p. 155\)](#)

- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the AWS SDK for Java \(p. 155\)](#)
- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the AWS SDK for .NET \(p. 157\)](#)
- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the REST API \(p. 160\)](#)

Enabling Cross-Origin Resource Sharing (CORS) Using the AWS Management Console

You can use the AWS Management Console to set a CORS configuration on your bucket. For instructions, see [How Do I Allow Cross-Domain Resource Sharing with CORS?](#) in the *Amazon Simple Storage Service Console User Guide*.

Enabling Cross-Origin Resource Sharing (CORS) Using the AWS SDK for Java

You can use the AWS SDK for Java to manage cross-origin resource sharing (CORS) for a bucket. For more information about CORS, see [Cross-Origin Resource Sharing \(CORS\) \(p. 151\)](#).

Example

The following example:

- Creates a CORS configuration and sets the configuration on a bucket
- Retrieves the configuration and modifies it by adding a rule
- Adds the modified configuration to the bucket
- Deletes the configuration

For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketCrossOriginConfiguration;
import com.amazonaws.services.s3.model.CORSRule;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class CORS {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";

        // Create two CORS rules.
        List<CORSRule.AllowedMethods> rule1AM = new ArrayList<CORSRule.AllowedMethods>();
        rule1AM.add(CORSRule.AllowedMethods.PUT);
        rule1AM.add(CORSRule.AllowedMethods.POST);
        rule1AM.add(CORSRule.AllowedMethods.DELETE);
        CORSRule rule1 = new CORSRule().withId("CORSRule1").withAllowedMethods(rule1AM)
```

```

        .withAllowedOrigins(Arrays.asList("http://*.example.com"));

List<CORSRule.AllowedMethods> rule2AM = new ArrayList<CORSRule.AllowedMethods>();
rule2AM.add(CORSRule.AllowedMethods.GET);
CORSRule rule2 = new CORSRule().withId("CORSRule2").withAllowedMethods(rule2AM)
    .withAllowedOrigins(Arrays.asList("*")).withMaxAgeSeconds(3000)
    .withExposedHeaders(Arrays.asList("x-amz-server-side-encryption"));

List<CORSRule> rules = new ArrayList<CORSRule>();
rules.add(rule1);
rules.add(rule2);

// Add the rules to a new CORS configuration.
BucketCrossOriginConfiguration configuration = new
BucketCrossOriginConfiguration();
configuration.setRules(rules);

try {
    AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(clientRegion)
        .build();

    // Add the configuration to the bucket.
    s3Client.setBucketCrossOriginConfiguration(bucketName, configuration);

    // Retrieve and display the configuration.
    configuration = s3Client.getBucketCrossOriginConfiguration(bucketName);
    printCORSConfiguration(configuration);

    // Add another new rule.
    List<CORSRule.AllowedMethods> rule3AM = new
ArrayList<CORSRule.AllowedMethods>();
    rule3AM.add(CORSRule.AllowedMethods.HEAD);
    CORSRule rule3 = new CORSRule().withId("CORSRule3").withAllowedMethods(rule3AM)
        .withAllowedOrigins(Arrays.asList("http://www.example.com"));

    rules = configuration.getRules();
    rules.add(rule3);
    configuration.setRules(rules);
    s3Client.setBucketCrossOriginConfiguration(bucketName, configuration);

    // Verify that the new rule was added by checking the number of rules in the
configuration.
    configuration = s3Client.getBucketCrossOriginConfiguration(bucketName);
    System.out.println("Expected # of rules = 3, found " +
configuration.getRules().size());

    // Delete the configuration.
    s3Client.deleteBucketCrossOriginConfiguration(bucketName);
    System.out.println("Removed CORS configuration.");

    // Retrieve and display the configuration to verify that it was
// successfully deleted.
    configuration = s3Client.getBucketCrossOriginConfiguration(bucketName);
    printCORSConfiguration(configuration);
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
// it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
// couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}

```

```

        private static void printCORSConfiguration(BucketCrossOriginConfiguration
configuration) {
            if (configuration == null) {
                System.out.println("Configuration is null.");
            } else {
                System.out.println("Configuration has " + configuration.getRules().size() + "
rules\n");

                for (CORSRule rule : configuration.getRules()) {
                    System.out.println("Rule ID: " + rule.getId());
                    System.out.println("MaxAgeSeconds: " + rule.getMaxAgeSeconds());
                    System.out.println("AllowedMethod: " + rule.getAllowedMethods());
                    System.out.println("AllowedOrigins: " + rule.getAllowedOrigins());
                    System.out.println("AllowedHeaders: " + rule.getAllowedHeaders());
                    System.out.println("ExposeHeader: " + rule.getExposedHeaders());
                    System.out.println();
                }
            }
        }
    }
}

```

Enabling Cross-Origin Resource Sharing (CORS) Using the AWS SDK for .NET

To manage cross-origin resource sharing (CORS) for a bucket, you can use the AWS SDK for .NET. For more information about CORS, see [Cross-Origin Resource Sharing \(CORS\) \(p. 151\)](#).

Example

The following C# code:

- Creates a CORS configuration and sets the configuration on a bucket
- Retrieves the configuration and modifies it by adding a rule
- Adds the modified configuration to the bucket
- Deletes the configuration

For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CORSTest
    {
        private const string bucketName = "**** bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);

```

```

        CORSConfigTestAsync().Wait();
    }
    private static async Task CORSConfigTestAsync()
    {
        try
        {
            // Create a new configuration request and add two rules
            CORSConfiguration configuration = new CORSConfiguration
            {
                Rules = new System.Collections.Generic.List<CORSRule>
                {
                    new CORSRule
                    {
                        Id = "CORSRule1",
                        AllowedMethods = new List<string> { "PUT", "POST", "DELETE" },
                        AllowedOrigins = new List<string> { "http://*.example.com" }
                    },
                    new CORSRule
                    {
                        Id = "CORSRule2",
                        AllowedMethods = new List<string> { "GET" },
                        AllowedOrigins = new List<string> { "*" },
                        MaxAgeSeconds = 3000,
                        ExposeHeaders = new List<string> { "x-amz-server-side-
encryption" }
                    }
                }
            };

            // Add the configuration to the bucket.
            await PutCORSConfigurationAsync(configuration);

            // Retrieve an existing configuration.
            configuration = await RetrieveCORSConfigurationAsync();

            // Add a new rule.
            configuration.Rules.Add(new CORSRule
            {
                Id = "CORSRule3",
                AllowedMethods = new List<string> { "HEAD" },
                AllowedOrigins = new List<string> { "http://www.example.com" }
            });

            // Add the configuration to the bucket.
            await PutCORSConfigurationAsync(configuration);

            // Verify that there are now three rules.
            configuration = await RetrieveCORSConfigurationAsync();
            Console.WriteLine();
            Console.WriteLine("Expected # of rulest=3; found:{0}",
configuration.Rules.Count);
            Console.WriteLine();
            Console.WriteLine("Pause before configuration delete. To continue, click
Enter...");
            Console.ReadKey();

            // Delete the configuration.
            await DeleteCORSConfigurationAsync();

            // Retrieve a nonexistent configuration.
            configuration = await RetrieveCORSConfigurationAsync();
        }
        catch (AmazonS3Exception e)
        {
            Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
        }
    }
}

```



```

    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}

static async Task PutCORSConfigurationAsync(CORSConfiguration configuration)
{
    PutCORSConfigurationRequest request = new PutCORSConfigurationRequest
    {
        BucketName = bucketName,
        Configuration = configuration
    };

    var response = await s3Client.PutCORSConfigurationAsync(request);
}

static async Task<CORSConfiguration> RetrieveCORSConfigurationAsync()
{
    GetCORSConfigurationRequest request = new GetCORSConfigurationRequest
    {
        BucketName = bucketName
    };

    var response = await s3Client.GetCORSConfigurationAsync(request);
    var configuration = response.Configuration;
    PrintCORSRules(configuration);
    return configuration;
}

static async Task DeleteCORSConfigurationAsync()
{
    DeleteCORSConfigurationRequest request = new DeleteCORSConfigurationRequest
    {
        BucketName = bucketName
    };
    await s3Client.DeleteCORSConfigurationAsync(request);
}

static void PrintCORSRules(CORSConfiguration configuration)
{
    Console.WriteLine();

    if (configuration == null)
    {
        Console.WriteLine("\nConfiguration is null");
        return;
    }

    Console.WriteLine("Configuration has {0} rules:", configuration.Rules.Count);
    foreach (CORSRule rule in configuration.Rules)
    {
        Console.WriteLine("Rule ID: {0}", rule.Id);
        Console.WriteLine("MaxAgeSeconds: {0}", rule.MaxAgeSeconds);
        Console.WriteLine("AllowedMethod: {0}", string.Join(", ",
rule.AllowedMethods.ToArray()));
        Console.WriteLine("AllowedOrigins: {0}", string.Join(", ",
rule.AllowedOrigins.ToArray()));
        Console.WriteLine("AllowedHeaders: {0}", string.Join(", ",
rule.AllowedHeaders.ToArray()));
        Console.WriteLine("ExposeHeader: {0}", string.Join(", ",
rule.ExposeHeaders.ToArray()));
    }
}

```

```
}  
}  
}
```

Enabling Cross-Origin Resource Sharing (CORS) Using the REST API

To set a CORS configuration on your bucket, you can use the AWS Management Console. If your application requires it, you can also send REST requests directly. The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API actions related to the CORS configuration:

- [PUT Bucket cors](#)
- [GET Bucket cors](#)
- [DELETE Bucket cors](#)
- [OPTIONS object](#)

Troubleshooting CORS Issues

If you encounter unexpected behavior while accessing buckets set with the CORS configuration, try the following steps to troubleshoot:

1. Verify that the CORS configuration is set on the bucket.

For instructions, see [Editing Bucket Permissions](#) in the *Amazon Simple Storage Service Console User Guide*. If the CORS configuration is set, the console displays an **Edit CORS Configuration** link in the **Permissions** section of the **Properties** bucket.

2. Capture the complete request and response using a tool of your choice. For each request Amazon S3 receives, there must be a CORS rule that matches the data in your request, as follows:
 - a. Verify that the request has the Origin header.

If the header is missing, Amazon S3 doesn't treat the request as a cross-origin request, and doesn't send CORS response headers in the response.

- b. Verify that the Origin header in your request matches at least one of the `AllowedOrigin` elements in the specified `CORSRule`.

The scheme, the host, and the port values in the Origin request header must match the `AllowedOrigin` elements in the `CORSRule`. For example, if you set the `CORSRule` to allow the origin `http://www.example.com`, then both `https://www.example.com` and `http://www.example.com:80` origins in your request don't match the allowed origin in your configuration.

- c. Verify that the method in your request (or in a preflight request, the method specified in the `Access-Control-Request-Method`) is one of the `AllowedMethod` elements in the same `CORSRule`.
- d. For a preflight request, if the request includes an `Access-Control-Request-Headers` header, verify that the `CORSRule` includes the `AllowedHeader` entries for each value in the `Access-Control-Request-Headers` header.

Operations on Objects

Amazon S3 enables you to store, retrieve, and delete objects. You can retrieve an entire object or a portion of an object. If you have enabled versioning on your bucket, you can retrieve a specific version

of the object. You can also retrieve a subresource associated with your object and update it where applicable. You can make a copy of your existing object. Depending on the object size, the following upload and copy related considerations apply:

- **Uploading objects**—You can upload objects of up to 5 GB in size in a single operation. For objects greater than 5 GB you must use the multipart upload API.

Using the multipart upload API you can upload objects up to 5 TB each. For more information, see [Uploading Objects Using Multipart Upload API \(p. 175\)](#).

- **Copying objects**—The copy operation creates a copy of an object that is already stored in Amazon S3.

You can create a copy of your object up to 5 GB in size in a single atomic operation. However, for copying an object greater than 5 GB, you must use the multipart upload API. For more information, see [Copying Objects \(p. 210\)](#).

You can use the REST API (see [Making Requests Using the REST API \(p. 44\)](#)) to work with objects or use one of the following AWS SDK libraries:

- [AWS SDK for Java](#)
- [AWS SDK for .NET](#)
- [AWS SDK for PHP](#)

These libraries provide a high-level abstraction that makes working with objects easy. However, if your application requires, you can use the REST API directly.

Getting Objects

Topics

- [Related Resources \(p. 162\)](#)
- [Get an Object Using the AWS SDK for Java \(p. 162\)](#)
- [Get an Object Using the AWS SDK for .NET \(p. 164\)](#)
- [Get an Object Using the AWS SDK for PHP \(p. 166\)](#)
- [Get an Object Using the REST API \(p. 167\)](#)
- [Share an Object with Others \(p. 167\)](#)

You can retrieve objects directly from Amazon S3. You have the following options when retrieving an object:

- **Retrieve an entire object**—A single GET operation can return you the entire object stored in Amazon S3.
- **Retrieve object in parts**—Using the Range HTTP header in a GET request, you can retrieve a specific range of bytes in an object stored in Amazon S3.

You resume fetching other parts of the object whenever your application is ready. This resumable download is useful when you need only portions of your object data. It is also useful where network connectivity is poor and you need to react to failures.

Note

Amazon S3 doesn't support retrieving multiple ranges of data per GET request.

When you retrieve an object, its metadata is returned in the response headers. There are times when you want to override certain response header values returned in a GET response. For example, you might override the `Content-Disposition` response header value in your GET request. The REST GET Object

API (see [GET Object](#)) allows you to specify query string parameters in your GET request to override these values.

The AWS SDKs for Java, .NET, and PHP also provide necessary objects you can use to specify values for these response headers in your GET request.

When retrieving objects that are stored encrypted using server-side encryption you will need to provide appropriate request headers. For more information, see [Protecting Data Using Encryption \(p. 264\)](#).

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Get an Object Using the AWS SDK for Java

When you download an object through the AWS SDK for Java, Amazon S3 returns all of the object's metadata and an input stream from which to read the object's contents.

To retrieve an object, you do the following:

- Execute the `AmazonS3Client.getObject()` method, providing the bucket name and object key in the request.
- Execute one of the `S3Object` instance methods to process the input stream.

Note

Your network connection remains open until you read all of the data or close the input stream. We recommend that you read the content of the stream as quickly as possible.

The following are some variations you might use:

- Instead of reading the entire object, you can read only a portion of the object data by specifying the byte range that you want in the request.
- You can optionally override the response header values (see [Getting Objects \(p. 161\)](#)) by using a `ResponseHeaderOverrides` object and setting the corresponding request property. For example, you can use this feature to indicate that the object should be downloaded into a file with a different file name than the object key name.

The following example retrieves an object from an Amazon S3 bucket three ways: first, as a complete object, then as a range of bytes from the object, then as a complete object with overridden response header values. For more information about getting objects from Amazon S3, see [GET Object](#).

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ResponseHeaderOverrides;
import com.amazonaws.services.s3.model.S3Object;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.io.InputStreamReader;

public class GetObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String key = "*** Object key ***";

        S3Object fullObject = null, objectPortion = null, headerOverrideObject = null;
        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Get an object and print its contents.
            System.out.println("Downloading an object");
            fullObject = s3Client.getObject(new GetObjectRequest(bucketName, key));
            System.out.println("Content-Type: " +
                fullObject.getObjectMetadata().getContentType());
            System.out.println("Content: ");
            displayTextInputStream(fullObject.getObjectContent());

            // Get a range of bytes from an object and print the bytes.
            GetObjectRequest rangeObjectRequest = new GetObjectRequest(bucketName, key)
                .withRange(0, 9);
            objectPortion = s3Client.getObject(rangeObjectRequest);
            System.out.println("Printing bytes retrieved.");
            displayTextInputStream(objectPortion.getObjectContent());

            // Get an entire object, overriding the specified response headers, and print
            the object's content.
            ResponseHeaderOverrides headerOverrides = new ResponseHeaderOverrides()
                .withCacheControl("No-cache")
                .withContentDisposition("attachment; filename=example.txt");
            GetObjectRequest getObjectRequestHeaderOverride = new
            GetObjectRequest(bucketName, key)
                .withResponseHeaders(headerOverrides);
            headerOverrideObject = s3Client.getObject(getObjectRequestHeaderOverride);
            displayTextInputStream(headerOverrideObject.getObjectContent());
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        } finally {
            // To ensure that the network connection doesn't remain open, close any open
            input streams.
            if (fullObject != null) {
                fullObject.close();
            }
            if (objectPortion != null) {
                objectPortion.close();
            }
            if (headerOverrideObject != null) {
                headerOverrideObject.close();
            }
        }
    }

    private static void displayTextInputStream(InputStream input) throws IOException {
        // Read the text input stream one line at a time and display each line.
    }
}
```

```
        BufferedReader reader = new BufferedReader(new InputStreamReader(input));
        String line = null;
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
        System.out.println();
    }
}
```

Get an Object Using the AWS SDK for .NET

When you download an object, you get all of the object's metadata and a stream from which to read the contents. You should read the content of the stream as quickly as possible because the data is streamed directly from Amazon S3 and your network connection will remain open until you read all the data or close the input stream. You do the following to get an object:

- Execute the `getObject` method by providing bucket name and object key in the request.
- Execute one of the `GetObjectResponse` methods to process the stream.

The following are some variations you might use:

- Instead of reading the entire object, you can read only the portion of the object data by specifying the byte range in the request, as shown in the following C# example:

Example

```
GetObjectRequest request = new GetObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    ByteRange = new ByteRange(0, 10)
};
```

- When retrieving an object, you can optionally override the response header values (see [Getting Objects \(p. 161\)](#)) by using the `ResponseHeaderOverrides` object and setting the corresponding request property. The following C# code example shows how to do this. For example, you can use this feature to indicate that the object should be downloaded into a file with a different filename than the object key name.

Example

```
GetObjectRequest request = new GetObjectRequest
{
    BucketName = bucketName,
    Key = keyName
};

ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
responseHeaders.CacheControl = "No-cache";
responseHeaders.ContentDisposition = "attachment; filename=testing.txt";

request.ResponseHeaderOverrides = responseHeaders;
```

Example

The following C# code example retrieves an object from an Amazon S3 bucket. From the response, the example reads the object data using the `GetObjectResponse.ResponseStream` property.

The example also shows how you can use the `GetObjectResponse.Metadata` collection to read object metadata. If the object you retrieve has the `x-amz-meta-title` metadata, the code prints the metadata value.

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class GetObjectTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** object key ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            ReadObjectDataAsync().Wait();
        }

        static async Task ReadObjectDataAsync()
        {
            string responseBody = "";
            try
            {
                GetObjectRequest request = new GetObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                };
                using (GetObjectResponse response = await client.GetObjectAsync(request))
                using (Stream responseStream = response.ResponseStream)
                using (StreamReader reader = new StreamReader(responseStream))
                {
                    string title = response.Metadata["x-amz-meta-title"]; // Assume you
                    have "title" as metadata added to the object.
                    string contentType = response.Headers["Content-Type"];
                    Console.WriteLine("Object metadata, Title: {0}", title);
                    Console.WriteLine("Content type: {0}", contentType);

                    responseBody = reader.ReadToEnd(); // Now you process the response
                    body.
                }
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
                object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
                writing an object", e.Message);
            }
        }
    }
}
```

```
}
}
```

Get an Object Using the AWS SDK for PHP

This topic explains how to use a class from the AWS SDK for PHP to retrieve an Amazon S3 object. You can retrieve an entire object or a byte range from the object. We assume that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

When retrieving an object, you can optionally override the response header values by adding the response keys, `ResponseContentType`, `ResponseContentLanguage`, `ResponseContentDisposition`, `ResponseCacheControl`, and `ResponseExpires`, to the `getObject()` method, as shown in the following PHP code example:

Example

```
$result = $s3->getObject([
    'Bucket'           => $bucket,
    'Key'              => $keyname,
    'ResponseContentType' => 'text/plain',
    'ResponseContentLanguage' => 'en-US',
    'ResponseContentDisposition' => 'attachment; filename=testing.txt',
    'ResponseCacheControl' => 'No-cache',
    'ResponseExpires'   => gmdate(DATE_RFC2822, time() + 3600),
]);
```

For more information about retrieving objects, see [Getting Objects \(p. 161\)](#).

The following PHP example retrieves an object and displays the content of the object in the browser. The example shows how to use the `getObject()` method. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

try {
    // Get the object.
    $result = $s3->getObject([
        'Bucket' => $bucket,
        'Key'    => $keyname
    ]);

    // Display the object in the browser.
    header("Content-Type: {$result['ContentType']}");
    echo $result['Body'];
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}
```


Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Get an Object Using the REST API

You can use the AWS SDK to retrieve object keys from a bucket. However, if your application requires it, you can send REST requests directly. You can send a GET request to retrieve object keys. For more information about the request and response format, go to [Get Object](#).

Share an Object with Others

Topics

- [Generate a Presigned Object URL using AWS Explorer for Visual Studio \(p. 167\)](#)
 - [Generate a presigned Object URL Using the AWS SDK for Java \(p. 167\)](#)
 - [Generate a Presigned Object URL Using AWS SDK for .NET \(p. 168\)](#)
-
- [Generate a Presigned Object URL Using AWS CLI](#)

All objects by default are private. Only the object owner has permission to access these objects. However, the object owner can optionally share objects with others by creating a presigned URL, using their own security credentials, to grant time-limited permission to download the objects.

When you create a presigned URL for your object, you must provide your security credentials, specify a bucket name, an object key, specify the HTTP method (GET to download the object) and expiration date and time. The presigned URLs are valid only for the specified duration.

Anyone who receives the presigned URL can then access the object. For example, if you have a video in your bucket and both the bucket and the object are private, you can share the video with others by generating a presigned URL.

Note

Anyone with valid security credentials can create a presigned URL. However, in order to successfully access an object, the presigned URL must be created by someone who has permission to perform the operation that the presigned URL is based upon.

You can generate presigned URL programmatically using the AWS SDK for Java and .NET.

Generate a Presigned Object URL using AWS Explorer for Visual Studio

If you are using Visual Studio, you can generate a presigned URL for an object without writing any code by using AWS Explorer for Visual Studio. Anyone with this URL can download the object. For more information, go to [Using Amazon S3 from AWS Explorer](#).

For instructions about how to install the AWS Explorer, see [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#).

Generate a presigned Object URL Using the AWS SDK for Java

Example

The following example generates a presigned URL that you can give to others so that they can retrieve an object from an S3 bucket. For more information, see [Share an Object with Others \(p. 167\)](#).

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;

import java.io.IOException;
import java.net.URL;

public class GeneratePresignedURL {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String objectKey = "*** Object key ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Set the presigned URL to expire after one hour.
            java.util.Date expiration = new java.util.Date();
            long expTimeMillis = expiration.getTime();
            expTimeMillis += 1000 * 60 * 60;
            expiration.setTime(expTimeMillis);

            // Generate the presigned URL.
            System.out.println("Generating pre-signed URL.");
            GeneratePresignedUrlRequest generatePresignedUrlRequest =
                new GeneratePresignedUrlRequest(bucketName, objectKey)
                    .withMethod(HttpMethod.GET)
                    .withExpiration(expiration);
            URL url = s3Client.generatePresignedUrl(generatePresignedUrlRequest);

            System.out.println("Pre-Signed URL: " + url.toString());
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Generate a Presigned Object URL Using AWS SDK for .NET

Example

The following example generates a presigned URL that you can give to others so that they can retrieve an object. For more information, see [Share an Object with Others \(p. 167\)](#).

For instructions about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;

namespace Amazon.DocSamples.S3
{
    class GenPresignedURLTest
    {
        private const string bucketName = "**** bucket name ****";
        private const string objectKey = "**** object key ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            string urlString = GeneratePreSignedURL();
        }
        static string GeneratePreSignedURL()
        {
            string urlString = "";
            try
            {
                GetPreSignedUrlRequest request1 = new GetPreSignedUrlRequest
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Expires = DateTime.Now.AddMinutes(5)
                };
                urlString = s3Client.GetPreSignedURL(request1);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when writing an object", e.Message);
            }
            return urlString;
        }
    }
}
```

Uploading Objects

Depending on the size of the data you are uploading, Amazon S3 offers the following options:

- **Upload objects in a single operation**—With a single PUT operation, you can upload objects up to 5 GB in size.

For more information, see [Uploading Objects in a Single Operation \(p. 170\)](#).

- **Upload objects in parts**—Using the multipart upload API, you can upload large objects, up to 5 TB.

The multipart upload API is designed to improve the upload experience for larger objects. You can upload objects in parts. These object parts can be uploaded independently, in any order, and in parallel. You can use a multipart upload for objects from 5 MB to 5 TB in size. For more information, see [Uploading Objects Using Multipart Upload API \(p. 175\)](#).

We recommend that you use multipart uploading in the following ways:

- If you're uploading large objects over a stable high-bandwidth network, use multipart uploading to maximize the use of your available bandwidth by uploading object parts in parallel for multi-threaded performance.
- If you're uploading over a spotty network, use multipart uploading to increase resiliency to network errors by avoiding upload restarts. When using multipart uploading, you need to retry uploading only parts that are interrupted during the upload. You don't need to restart uploading your object from the beginning.

For more information about multipart uploads, see [Multipart Upload Overview \(p. 175\)](#).

Topics

- [Uploading Objects in a Single Operation \(p. 170\)](#)
- [Uploading Objects Using Multipart Upload API \(p. 175\)](#)
- [Uploading Objects Using Presigned URLs \(p. 206\)](#)

When uploading an object, you can optionally request that Amazon S3 encrypt it before saving it to disk, and decrypt it when you download it. For more information, see [Protecting Data Using Encryption \(p. 264\)](#).

Related Topics

[Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Uploading Objects in a Single Operation

Topics

- [Upload an Object Using the AWS SDK for Java \(p. 170\)](#)
- [Upload an Object Using the AWS SDK for .NET \(p. 171\)](#)
- [Upload an Object Using the AWS SDK for PHP \(p. 173\)](#)
- [Upload an Object Using the AWS SDK for Ruby \(p. 173\)](#)
- [Upload an Object Using the REST API \(p. 174\)](#)

You can use the AWS SDK to upload objects. The SDK provides wrapper libraries for you to upload data easily. However, if your application requires it, you can use the REST API directly in your application.

Upload an Object Using the AWS SDK for Java

Example

The following example creates two objects. The first object has a text string as data, and the second object is a file. The example creates the first object by specifying the bucket name, object key, and text data directly in a call to `AmazonS3Client.putObject()`. The example creates the second object by using a `PutObjectRequest` that specifies the bucket name, object key, and file path. The `PutObjectRequest` also specifies the `ContentType` header and title metadata.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.regions.Regions;
```

```
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;

import java.io.File;
import java.io.IOException;

public class UploadObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String stringObjKeyName = "*** String object key name ***";
        String fileObjKeyName = "*** File object key name ***";
        String fileName = "*** Path to file to upload ***";

        try {
            //This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .build();

            // Upload a text string as a new object.
            s3Client.putObject(bucketName, stringObjKeyName, "Uploaded String Object");

            // Upload a file as a new object with ContentType and title specified.
            PutObjectRequest request = new PutObjectRequest(bucketName, fileObjKeyName, new
File(fileName));
            ObjectMetadata metadata = new ObjectMetadata();
            metadata.setContentType("plain/text");
            metadata.addUserMetadata("x-amz-meta-title", "someTitle");
            request.setMetadata(metadata);
            s3Client.putObject(request);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Upload an Object Using the AWS SDK for .NET

Example

The following C# code example creates two objects with two `PutObjectRequest` requests:

- The first `PutObjectRequest` request saves a text string as sample object data. It also specifies the bucket and object key names.
- The second `PutObjectRequest` request uploads a file by specifying the file name. This request also specifies the `ContentType` header and optional object metadata (a title).

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadObjectTest
    {
        private const string bucketName = "*** bucket name ***";
        // For simplicity the example creates two objects from the same file.
        // You specify key names for these objects.
        private const string keyName1 = "*** key name for first object created ***";
        private const string keyName2 = "*** key name for second object created ***";
        private const string filePath = @ "*** file path ***";
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.EUWest1;

        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            WritingAnObjectAsync().Wait();
        }

        static async Task WritingAnObjectAsync()
        {
            try
            {
                // 1. Put object-specify only key name for the new object.
                var putRequest1 = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName1,
                    ContentBody = "sample text"
                };

                PutObjectResponse response1 = await client.PutObjectAsync(putRequest1);

                // 2. Put the object-set ContentType and add metadata.
                var putRequest2 = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName2,
                    FilePath = filePath,
                    ContentType = "text/plain"
                };
                putRequest2.Metadata.Add("x-amz-meta-title", "someTitle");
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine(
                    "Error encountered ***. Message:'{0}' when writing an object"
                    , e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine(
                    "Unknown encountered on server. Message:'{0}' when writing an object"
                    , e.Message);
            }
        }
    }
}

```

```
}
```

Upload an Object Using the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to upload an object of up to 5 GB in size. For larger files you must use multipart upload API. For more information, see [Uploading Objects Using Multipart Upload API \(p. 175\)](#).

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

Example of Creating an Object in an Amazon S3 bucket by Uploading Data

The following PHP example creates an object in a specified bucket by uploading data using the `putObject()` method. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 679\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

try {
    // Upload data.
    $result = $s3->putObject([
        'Bucket' => $bucket,
        'Key'    => $keyname,
        'Body'   => 'Hello, world!',
        'ACL'    => 'public-read'
    ]);

    // Print the URL to the object.
    echo $result['ObjectURL'] . PHP_EOL;
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Upload an Object Using the AWS SDK for Ruby

The AWS SDK for Ruby - Version 3 has two ways of uploading an object to Amazon S3. The first uses a managed file uploader, which makes it easy to upload files of any size from disk. To use the managed file uploader method:

1. Create an instance of the `Aws::S3::Resource` class.
2. Reference the target object by bucket name and key. Objects live in a bucket and have unique keys that identify each object.

3. Call `#upload_file` on the object.

Example

```
require 'aws-sdk-s3'

s3 = Aws::S3::Resource.new(region: 'us-west-2')
obj = s3.bucket('bucket-name').object('key')
obj.upload_file('/path/to/source/file')
```

The second way that AWS SDK for Ruby - Version 3 can upload an object uses the `#put` method of `Aws::S3::Object`. This is useful if the object is a string or an I/O object that is not a file on disk. To use this method:

1. Create an instance of the `Aws::S3::Resource` class.
2. Reference the target object by bucket name and key.
3. Call `#put`, passing in the string or I/O object.

Example

```
require 'aws-sdk-s3'

s3 = Aws::S3::Resource.new(region: 'us-west-2')
obj = s3.bucket('bucket-name').object('key')

# I/O object
File.open('/path/to/source.file', 'rb') do |file|
  obj.put(body: file)
end
```

Upload an Object Using the REST API

You can use AWS SDK to upload an object. However, if your application requires it, you can send REST requests directly. You can send a PUT request to upload data in a single operation. For more information, see [PUT Object](#).

Uploading Objects Using Multipart Upload API

Topics

- [Multipart Upload Overview \(p. 175\)](#)
- [Using the AWS Java SDK for Multipart Upload \(High-Level API\) \(p. 182\)](#)
- [Using the AWS Java SDK for a Multipart Upload \(Low-Level API\) \(p. 186\)](#)
- [Using the AWS SDK for .NET for Multipart Upload \(High-Level API\) \(p. 191\)](#)
- [Using the AWS SDK for .NET for Multipart Upload \(Low-Level API\) \(p. 197\)](#)
- [Using the AWS PHP SDK for Multipart Upload \(p. 201\)](#)
- [Using the AWS PHP SDK for Multipart Upload \(Low-Level API\) \(p. 203\)](#)
- [Using the AWS SDK for Ruby for Multipart Upload \(p. 205\)](#)
- [Using the REST API for Multipart Upload \(p. 206\)](#)

Multipart upload allows you to upload a single object as a set of parts. Each part is a contiguous portion of the object's data. You can upload these object parts independently and in any order. If transmission of any part fails, you can retransmit that part without affecting other parts. After all parts of your object are uploaded, Amazon S3 assembles these parts and creates the object. In general, when your object size reaches 100 MB, you should consider using multipart uploads instead of uploading the object in a single operation.

Using multipart upload provides the following advantages:

- Improved throughput - You can upload parts in parallel to improve throughput.
- Quick recovery from any network issues - Smaller part size minimizes the impact of restarting a failed upload due to a network error.
- Pause and resume object uploads - You can upload object parts over time. Once you initiate a multipart upload there is no expiry; you must explicitly complete or abort the multipart upload.
- Begin an upload before you know the final object size - You can upload an object as you are creating it.

For more information, see [Multipart Upload Overview \(p. 175\)](#).

Multipart Upload Overview

Topics

- [Concurrent Multipart Upload Operations \(p. 176\)](#)
- [Multipart Upload and Pricing \(p. 177\)](#)
- [Aborting Incomplete Multipart Uploads Using a Bucket Lifecycle Policy \(p. 177\)](#)
- [Amazon S3 Multipart Upload Limits \(p. 178\)](#)
- [API Support for Multipart Upload \(p. 179\)](#)
- [Multipart Upload API and Permissions \(p. 179\)](#)

The Multipart upload API enables you to upload large objects in parts. You can use this API to upload new large objects or make a copy of an existing object (see [Operations on Objects \(p. 160\)](#)).

Multipart uploading is a three-step process: You initiate the upload, you upload the object parts, and after you have uploaded all the parts, you complete the multipart upload. Upon receiving the complete multipart upload request, Amazon S3 constructs the object from the uploaded parts, and you can then access the object just as you would any other object in your bucket.

You can list all of your in-progress multipart uploads or get a list of the parts that you have uploaded for a specific multipart upload. Each of these operations is explained in this section.

Multipart Upload Initiation

When you send a request to initiate a multipart upload, Amazon S3 returns a response with an upload ID, which is a unique identifier for your multipart upload. You must include this upload ID whenever you upload parts, list the parts, complete an upload, or abort an upload. If you want to provide any metadata describing the object being uploaded, you must provide it in the request to initiate multipart upload.

Parts Upload

When uploading a part, in addition to the upload ID, you must specify a part number. You can choose any part number between 1 and 10,000. A part number uniquely identifies a part and its position in the object you are uploading. The part number that you choose doesn't need to be in a consecutive sequence (for example, it can be 1, 5, and 14). If you upload a new part using the same part number as a previously uploaded part, the previously uploaded part is overwritten. Whenever you upload a part, Amazon S3 returns an *ETag* header in its response. For each part upload, you must record the part number and the ETag value. You need to include these values in the subsequent request to complete the multipart upload.

Note

After you initiate a multipart upload and upload one or more parts, you must either complete or abort the multipart upload in order to stop getting charged for storage of the uploaded parts. Only *after* you either complete or abort a multipart upload will Amazon S3 free up the parts storage and stop charging you for the parts storage.

Multipart Upload Completion (or Abort)

When you complete a multipart upload, Amazon S3 creates an object by concatenating the parts in ascending order based on the part number. If any object metadata was provided in the *initiate multipart upload* request, Amazon S3 associates that metadata with the object. After a successful *complete* request, the parts no longer exist. Your *complete multipart upload* request must include the upload ID and a list of both part numbers and corresponding ETag values. Amazon S3 response includes an ETag that uniquely identifies the combined object data. This ETag will not necessarily be an MD5 hash of the object data. You can optionally abort the multipart upload. After aborting a multipart upload, you cannot upload any part using that upload ID again. All storage that any parts from the aborted multipart upload consumed is then freed. If any part uploads were in-progress, they can still succeed or fail even after you aborted. To free all storage consumed by all parts, you must abort a multipart upload only after all part uploads have completed.

Multipart Upload Listings

You can list the parts of a specific multipart upload or all in-progress multipart uploads. The list parts operation returns the parts information that you have uploaded for a specific multipart upload. For each list parts request, Amazon S3 returns the parts information for the specified multipart upload, up to a maximum of 1,000 parts. If there are more than 1,000 parts in the multipart upload, you must send a series of list part requests to retrieve all the parts. Note that the returned list of parts doesn't include parts that haven't completed uploading. Using the *list multipart uploads* operation, you can obtain a list of multipart uploads in progress. An in-progress multipart upload is an upload that you have initiated, but have not yet completed or aborted. Each request returns at most 1000 multipart uploads. If there are more than 1,000 multipart uploads in progress, you need to send additional requests to retrieve the remaining multipart uploads. Only use the returned listing for verification. You should not use the result of this listing when sending a *complete multipart upload* request. Instead, maintain your own list of the part numbers you specified when uploading parts and the corresponding ETag values that Amazon S3 returns.

Concurrent Multipart Upload Operations

In a distributed development environment, it is possible for your application to initiate several updates on the same object at the same time. Your application might initiate several multipart uploads using the same object key. For each of these uploads, your application can then upload parts and send a complete upload request to Amazon S3 to create the object. When the buckets have versioning enabled,

completing a multipart upload always creates a new version. For buckets that do not have versioning enabled, it is possible that some other request received between the time when a multipart upload is initiated and when it is completed might take precedence.

Note

It is possible for some other request received between the time you initiated a multipart upload and completed it to take precedence. For example, if another operation deletes a key after you initiate a multipart upload with that key, but before you complete it, the complete multipart upload response might indicate a successful object creation without you ever seeing the object.

Multipart Upload and Pricing

Once you initiate a multipart upload, Amazon S3 retains all the parts until you either complete or abort the upload. Throughout its lifetime, you are billed for all storage, bandwidth, and requests for this multipart upload and its associated parts. If you abort the multipart upload, Amazon S3 deletes upload artifacts and any parts that you have uploaded, and you are no longer billed for them. For more information about pricing, see [Amazon S3 Pricing](#).

Aborting Incomplete Multipart Uploads Using a Bucket Lifecycle Policy

After you initiate a multipart upload, you begin uploading parts. Amazon S3 stores these parts, but it creates the object from the parts only after you upload all of them and send a successful request to complete the multipart upload (you should verify that your request to complete multipart upload is successful). Upon receiving the complete multipart upload request, Amazon S3 assembles the parts and creates an object.

If you don't send the complete multipart upload request successfully, Amazon S3 will not assemble the parts and will not create any object. Therefore, the parts remain in Amazon S3 and you pay for the parts that are stored in Amazon S3. As a best practice, we recommend you configure a lifecycle rule (using the `AbortIncompleteMultipartUpload` action) to minimize your storage costs.

Amazon S3 supports a bucket lifecycle rule that you can use to direct Amazon S3 to abort multipart uploads that don't complete within a specified number of days after being initiated. When a multipart upload is not completed within the time frame, it becomes eligible for an abort operation and Amazon S3 aborts the multipart upload (and deletes the parts associated with the multipart upload).

The following is an example lifecycle configuration that specifies a rule with the `AbortIncompleteMultipartUpload` action.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Prefix></Prefix>
    <Status>Enabled</Status>
    <AbortIncompleteMultipartUpload>
      <DaysAfterInitiation>7</DaysAfterInitiation>
    </AbortIncompleteMultipartUpload>
  </Rule>
</LifecycleConfiguration>
```

In the example, the rule does not specify a value for the `Prefix` element (object key name prefix) and therefore it applies to all objects in the bucket for which you initiated multipart uploads. Any multipart uploads that were initiated and did not complete within seven days become eligible for an abort operation (the action has no effect on completed multipart uploads).

For more information about the bucket lifecycle configuration, see [Object Lifecycle Management](#) (p. 119).

Note

If the multipart upload is completed within the number of days specified in the rule, the `AbortIncompleteMultipartUpload` lifecycle action does not apply (that is, Amazon S3 will

not take any action). Also, this action does not apply to objects, no objects are deleted by this lifecycle action.

The following `put-bucket-lifecycle` CLI command adds the lifecycle configuration for the specified bucket.

```
$ aws s3api put-bucket-lifecycle \
    --bucket bucketname \
    --lifecycle-configuration filename-containing-lifecycle-configuration
```

To test the CLI command, do the following:

1. Set up the AWS CLI. For instructions, see [Setting Up the AWS CLI \(p. 675\)](#).
2. Save the following example lifecycle configuration in a file (`lifecycle.json`). The example configuration specifies empty prefix and therefore it applies to all objects in the bucket. You can specify a prefix to restrict the policy to a subset of objects.

```
{
  "Rules": [
    {
      "ID": "Test Rule",
      "Status": "Enabled",
      "Prefix": "",
      "AbortIncompleteMultipartUpload": {
        "DaysAfterInitiation": 7
      }
    }
  ]
}
```

3. Run the following CLI command to set lifecycle configuration on your bucket.

```
aws s3api put-bucket-lifecycle \
  --bucket bucketname \
  --lifecycle-configuration file:///lifecycle.json
```

4. To verify, retrieve the lifecycle configuration using the `get-bucket-lifecycle` CLI command.

```
aws s3api get-bucket-lifecycle \
  --bucket bucketname
```

5. To delete the lifecycle configuration use the `delete-bucket-lifecycle` CLI command.

```
aws s3api delete-bucket-lifecycle \
  --bucket bucketname
```

Amazon S3 Multipart Upload Limits

The following table provides multipart upload core specifications. For more information, see [Multipart Upload Overview \(p. 175\)](#).

Item	Specification
Maximum object size	5 TB
Maximum number of parts per upload	10,000

Item	Specification
Part numbers	1 to 10,000 (inclusive)
Part size	5 MB to 5 GB, last part can be < 5 MB
Maximum number of parts returned for a list parts request	1000
Maximum number of multipart uploads returned in a list multipart uploads request	1000

API Support for Multipart Upload

You can use an AWS SDK to upload an object in parts. The following AWS SDK libraries support multipart upload:

- [AWS SDK for Java](#)
- [AWS SDK for .NET](#)
- [AWS SDK for PHP](#)

These libraries provide a high-level abstraction that makes uploading multipart objects easy. However, if your application requires, you can use the REST API directly. The following sections in the Amazon Simple Storage Service API Reference describe the REST API for multipart upload.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

Multipart Upload API and Permissions

An individual must have the necessary permissions to use the multipart upload operations. You can use ACLs, the bucket policy, or the user policy to grant individuals permissions to perform these operations. The following table lists the required permissions for various multipart upload operations when using ACLs, bucket policy, or the user policy.

Action	Required Permissions
Initiate Multipart Upload	You must be allowed to perform the <code>s3:PutObject</code> action on an object to initiate multipart upload. The bucket owner can allow other principals to perform the <code>s3:PutObject</code> action.
Initiator	Container element that identifies who initiated the multipart upload. If the initiator is an AWS account, this element provides the same information as the Owner element. If the initiator is an IAM User, this element provides the user ARN and display name.
Upload Part	You must be allowed to perform the <code>s3:PutObject</code> action on an object to upload a part.

Action	Required Permissions
	Only the initiator of a multipart upload can upload parts. The bucket owner must allow the initiator to perform the <code>s3:PutObject</code> action on an object in order for the initiator to upload a part for that object.
Upload Part (Copy)	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to upload a part. Because you are uploading a part from an existing object, you must be allowed <code>s3:GetObject</code> on the source object.</p> <p>Only the initiator of a multipart upload can upload parts. For the initiator to upload a part for an object, the owner of the bucket must allow the initiator to perform the <code>s3:PutObject</code> action on the object.</p>
Complete Multipart Upload	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to complete a multipart upload.</p> <p>Only the initiator of a multipart upload can complete that multipart upload. The bucket owner must allow the initiator to perform the <code>s3:PutObject</code> action on an object in order for the initiator to complete a multipart upload for that object.</p>
Abort Multipart Upload	<p>You must be allowed to perform the <code>s3:AbortMultipartUpload</code> action to abort a multipart upload.</p> <p>By default, the bucket owner and the initiator of the multipart upload are allowed to perform this action. If the initiator is an IAM user, that user's AWS account is also allowed to abort that multipart upload.</p> <p>In addition to these defaults, the bucket owner can allow other principals to perform the <code>s3:AbortMultipartUpload</code> action on an object. The bucket owner can deny any principal the ability to perform the <code>s3:AbortMultipartUpload</code> action.</p>
List Parts	<p>You must be allowed to perform the <code>s3:ListMultipartUploadParts</code> action to list parts in a multipart upload.</p> <p>By default, the bucket owner has permission to list parts for any multipart upload to the bucket. The initiator of the multipart upload has the permission to list parts of the specific multipart upload. If the multipart upload initiator is an IAM user, the AWS account controlling that IAM user also has permission to list parts of that upload.</p> <p>In addition to these defaults, the bucket owner can allow other principals to perform the <code>s3:ListMultipartUploadParts</code> action on an object. The bucket owner can also deny any principal the ability to perform the <code>s3:ListMultipartUploadParts</code> action.</p>
List Multipart Uploads	<p>You must be allowed to perform the <code>s3:ListBucketMultipartUploads</code> action on a bucket to list multipart uploads in progress to that bucket.</p> <p>In addition to the default, the bucket owner can allow other principals to perform the <code>s3:ListBucketMultipartUploads</code> action on the bucket.</p>

Action	Required Permissions
KMS Encrypt and Decrypt related permissions	<p>To perform a multipart upload with encryption using an AWS KMS key, the requester must have permission to the <code>kms:Encrypt</code>, <code>kms:Decrypt</code>, <code>kms:ReEncrypt*</code>, <code>kms:GenerateDataKey*</code>, and <code>kms:DescribeKey</code> actions on the key. These permissions are required because Amazon S3 must decrypt and read data from the encrypted file parts before it completes the multipart upload.</p> <p>If your IAM user or role is in the same AWS account as the AWS KMS key, then you must have these permissions on the key policy. If your IAM user or role belongs to a different account than the key, then you must have the permissions on both the key policy and your IAM user or role.</p> <p>></p>

For information on the relationship between ACL permissions and permissions in access policies, see [Mapping of ACL Permissions and Access Policy Permissions \(p. 406\)](#). For information on IAM users, go to [Working with Users and Groups](#).

Using the AWS Java SDK for Multipart Upload (High-Level API)

Topics

- [Upload a File \(p. 182\)](#)
- [Abort Multipart Uploads \(p. 183\)](#)
- [Track Multipart Upload Progress \(p. 184\)](#)

The AWS SDK for Java exposes a high-level API, called `TransferManager`, that simplifies multipart uploads (see [Uploading Objects Using Multipart Upload API \(p. 175\)](#)). You can upload data from a file or a stream. You can also set advanced options, such as the part size you want to use for the multipart upload, or the number of concurrent threads you want to use when uploading the parts. You can also set optional object properties, the storage class, or the ACL. You use the `PutObjectRequest` and the `TransferManagerConfiguration` classes to set these advanced options.

When possible, `TransferManager` attempts to use multiple threads to upload multiple parts of a single upload at once. When dealing with large content sizes and high bandwidth, this can increase throughput significantly.

In addition to file-upload functionality, the `TransferManager` class enables you to abort an in-progress multipart upload. An upload is considered to be in progress after you initiate it and until you complete or abort it. The `TransferManager` aborts all in-progress multipart uploads on a specified bucket that were initiated before a specified date and time.

For more information about multipart uploads, including additional functionality offered by the low-level API methods, see [Uploading Objects Using Multipart Upload API \(p. 175\)](#).

Upload a File

Example

The following example shows how to upload an object using the high-level multipart-upload Java API (the `TransferManager` class). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;

public class HighLevelMultipartUpload {

    public static void main(String[] args) throws Exception {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String keyName = "**** Object key ****";
        String filePath = "**** Path for file to upload ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            TransferManager transferManager = TransferManagerBuilder.standard()
                .withS3Client(s3Client)
                .build();

            Upload upload = transferManager.upload(filePath, bucketName, keyName);
            upload.waitForCompletion();
        } catch (AmazonServiceException e) {
            // Amazon S3 error
        } catch (SdkClientException e) {
            // Client error
        }
    }
}
```



```
TransferManager tm = TransferManagerBuilder.standard()
    .withS3Client(s3Client)
    .build();

// TransferManager processes all transfers asynchronously,
// so this call returns immediately.
Upload upload = tm.upload(bucketName, keyName, new File(filePath));
System.out.println("Object upload started");

// Optionally, wait for the upload to finish before continuing.
upload.waitForCompletion();
System.out.println("Object upload complete");
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Abort Multipart Uploads

Example

The following example uses the high-level Java API (the `TransferManager` class) to abort all in-progress multipart uploads that were initiated on a specific bucket over a week ago. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;

import java.util.Date;

public class HighLevelAbortMultipartUpload {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            TransferManager tm = TransferManagerBuilder.standard()
                .withS3Client(s3Client)
                .build();

            // sevenDays is the duration of seven days in milliseconds.
            long sevenDays = 1000 * 60 * 60 * 24 * 7;
            Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDays);
            tm.abortMultipartUploads(bucketName, oneWeekAgo);
        }
```

```
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client couldn't
        // parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Track Multipart Upload Progress

The high-level Java multipart upload API provides a listen interface, `ProgressListener`, to track progress when uploading an object to Amazon S3. Progress events periodically notify the listener that bytes have been transferred.

The following example demonstrates how to subscribe to a `ProgressEvent` event and write a handler:

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;

public class HighLevelTrackMultipartUpload {

    public static void main(String[] args) throws Exception {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Object key ***";
        String filePath = "*** Path to file to upload ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            TransferManager tm = TransferManagerBuilder.standard()
                .withS3Client(s3Client)
                .build();

            PutObjectRequest request = new PutObjectRequest(bucketName, keyName, new
            File(filePath));

            // To receive notifications when bytes are transferred, add a
            // ProgressListener to your request.
            request.setGeneralProgressListener(new ProgressListener() {
                public void progressChanged(ProgressEvent progressEvent) {
                    System.out.println("Transferred bytes: " +
            progressEvent.getBytesTransferred());
                }
            });
        }
    }
}
```

```
        }  
    });  
    // TransferManager processes all transfers asynchronously,  
    // so this call returns immediately.  
    Upload upload = tm.upload(request);  
  
    // Optionally, you can wait for the upload to finish before continuing.  
    upload.waitForCompletion();  
} catch (AmazonServiceException e) {  
    // The call was transmitted successfully, but Amazon S3 couldn't process  
    // it, so it returned an error response.  
    e.printStackTrace();  
} catch (SdkClientException e) {  
    // Amazon S3 couldn't be contacted for a response, or the client  
    // couldn't parse the response from Amazon S3.  
    e.printStackTrace();  
}  
}  
}
```

Using the AWS Java SDK for a Multipart Upload (Low-Level API)

Topics

- [Upload a File \(p. 186\)](#)
- [List Multipart Uploads \(p. 188\)](#)
- [Abort a Multipart Upload \(p. 188\)](#)

The AWS SDK for Java exposes a low-level API that closely resembles the Amazon S3 REST API for multipart uploads (see [Uploading Objects Using Multipart Upload API \(p. 175\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the upload data in advance. When you don't have these requirements, use the high-level API (see [Using the AWS Java SDK for Multipart Upload \(High-Level API\) \(p. 182\)](#)).

Upload a File

The following example shows how to use the low-level Java classes to upload a file. It performs the following steps:

- Initiates a multipart upload using the `AmazonS3Client.initiateMultipartUpload()` method, and passes in an `InitiateMultipartUploadRequest` object.
- Saves the upload ID that the `AmazonS3Client.initiateMultipartUpload()` method returns. You provide this upload ID for each subsequent multipart upload operation.
- Uploads the parts of the object. For each part, you call the `AmazonS3Client.uploadPart()` method. You provide part upload information using an `UploadPartRequest` object.
- For each part, saves the ETag from the response of the `AmazonS3Client.uploadPart()` method in a list. You use the ETag values to complete the multipart upload.
- Calls the `AmazonS3Client.completeMultipartUpload()` method to complete the multipart upload.

Example

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class LowLevelMultipartUpload {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";
        String filePath = "*** Path to file to upload ***";

        File file = new File(filePath);
        long contentLength = file.length();
```

```
        long partSize = 5 * 1024 * 1024; // Set part size to 5 MB.

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Create a list of ETag objects. You retrieve ETags for each object part
            // then, after each individual part has been uploaded, pass the list of ETags
            // the request to complete the upload.
            List<PartETag> partETags = new ArrayList<PartETag>();

            // Initiate the multipart upload.
            InitiateMultipartUploadRequest initRequest = new
            InitiateMultipartUploadRequest(bucketName, keyName);
            InitiateMultipartUploadResult initResponse =
            s3Client.initiateMultipartUpload(initRequest);

            // Upload the file parts.
            long filePosition = 0;
            for (int i = 1; filePosition < contentLength; i++) {
                // Because the last part could be less than 5 MB, adjust the part size as
                needed.
                partSize = Math.min(partSize, (contentLength - filePosition));

                // Create the request to upload a part.
                UploadPartRequest uploadRequest = new UploadPartRequest()
                    .withBucketName(bucketName)
                    .withKey(keyName)
                    .withUploadId(initResponse.getUploadId())
                    .withPartNumber(i)
                    .withFileOffset(filePosition)
                    .withFile(file)
                    .withPartSize(partSize);

                // Upload the part and add the response's ETag to our list.
                UploadPartResult uploadResult = s3Client.uploadPart(uploadRequest);
                partETags.add(uploadResult.getPartETag());

                filePosition += partSize;
            }

            // Complete the multipart upload.
            CompleteMultipartUploadRequest compRequest = new
            CompleteMultipartUploadRequest(bucketName, keyName,
                initResponse.getUploadId(), partETags);
            s3Client.completeMultipartUpload(compRequest);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

List Multipart Uploads

Example

The following example shows how to retrieve a list of in-progress multipart uploads using the low-level Java API:

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListMultipartUploadsRequest;
import com.amazonaws.services.s3.model.MultipartUpload;
import com.amazonaws.services.s3.model.MultipartUploadListing;

import java.util.List;

public class ListMultipartUploads {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Retrieve a list of all in-progress multipart uploads.
            ListMultipartUploadsRequest allMultipartUploadsRequest = new
            ListMultipartUploadsRequest(bucketName);
            MultipartUploadListing multipartUploadListing =
            s3Client.listMultipartUploads(allMultipartUploadsRequest);
            List<MultipartUpload> uploads = multipartUploadListing.getMultipartUploads();

            // Display information about all in-progress multipart uploads.
            System.out.println(uploads.size() + " multipart upload(s) in progress.");
            for (MultipartUpload u : uploads) {
                System.out.println("Upload in progress: Key = \"" + u.getKey() + "\", id =
                " + u.getUploadId());
            }
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Abort a Multipart Upload

You can abort an in-progress multipart upload by calling the `AmazonS3Client.abortMultipartUpload()` method. This method deletes all parts that were uploaded to Amazon S3 and frees the resources. You provide the upload ID, bucket name, and key name.

Example

The following example shows how to abort multipart uploads using the low-level Java API.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AbortMultipartUploadRequest;
import com.amazonaws.services.s3.model.ListMultipartUploadsRequest;
import com.amazonaws.services.s3.model.MultipartUpload;
import com.amazonaws.services.s3.model.MultipartUploadListing;

import java.util.List;

public class LowLevelAbortMultipartUpload {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Find all in-progress multipart uploads.
            ListMultipartUploadsRequest allMultipartUploadsRequest = new
            ListMultipartUploadsRequest(bucketName);
            MultipartUploadListing multipartUploadListing =
            s3Client.listMultipartUploads(allMultipartUploadsRequest);

            List<MultipartUpload> uploads = multipartUploadListing.getMultipartUploads();
            System.out.println("Before deletions, " + uploads.size() + " multipart uploads
            in progress.");

            // Abort each upload.
            for (MultipartUpload u : uploads) {
                System.out.println("Upload in progress: Key = \"" + u.getKey() + "\", id =
                " + u.getUploadId());
                s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(bucketName,
                u.getKey(), u.getUploadId()));
                System.out.println("Upload deleted: Key = \"" + u.getKey() + "\", id = " +
                u.getUploadId());
            }

            // Verify that all in-progress multipart uploads have been aborted.
            multipartUploadListing =
            s3Client.listMultipartUploads(allMultipartUploadsRequest);
            uploads = multipartUploadListing.getMultipartUploads();
            System.out.println("After aborting uploads, " + uploads.size() + " multipart
            uploads in progress.");
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

Note

Instead of aborting multipart uploads individually, you can abort all of your in-progress multipart uploads that were initiated before a specific time. This clean-up operation is useful for aborting multipart uploads that you initiated but that didn't complete or were aborted. For more information, see [Abort Multipart Uploads \(p. 183\)](#).

Using the AWS SDK for .NET for Multipart Upload (High-Level API)

Topics

- [Upload a File to an S3 Bucket Using the AWS SDK for .NET \(High-Level API\) \(p. 191\)](#)
- [Upload a Directory \(p. 192\)](#)
- [Abort Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET \(High-Level API\) \(p. 194\)](#)
- [Track the Progress of a Multipart Upload to an S3 Bucket Using the AWS SDK for .NET \(High-level API\) \(p. 195\)](#)

The AWS SDK for .NET exposes a high-level API that simplifies multipart uploads (see [Uploading Objects Using Multipart Upload API \(p. 175\)](#)). You can upload data from a file, a directory, or a stream. For more information about Amazon S3 multipart uploads, see [Multipart Upload Overview \(p. 175\)](#).

The `TransferUtility` class provides a methods for uploading files and directories, tracking upload progress, and aborting multipart uploads.

Upload a File to an S3 Bucket Using the AWS SDK for .NET (High-Level API)

To upload a file to an S3 bucket, use the `TransferUtility` class. When uploading data from a file, you must provide the object's key name. If you don't, the API uses the file name for the key name. When uploading data from a stream, you must provide the object's key name.

To set advanced upload options—such as the part size, the number of threads when uploading the parts concurrently, metadata, the storage class, or ACL—use the `TransferUtilityUploadRequest` class.

The following C# example uploads a file to an Amazon S3 bucket in multiple parts. It shows how to use various `TransferUtility.Upload` overloads to upload a file. Each successive call to upload replaces the previous upload. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadFileMPUHighLevelAPITest
    {
        private const string bucketName = "**** provide bucket name ****";
        private const string keyName = "**** provide a name for the uploaded object ****";
        private const string filePath = "**** provide the full path name of the file to
upload ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            UploadFileAsync().Wait();
        }

        private static async Task UploadFileAsync()
        {
            try
```

```
{
    var fileTransferUtility =
        new TransferUtility(s3Client);

    // Option 1. Upload a file. The file name is used as the object key name.
    await fileTransferUtility.UploadAsync(filePath, bucketName);
    Console.WriteLine("Upload 1 completed");

    // Option 2. Specify object key name explicitly.
    await fileTransferUtility.UploadAsync(filePath, bucketName, keyName);
    Console.WriteLine("Upload 2 completed");

    // Option 3. Upload data from a type of System.IO.Stream.
    using (var fileToUpload =
        new FileStream(filePath, FileMode.Open, FileAccess.Read))
    {
        await fileTransferUtility.UploadAsync(fileToUpload,
            bucketName, keyName);
    }
    Console.WriteLine("Upload 3 completed");

    // Option 4. Specify advanced settings.
    var fileTransferUtilityRequest = new TransferUtilityUploadRequest
    {
        BucketName = bucketName,
        FilePath = filePath,
        StorageClass = S3StorageClass.StandardInfrequentAccess,
        PartSize = 6291456, // 6 MB.
        Key = keyName,
        CannedACL = S3CannedACL.PublicRead
    };
    fileTransferUtilityRequest.Metadata.Add("param1", "Value1");
    fileTransferUtilityRequest.Metadata.Add("param2", "Value2");

    await fileTransferUtility.UploadAsync(fileTransferUtilityRequest);
    Console.WriteLine("Upload 4 completed");
}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
}
}
```

More Info

[AWS SDK for .NET](#)

[Upload a Directory](#)

You can use the `TransferUtility` class to upload an entire directory. By default, the API uploads only the files at the root of the specified directory. You can, however, specify recursively uploading files in all of the subdirectories.

To select files in the specified directory based on filtering criteria, specify filtering expressions. For example, to upload only the .pdf files from a directory, specify the `"*.pdf"` filter expression.

When uploading files from a directory, you don't specify the key names for the resulting objects. Amazon S3 constructs the key names using the original file path. For example, assume that you have a directory called `c:\myfolder` with the following structure:

Example

```
C:\myfolder
  \a.txt
  \b.pdf
  \media\
    An.mp3
```

When you upload this directory, Amazon S3 uses the following key names:

Example

```
a.txt
b.pdf
media/An.mp3
```

Example

The following C# example uploads a directory to an Amazon S3 bucket. It shows how to use various `TransferUtility.UploadDirectory` overloads to upload the directory. Each successive call to upload replaces the previous upload. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadDirMPUHighLevelAPITest
    {
        private const string existingBucketName = "**** bucket name ****";
        private const string directoryPath = @"**** directory path ****";
        // The example uploads only .txt files.
        private const string wildCard = "*.txt";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            UploadDirAsync().Wait();
        }

        private static async Task UploadDirAsync()
        {
            try
            {
                var directoryTransferUtility =
                    new TransferUtility(s3Client);

                // 1. Upload a directory.
                await directoryTransferUtility.UploadDirectoryAsync(directoryPath,
                    existingBucketName);
            }
        }
    }
}
```

```
        Console.WriteLine("Upload statement 1 completed");

        // 2. Upload only the .txt files from a directory
        //    and search recursively.
        await directoryTransferUtility.UploadDirectoryAsync(
            directoryPath,
            existingBucketName,
            wildCard,
            SearchOption.AllDirectories);
        Console.WriteLine("Upload statement 2 completed");

        // 3. The same as Step 2 and some optional configuration.
        //    Search recursively for .txt files to upload.
        var request = new TransferUtilityUploadDirectoryRequest
        {
            BucketName = existingBucketName,
            Directory = directoryPath,
            SearchOption = SearchOption.AllDirectories,
            SearchPattern = wildCard
        };

        await directoryTransferUtility.UploadDirectoryAsync(request);
        Console.WriteLine("Upload statement 3 completed");
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine(
            "Error encountered ***. Message:'{0}' when writing an object",
            e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine(
            "Unknown encountered on server. Message:'{0}' when writing an object",
            e.Message);
    }
}
}
```

Abort Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET (High-Level API)

To abort in-progress multipart uploads, use the `TransferUtility` class from the AWS SDK for .NET. You provide a `DateTimeValue`. The API then aborts all of the multipart uploads that were initiated before the specified date and time and remove the uploaded parts. An upload is considered to be in-progress after you initiate it and it completes or you abort it.

Because you are billed for all storage associated with uploaded parts, it's important that you either complete the multipart upload to finish creating the object or abort it to remove uploaded parts. For more information about Amazon S3 multipart uploads, see [Multipart Upload Overview \(p. 175\)](#). For information about pricing, see [Multipart Upload and Pricing \(p. 177\)](#).

The following C# example aborts all in-progress multipart uploads that were initiated on a specific bucket over a week ago. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions on creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.Threading.Tasks;
```

```
namespace Amazon.DocSamples.S3
{
    class AbortMPUUsingHighLevelAPITest
    {
        private const string bucketName = "**** provide bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            AbortMPUAsync().Wait();
        }

        private static async Task AbortMPUAsync()
        {
            try
            {
                var transferUtility = new TransferUtility(s3Client);

                // Abort all in-progress uploads initiated before the specified date.
                await transferUtility.AbortMultipartUploadsAsync(
                    bucketName, DateTime.Now.AddDays(-7));
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when writing an object", e.Message);
            }
        }
    }
}
```

Note

You can also abort a specific multipart upload. For more information, see [List Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET \(Low-level\)](#) (p. 199).

More Info

[AWS SDK for .NET](#)

[Track the Progress of a Multipart Upload to an S3 Bucket Using the AWS SDK for .NET \(High-level API\)](#)

The following C# example uploads a file to an S3 bucket using the `TransferUtility` class, and tracks the progress of the upload. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples](#) (p. 678).

```
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{

```

```

class TrackMPUUsingHighLevelAPITest
{
    private const string bucketName = "**** provide the bucket name ****";
    private const string keyName = "**** provide the name for the uploaded object ****";
    private const string filePath = " **** provide the full path name of the file to
upload ****";
    // Specify your bucket region (an example region is shown).
    private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
    private static IAmazonS3 s3Client;

    public static void Main()
    {
        s3Client = new AmazonS3Client(bucketRegion);
        TrackMPUAsync().Wait();
    }

    private static async Task TrackMPUAsync()
    {
        try
        {
            var fileTransferUtility = new TransferUtility(s3Client);

            // Use TransferUtilityUploadRequest to configure options.
            // In this example we subscribe to an event.
            var uploadRequest =
                new TransferUtilityUploadRequest
                {
                    BucketName = bucketName,
                    FilePath = filePath,
                    Key = keyName
                };

            uploadRequest.UploadProgressEvent +=
                new EventHandler<UploadProgressArgs>
                (uploadRequest_UploadPartProgressEvent);

            await fileTransferUtility.UploadAsync(uploadRequest);
            Console.WriteLine("Upload completed");
        }
        catch (AmazonS3Exception e)
        {
            Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
        }
    }

    static void uploadRequest_UploadPartProgressEvent(object sender, UploadProgressArgs
e)
    {
        // Process event.
        Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
    }
}

```

[More Info](#)

[AWS SDK for .NET](#)

Using the AWS SDK for .NET for Multipart Upload (Low-Level API)

The AWS SDK for .NET exposes a low-level API that closely resembles the Amazon S3 REST API for multipart upload (see [Using the REST API for Multipart Upload \(p. 206\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or when you do not know the size of the data in advance. Use the high-level API (see [Using the AWS SDK for .NET for Multipart Upload \(High-Level API\) \(p. 191\)](#)), whenever you don't have these requirements.

Topics

- [Upload a File to an S3 Bucket Using the AWS SDK for .NET \(Low-Level API\) \(p. 197\)](#)
- [List Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET \(Low-level\) \(p. 199\)](#)
- [Track the Progress of a Multipart Upload to an S3 Bucket Using the AWS SDK for .NET \(Low-Level\) \(p. 199\)](#)
- [Abort Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET \(Low-Level\) \(p. 200\)](#)

Upload a File to an S3 Bucket Using the AWS SDK for .NET (Low-Level API)

The following C# example shows how to use the low-level AWS SDK for .NET multipart upload API to upload a file to an S3 bucket. For information about Amazon S3 multipart uploads, see [Multipart Upload Overview \(p. 175\)](#).

Note

When you use the AWS SDK for .NET API to upload large objects, a timeout might occur while data is being written to the request stream. You can set an explicit timeout using the `UploadPartRequest`.

The following C# example uploads a file to an S3 bucket using the low-level multipart upload API. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadFileMPULowLevelAPITest
    {
        private const string bucketName = "**** provide bucket name ****";
        private const string keyName = "**** provide a name for the uploaded object ****";
        private const string filePath = "**** provide the full path name of the file to
upload ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth1;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            Console.WriteLine("Uploading an object");
            UploadObjectAsync().Wait();
        }
    }
}
```

```
private static async Task UploadObjectAsync()
{
    // Create list to store upload part responses.
    List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

    // Setup information required to initiate the multipart upload.
    InitiateMultipartUploadRequest initiateRequest = new
InitiateMultipartUploadRequest
    {
        BucketName = bucketName,
        Key = keyName
    };

    // Initiate the upload.
    InitiateMultipartUploadResponse initResponse =
        await s3Client.InitiateMultipartUploadAsync(initiateRequest);

    // Upload parts.
    long contentLength = new FileInfo(filePath).Length;
    long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

    try
    {
        Console.WriteLine("Uploading parts");

        long filePosition = 0;
        for (int i = 1; filePosition < contentLength; i++)
        {
            UploadPartRequest uploadRequest = new UploadPartRequest
            {
                BucketName = bucketName,
                Key = keyName,
                UploadId = initResponse.UploadId,
                PartNumber = i,
                PartSize = partSize,
                FilePosition = filePosition,
                FilePath = filePath
            };

            // Track upload progress.
            uploadRequest.StreamTransferProgress +=
                new
EventHandler<StreamTransferProgressArgs>(UploadPartProgressEventCallback);

            // Upload a part and add the response to our list.
            uploadResponses.Add(await s3Client.UploadPartAsync(uploadRequest));

            filePosition += partSize;
        }

        // Setup to complete the upload.
        CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
        {
            BucketName = bucketName,
            Key = keyName,
            UploadId = initResponse.UploadId
        };
        completeRequest.AddPartETags(uploadResponses);

        // Complete the upload.
        CompleteMultipartUploadResponse completeUploadResponse =
            await s3Client.CompleteMultipartUploadAsync(completeRequest);
    }
    catch (Exception exception)
    {
    }
}
```



```
        Console.WriteLine("An AmazonS3Exception was thrown: { 0}",
exception.Message);

        // Abort the upload.
        AbortMultipartUploadRequest abortMPURequest = new
AbortMultipartUploadRequest
        {
            BucketName = bucketName,
            Key = keyName,
            UploadId = initResponse.UploadId
        };
        await s3Client.AbortMultipartUploadAsync(abortMPURequest);
    }
}

public static void UploadPartProgressEventCallback(object sender,
StreamTransferProgressArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
}
```

More Info

[AWS SDK for .NET](#)

List Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET (Low-level)

To list all of the in-progress multipart uploads on a specific bucket, use the AWS SDK for .NET low-level multipart upload API's `ListMultipartUploadsRequest` class. The `AmazonS3Client.ListMultipartUploads` method returns an instance of the `ListMultipartUploadsResponse` class that provides information about the in-progress multipart uploads.

An in-progress multipart upload is a multipart upload that has been initiated using the initiate multipart upload request, but has not yet been completed or aborted. For more information about Amazon S3 multipart uploads, see [Multipart Upload Overview](#) (p. 175).

The following C# example shows how to use the AWS SDK for .NET to list all in-progress multipart uploads on a bucket. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples](#) (p. 678).

```
ListMultipartUploadsRequest request = new ListMultipartUploadsRequest
{
    BucketName = bucketName // Bucket receiving the uploads.
};

ListMultipartUploadsResponse response = await
AmazonS3Client.ListMultipartUploadsAsync(request);
```

More Info

[AWS SDK for .NET](#)

Track the Progress of a Multipart Upload to an S3 Bucket Using the AWS SDK for .NET (Low-Level)

To track the progress of a multipart upload, use the `UploadPartRequest.StreamTransferProgress` event provided by the AWS SDK for .NET low-level multipart upload API. The event occurs periodically. It returns information such as the total number of bytes to transfer and the number of bytes transferred.

The following C# example shows how to track the progress of multipart uploads. For a complete C# sample that includes the following code, see [Upload a File to an S3 Bucket Using the AWS SDK for .NET \(Low-Level API\)](#) (p. 197).

```
UploadPartRequest uploadRequest = new UploadPartRequest
{
    // Provide the request data.
};

uploadRequest.StreamTransferProgress +=
    new EventHandler<StreamTransferProgressArgs>(UploadPartProgressEventCallback);

...
public static void UploadPartProgressEventCallback(object sender,
    StreamTransferProgressArgs e)
{
    // Process the event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
```

More Info

[AWS SDK for .NET](#)

Abort Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET (Low-Level)

You can abort an in-progress multipart upload by calling the `AmazonS3Client.AbortMultipartUploadAsync` method. In addition to aborting the upload, this method deletes all parts that were uploaded to Amazon S3.

To abort a multipart upload, you provide the upload ID, and the bucket and key names that are used in the upload. After you have aborted a multipart upload, you can't use the upload ID to upload additional parts. For more information about Amazon S3 multipart uploads, see [Multipart Upload Overview](#) (p. 175).

The following C# example shows how to abort an multipart upload. For a complete C# sample that includes the following code, see [Upload a File to an S3 Bucket Using the AWS SDK for .NET \(Low-Level API\)](#) (p. 197).

```
AbortMultipartUploadRequest abortMPURequest = new AbortMultipartUploadRequest
{
    BucketName = existingBucketName,
    Key = keyName,
    UploadId = initResponse.UploadId
};
await AmazonS3Client.AbortMultipartUploadAsync(abortMPURequest);
```

You can also abort all in-progress multipart uploads that were initiated prior to a specific time. This clean-up operation is useful for aborting multipart uploads that didn't complete or were aborted. For more information, see [Abort Multipart Uploads to an S3 Bucket Using the AWS SDK for .NET \(High-Level API\)](#) (p. 194).

More Info

[AWS SDK for .NET](#)

Using the AWS PHP SDK for Multipart Upload

You can upload large files to Amazon S3 in multiple parts. You must use a multipart upload for files larger than 5 GB. The AWS SDK for PHP exposes the [MultipartUploader](#) class that simplifies multipart uploads.

The `upload` method of the `MultipartUploader` class is best used for a simple multipart upload. If you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the data in advance, use the low-level PHP API. For more information, see [Using the AWS PHP SDK for Multipart Upload \(Low-Level API\)](#) (p. 203).

For more information about multipart uploads, see [Uploading Objects Using Multipart Upload API](#) (p. 175). For information on uploading files that are less than 5GB in size, see [Upload an Object Using the AWS SDK for PHP](#) (p. 173).

Upload a File Using the High-Level Multipart Upload

This topic explains how to use the high-level `Aws\S3\Model\MultipartUpload\UploadBuilder` class from the AWS SDK for PHP for multipart file uploads. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples](#) (p. 678) and have the AWS SDK for PHP properly installed.

The following PHP example uploads a file to an Amazon S3 bucket. The example demonstrates how to set parameters for the `MultipartUploader` object.

For information about running the PHP examples in this guide, see [Running PHP Examples](#) (p. 679).

```
require 'vendor/autoload.php';

use Aws\Common\Exception\MultipartUploadException;
use Aws\S3\MultipartUploader;
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Prepare the upload parameters.
$uploader = new MultipartUploader($s3, '/path/to/large/file.zip', [
    'bucket' => $bucket,
    'key'    => $keyname
]);

// Perform the upload.
try {
    $result = $uploader->upload();
    echo "Upload complete: {$result['ObjectURL']}" . PHP_EOL;
} catch (MultipartUploadException $e) {
    echo $e->getMessage() . PHP_EOL;
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 `Aws\S3\S3Client` Class](#)
- [Amazon S3 Multipart Uploads](#)
- [AWS SDK for PHP Documentation](#)

Using the AWS PHP SDK for Multipart Upload (Low-Level API)

Topics

- [Upload a File in Multiple Parts Using the PHP SDK Low-Level API \(p. 203\)](#)
- [List Multipart Uploads Using the Low-Level AWS SDK for PHP API \(p. 204\)](#)
- [Abort a Multipart Upload \(p. 205\)](#)

The AWS SDK for PHP exposes a low-level API that closely resembles the Amazon S3 REST API for multipart upload (see [Using the REST API for Multipart Upload \(p. 206\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or if you do not know the size of the data in advance. Use the AWS SDK for PHP high-level abstractions (see [Using the AWS PHP SDK for Multipart Upload \(p. 201\)](#)) whenever you don't have these requirements.

Upload a File in Multiple Parts Using the PHP SDK Low-Level API

This topic guides shows how to use the low-level `uploadPart` method from version 3 of the AWS SDK for PHP to upload a file in multiple parts. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example uploads a file to an Amazon S3 bucket using the low-level PHP API multipart upload. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filename = '*** Path to and Name of the File to Upload ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

$result = $s3->createMultipartUpload([
    'Bucket' => $bucket,
    'Key' => $keyname,
    'StorageClass' => 'REDUCED_REDUNDANCY',
    'ACL' => 'public-read',
    'Metadata' => [
        'param1' => 'value 1',
        'param2' => 'value 2',
        'param3' => 'value 3'
    ]
]);

$uploadId = $result['UploadId'];

// Upload the file in parts.
try {
    $file = fopen($filename, 'r');
    $partNumber = 1;
    while (!feof($file)) {
        $result = $s3->uploadPart([
            'Bucket' => $bucket,
            'Key' => $keyname,
            'UploadId' => $uploadId,
            'PartNumber' => $partNumber,
```

```
        'Body'          => fread($file, 5 * 1024 * 1024),
    ]);
    $parts['Parts'][$partNumber] = [
        'PartNumber' => $partNumber,
        'ETag' => $result['ETag'],
    ];
    $partNumber++;

    echo "Uploading part {$partNumber} of {$filename}." . PHP_EOL;
}
fclose($file);
} catch (S3Exception $e) {
    $result = $s3->abortMultipartUpload([
        'Bucket' => $bucket,
        'Key' => $keyname,
        'UploadId' => $uploadId
    ]);

    echo "Upload of {$filename} failed." . PHP_EOL;
}

// Complete the multipart upload.
$result = $s3->completeMultipartUpload([
    'Bucket' => $bucket,
    'Key' => $keyname,
    'UploadId' => $uploadId,
    'MultipartUpload' => $parts,
]);
$url = $result['Location'];

echo "Uploaded {$filename} to {$url}." . PHP_EOL;
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [Amazon S3 Multipart Uploads](#)
- [AWS SDK for PHP Documentation](#)

List Multipart Uploads Using the Low-Level AWS SDK for PHP API

This topic shows how to use the low-level API classes from version 3 of the AWS SDK for PHP to list all in-progress multipart uploads on a bucket. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example demonstrates listing all in-progress multipart uploads on a bucket.

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

// Retrieve a list of the current multipart uploads.
$result = $s3->listMultipartUploads([
    'Bucket' => $bucket
```

```
]);  
  
// Write the list of uploads to the page.  
print_r($result->toArray());
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [Amazon S3 Multipart Uploads](#)
- [AWS SDK for PHP Documentation](#)

Abort a Multipart Upload

This topic describes how to use a class from version 3 of the AWS SDK for PHP to abort a multipart upload that is in progress. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example shows how to abort an in-progress multipart upload using the `abortMultipartUpload()` method. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

```
require 'vendor/autoload.php';  
  
use Aws\S3\S3Client;  
  
$bucket = '*** Your Bucket Name ***';  
$keyname = '*** Your Object Key ***';  
$uploadId = '*** Upload ID of upload to Abort ***';  
  
$s3 = new S3Client([  
    'version' => 'latest',  
    'region'  => 'us-east-1'  
]);  
  
// Abort the multipart upload.  
$s3->abortMultipartUpload([  
    'Bucket'    => $bucket,  
    'Key'       => $keyname,  
    'UploadId' => $uploadId,  
]);
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [Amazon S3 Multipart Uploads](#)
- [AWS SDK for PHP Documentation](#)

Using the AWS SDK for Ruby for Multipart Upload

The AWS SDK for Ruby version 3 supports Amazon S3 multipart uploads in two ways. For the first option, you can use a managed file upload helper. This is the recommended method for uploading files to a bucket and it provides the following benefits:

- Manages multipart uploads for objects larger than 15MB.
- Correctly opens files in binary mode to avoid encoding issues.
- Uses multiple threads for uploading parts of large objects in parallel.

For more information, see [Uploading Files to Amazon S3](#) in the AWS Developer Blog.

Alternatively, you can use the following multipart upload client operations directly:

- [create_multipart_upload](#) – Initiates a multipart upload and returns an upload ID.
- [upload_part](#) – Uploads a part in a multipart upload.
- [upload_part_copy](#) – Uploads a part by copying data from an existing object as data source.
- [complete_multipart_upload](#) – Completes a multipart upload by assembling previously uploaded parts.
- [abort_multipart_upload](#) – Aborts a multipart upload.

For more information, see [Using the AWS SDK for Ruby - Version 3 \(p. 679\)](#).

Using the REST API for Multipart Upload

The following sections in the Amazon Simple Storage Service API Reference describe the REST API for multipart upload.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

You can use these APIs to make your own REST requests, or you can use one the SDKs we provide. For more information about the SDKs, see [API Support for Multipart Upload \(p. 179\)](#).

Uploading Objects Using Presigned URLs

Topics

- [Upload an Object Using a Presigned URL \(AWS SDK for Java\) \(p. 207\)](#)
- [Upload an Object to an S3 Bucket Using a Presigned URL \(AWS SDK for .NET\) \(p. 208\)](#)
- [Upload an Object Using a Presigned URL \(AWS SDK for Ruby\) \(p. 209\)](#)

A presigned URL gives you access to the object identified in the URL, provided that the creator of the presigned URL has permissions to access that object. That is, if you receive a presigned URL to upload an object, you can upload the object only if the creator of the presigned URL has the necessary permissions to upload that object.

All objects and buckets by default are private. The presigned URLs are useful if you want your user/customer to be able to upload a specific object to your bucket, but you don't require them to have AWS security credentials or permissions. When you create a presigned URL, you must provide your security credentials and then specify a bucket name, an object key, an HTTP method (PUT for uploading objects), and an expiration date and time. The presigned URLs are valid only for the specified duration.

You can generate a presigned URL programmatically using the AWS SDK for Java or the AWS SDK for .NET. If you are using Microsoft Visual Studio, you can also use AWS Explorer to generate a presigned object URL without writing any code. Anyone who receives a valid presigned URL can then programmatically upload an object.

For more information, go to [Using Amazon S3 from AWS Explorer](#).

For instructions about how to install AWS Explorer, see [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#).

Note

Anyone with valid security credentials can create a presigned URL. However, in order for you to successfully upload an object, the presigned URL must be created by someone who has permission to perform the operation that the presigned URL is based upon.

Upload an Object Using a Presigned URL (AWS SDK for Java)

You can use the AWS SDK for Java to generate a presigned URL that you, or anyone you give the URL, can use to upload an object to Amazon S3. When you use the URL to upload an object, Amazon S3 creates the object in the specified bucket. If an object with the same key that is specified in the presigned URL already exists in the bucket, Amazon S3 replaces the existing object with the uploaded object. To successfully complete an upload, you must do the following:

- Specify the HTTP PUT verb when creating the `GeneratePresignedUrlRequest` and `HttpURLConnection` objects.
- Interact with the `HttpURLConnection` object in some way after finishing the upload. The following example accomplishes this by using the `HttpURLConnection` object to check the HTTP response code.

Example

This example generates a presigned URL and uses it to upload sample data as an object. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.S3Object;

import java.io.IOException;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;

public class GeneratePresignedUrlAndUploadObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String objectKey = "*** Object key ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Set the pre-signed URL to expire after one hour.
            java.util.Date expiration = new java.util.Date();
            long expTimeMillis = expiration.getTime();
            expTimeMillis += 1000 * 60 * 60;
            expiration.setTime(expTimeMillis);
```

```

        // Generate the pre-signed URL.
        System.out.println("Generating pre-signed URL.");
        GeneratePresignedUrlRequest generatePresignedUrlRequest = new
GeneratePresignedUrlRequest(bucketName, objectKey)
            .withMethod(HttpMethod.PUT)
            .withExpiration(expiration);
        URL url = s3Client.generatePresignedUrl(generatePresignedUrlRequest);

        // Create the connection and use it to upload the new object using the pre-
signed URL.
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setDoOutput(true);
        connection.setRequestMethod("PUT");
        OutputStreamWriter out = new OutputStreamWriter(connection.getOutputStream());
        out.write("This text uploaded as an object via presigned URL.");
        out.close();

        // Check the HTTP response code. To complete the upload and make the object
available,
        // you must interact with the connection object in some way.
        connection.getResponseCode();
        System.out.println("HTTP response code: " + connection.getResponseCode());

        // Check to make sure that the object was uploaded successfully.
        S3Object object = s3Client.getObject(bucketName, objectKey);
        System.out.println("Object " + object.getKey() + " created in bucket " +
object.getBucketName());
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
}

```

Upload an Object to an S3 Bucket Using a Presigned URL (AWS SDK for .NET)

The following C# example shows how to use the AWS SDK for .NET to upload an object to an S3 bucket using a presigned URL. For more information about presigned URLs, see [Uploading Objects Using Presigned URLs \(p. 206\)](#).

This example generates a presigned URL for a specific object and uses it to upload a file. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.IO;
using System.Net;

namespace Amazon.DocSamples.S3
{
    class UploadObjectUsingPresignedURLTest
    {
        private const string bucketName = "**** provide bucket name ****";
        private const string objectKey = "**** provide the name for the uploaded object
****";
    }
}

```

```

private const string filePath = "*** provide the full path name of the file to
upload ***";
// Specify your bucket region (an example region is shown).
private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
private static IAmazonS3 s3Client;

public static void Main()
{
    s3Client = new AmazonS3Client(bucketRegion);
    var url = GeneratePreSignedURL();
    UploadObject(url);
}

private static void UploadObject(string url)
{
    HttpWebRequest httpRequest = WebRequest.Create(url) as HttpWebRequest;
    httpRequest.Method = "PUT";
    using (Stream dataStream = httpRequest.GetRequestStream())
    {
        var buffer = new byte[8000];
        using (FileStream fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
        {
            int bytesRead = 0;
            while ((bytesRead = fileStream.Read(buffer, 0, buffer.Length)) > 0)
            {
                dataStream.Write(buffer, 0, bytesRead);
            }
        }
    }
    HttpWebResponse response = httpRequest.GetResponse() as HttpWebResponse;
}

private static string GeneratePreSignedURL()
{
    var request = new GetPreSignedUrlRequest
    {
        BucketName = bucketName,
        Key = objectKey,
        Verb = HttpVerb.PUT,
        Expires = DateTime.Now.AddMinutes(5)
    };

    string url = s3Client.GetPreSignedURL(request);
    return url;
}
}

```

More Info

[AWS SDK for .NET](#)

Upload an Object Using a Presigned URL (AWS SDK for Ruby)

The following tasks guide you through using a Ruby script to upload an object using a presigned URL for SDK for Ruby - Version 3.

Uploading Objects - SDK for Ruby - Version 3

1	Create an instance of the <code>Aws::S3::Resource</code> class.
2	Provide a bucket name and an object key by calling the <code>#bucket[]</code> and the <code>#object[]</code> methods of your <code>Aws::S3::Resource</code> class instance.

	Generate a presigned URL by creating an instance of the <code>URI</code> class, and use it to parse the <code>.presigned_url</code> method of your <code>Aws::S3::Resource</code> class instance. You must specify <code>:put</code> as an argument to <code>.presigned_url</code> , and you must specify <code>PUT</code> to <code>Net::HTTP::Session#send_request</code> if you want to upload an object.
3	Anyone with the presigned URL can upload an object. The upload creates an object or replaces any existing object with the same key that is specified in the presigned URL.

The following Ruby code example demonstrates the preceding tasks for SDK for Ruby - Version 3.

Example

```
#Uploading an object using a presigned URL for SDK for Ruby - Version 3.

require 'aws-sdk-s3'
require 'net/http'

s3 = Aws::S3::Resource.new(region:'us-west-2')

obj = s3.bucket('BucketName').object('KeyName')
# Replace BucketName with the name of your bucket.
# Replace KeyName with the name of the object you are creating or replacing.

url = URI.parse(obj.presigned_url(:put))

body = "Hello World!"
# This is the contents of your object. In this case, it's a simple string.

Net::HTTP.start(url.host) do |http|
  http.send_request("PUT", url.request_uri, body, {
    # This is required, or Net::HTTP will add a default unsigned content-type.
    "content-type" => "",
  })
end

puts obj.get.body.read
# This will print out the contents of your object to the terminal window.
```

Copying Objects

The copy operation creates a copy of an object that is already stored in Amazon S3. You can create a copy of your object up to 5 GB in a single atomic operation. However, for copying an object that is greater than 5 GB, you must use the multipart upload API. Using the copy operation, you can:

- Create additional copies of objects
- Rename objects by copying them and deleting the original ones
- Move objects across Amazon S3 locations (e.g., us-west-1 and EU)
- Change object metadata

Each Amazon S3 object has metadata. It is a set of name-value pairs. You can set object metadata at the time you upload it. After you upload the object, you cannot modify object metadata. The only way to modify object metadata is to make a copy of the object and set the metadata. In the copy operation you set the same object as the source and target.

Each object has metadata. Some of it is system metadata and other user-defined. Users control some of the system metadata such as storage class configuration to use for the object, and configure server-side

encryption. When you copy an object, user-controlled system metadata and user-defined metadata are also copied. Amazon S3 resets the system-controlled metadata. For example, when you copy an object, Amazon S3 resets the creation date of the copied object. You don't need to set any of these values in your copy request.

When copying an object, you might decide to update some of the metadata values. For example, if your source object is configured to use standard storage, you might choose to use reduced redundancy storage for the object copy. You might also decide to alter some of the user-defined metadata values present on the source object. Note that if you choose to update any of the object's user-configurable metadata (system or user-defined) during the copy, then you must explicitly specify all of the user-configurable metadata present on the source object in your request, even if you are only changing only one of the metadata values.

For more information about the object metadata, see [Object Key and Metadata](#) (p. 99).

Note

- Copying objects across locations incurs bandwidth charges.
- If the source object is archived in `GLACIER` or `DEEP_ARCHIVE`, you must first restore a temporary copy before you can copy the object to another bucket. For information about archiving objects, see [Transitioning to the GLACIER and DEEP ARCHIVE Storage Classes \(Object Archival\)](#) (p. 123).

When copying objects, you can request Amazon S3 to save the target object encrypted using an AWS Key Management Service (AWS KMS) encryption key, an Amazon S3-managed encryption key, or a customer-provided encryption key. Accordingly, you must specify encryption information in your request. If the copy source is an object that is stored in Amazon S3 using server-side encryption with customer provided key, you will need to provide encryption information in your request so Amazon S3 can decrypt the object for copying. For more information, see [Protecting Data Using Encryption](#) (p. 264).

To copy more than one Amazon S3 object with a single request, you can use Amazon S3 batch operations. You provide Amazon S3 batch operations with a list of objects to operate on. Amazon S3 batch operations call the respective API to perform the specified operation. A single Amazon S3 batch operations job can perform the specified operation on billions of objects containing exabytes of data.

Amazon S3 batch operations track progress, send notifications, and store a detailed completion report of all actions, providing a fully managed, auditable, serverless experience. You can use Amazon S3 batch operations through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see [the section called "The Basics: Jobs"](#) (p. 468).

Related Resources

- [Using the AWS SDKs, CLI, and Explorers](#) (p. 669)

Copying Objects in a Single Operation

The examples in this section show how to copy objects up to 5 GB in a single operation. For copying objects greater than 5 GB, you must use multipart upload API. For more information, see [Copying Objects Using the Multipart Upload API](#) (p. 217).

Topics

- [Copy an Object Using the AWS SDK for Java](#) (p. 212)
- [Copy an Amazon S3 Object in a Single Operation Using the AWS SDK for .NET](#) (p. 212)
- [Copy an Object Using the AWS SDK for PHP](#) (p. 213)
- [Copy an Object Using the AWS SDK for Ruby](#) (p. 215)

- [Copy an Object Using the REST API \(p. 215\)](#)

Copy an Object Using the AWS SDK for Java

Example

The following example shows how to copy an object in Amazon S3 using the AWS SDK for Java. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CopyObjectRequest;

import java.io.IOException;

public class CopyObjectSingleOperation {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String sourceKey = "**** Source object key *** ";
        String destinationKey = "**** Destination object key ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Copy the object into a new object in the same bucket.
            CopyObjectRequest copyObjRequest = new CopyObjectRequest(bucketName, sourceKey,
            bucketName, destinationKey);
            s3Client.copyObject(copyObjRequest);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Copy an Amazon S3 Object in a Single Operation Using the AWS SDK for .NET

The following C# example shows how to use the high-level AWS SDK for .NET to copy objects that are as big as 5 GB in a single operation. For objects that are bigger than 5 GB, use the multipart upload copy example described in [Copy an Amazon S3 Object Using the AWS SDK for .NET Multipart Upload API \(p. 219\)](#).

This example makes a copy of an object that is a maximum of 5 GB. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CopyObjectTest
    {
        private const string sourceBucket = "*** provide the name of the bucket with source object ***";
        private const string destinationBucket = "*** provide the name of the bucket to copy the object to ***";
        private const string objectKey = "*** provide the name of object to copy ***";
        private const string destObjectKey = "*** provide the destination object key name ***";

        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            Console.WriteLine("Copying an object");
            CopyingObjectAsync().Wait();
        }

        private static async Task CopyingObjectAsync()
        {
            try
            {
                CopyObjectRequest request = new CopyObjectRequest
                {
                    SourceBucket = sourceBucket,
                    SourceKey = objectKey,
                    DestinationBucket = destinationBucket,
                    DestinationKey = destObjectKey
                };
                CopyObjectResponse response = await s3Client.CopyObjectAsync(request);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when writing an object", e.Message);
            }
        }
    }
}
```

More Info

[AWS SDK for .NET](#)

Copy an Object Using the AWS SDK for PHP

This topic guides you through using classes from version 3 of the AWS SDK for PHP to copy a single object and multiple objects within Amazon S3, from one bucket to another or within the same bucket.

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

The following tasks guide you through using PHP SDK classes to copy an object that is already stored in Amazon S3.

The following tasks guide you through using PHP classes to make multiple copies of an object within Amazon S3.

Copying Objects

1	Create an instance of an Amazon S3 client by using the <code>Aws\S3\S3Client</code> class constructor.
2	To make multiple copies of an object, you execute a batch of calls to the Amazon S3 client <code>getCommand()</code> method, which is inherited from the <code>Aws\CommandInterface</code> class. You provide the <code>CopyObject</code> command as the first argument and an array containing the source bucket, source key name, target bucket, and target key name as the second argument.

Example of Copying Objects within Amazon S3

The following PHP example illustrates the use of the `copyObject()` method to copy a single object within Amazon S3 and using a batch of calls to `CopyObject` using the `getCommand()` method to make multiple copies of an object.

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

// Copy an object.
$s3->copyObject([
    'Bucket' => $targetBucket,
    'Key' => "{$sourceKeyname}-copy",
    'CopySource' => "{$sourceBucket}/{$sourceKeyname}",
]);

// Perform a batch of CopyObject operations.
$batch = array();
for ($i = 1; $i <= 3; $i++) {
    $batch[] = $s3->getCommand('CopyObject', [
        'Bucket' => $targetBucket,
        'Key' => "{$targetKeyname}-{$i}",
        'CopySource' => "{$sourceBucket}/{$sourceKeyname}",
    ]);
}
try {
    $results = CommandPool::batch($s3, $batch);
    foreach($results as $result) {
        if ($result instanceof ResultInterface) {
            // Result handling here
        }
        if ($result instanceof AwsException) {

```



```
        // AwsException handling here
    }
}
} catch (\Exception $e) {
    // General error handling here
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Copy an Object Using the AWS SDK for Ruby

The following tasks guide you through using the Ruby classes to copy an object in Amazon S3, from one bucket to another or to copy an object within the same bucket.

Copying Objects

1	Use the Amazon S3 modularized gem for version 3 of the AWS SDK for Ruby, require 'aws-sdk-s3', and provide your AWS credentials. For more information about how to provide your credentials, see Making Requests Using AWS Account or IAM User Credentials (p. 19) .
2	Provide the request information, such as source bucket name, source key name, destination bucket name, and destination key.

The following Ruby code example demonstrates the preceding tasks using the `#copy_object` method to copy an object from one bucket to another.

Example

```
require 'aws-sdk-s3'

source_bucket_name = '*** Provide bucket name ***'
target_bucket_name = '*** Provide bucket name ***'
source_key = '*** Provide source key ***'
target_key = '*** Provide target key ***'

begin
  s3 = Aws::S3::Client.new(region: 'us-west-2')
  s3.copy_object(bucket: target_bucket_name, copy_source: source_bucket_name + '/' +
    source_key, key: target_key)
rescue StandardError => ex
  puts 'Caught exception copying object ' + source_key + ' from bucket ' +
    source_bucket_name + ' to bucket ' + target_bucket_name + ' as ' + target_key + ':'
  puts ex.message
end

puts 'Copied ' + source_key + ' from bucket ' + source_bucket_name + ' to bucket ' +
  target_bucket_name + ' as ' + target_key
```

Copy an Object Using the REST API

This example describes how to copy an object using REST. For more information about the REST API, go to [PUT Object \(Copy\)](#).

This example copies the `flotsam` object from the `pacific` bucket to the `jetsam` object of the `atlantic` bucket, preserving its metadata.

```
PUT /jetsam HTTP/1.1
Host: atlantic.s3.amazonaws.com
x-amz-copy-source: /pacific/flotsam
Authorization: AWS AKIAIOSFODNN7EXAMPLE:ENoSbxYByFA0UGLZUqJN5EUnLDg=
Date: Wed, 20 Feb 2008 22:12:21 +0000
```

The signature was generated from the following information.

```
PUT\r\n
\r\n
\r\n
Wed, 20 Feb 2008 22:12:21 +0000\r\n

x-amz-copy-source:/pacific/flotsam\r\n
/atlantic/jetsam
```

Amazon S3 returns the following response that specifies the ETag of the object and when it was last modified.

```
HTTP/1.1 200 OK
x-amz-id-2: Vyaxt7qEbzv34BnSu5hctyyNSlHTYZFMWK4FtzO+ix8JQNyaLdTshL0Kxatba0Zt
x-amz-request-id: 6B13C3C5B34AF333
Date: Wed, 20 Feb 2008 22:13:01 +0000

Content-Type: application/xml
Transfer-Encoding: chunked
Connection: close
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>

<CopyObjectResult>
  <LastModified>2008-02-20T22:13:01</LastModified>
  <ETag>"7e9c608af58950deeb370c98608ed097"</ETag>
</CopyObjectResult>
```

Copying Objects Using the Multipart Upload API

The examples in this section show you how to copy objects greater than 5 GB using the multipart upload API. You can copy objects less than 5 GB in a single operation. For more information, see [Copying Objects in a Single Operation \(p. 211\)](#).

Topics

- [Copy an Object Using the AWS SDK for Java Multipart Upload API \(p. 217\)](#)
- [Copy an Amazon S3 Object Using the AWS SDK for .NET Multipart Upload API \(p. 219\)](#)
- [Copy Object Using the REST Multipart Upload API \(p. 221\)](#)

Copy an Object Using the AWS SDK for Java Multipart Upload API

To copy an Amazon S3 object that is larger than 5 GB with the AWS SDK for Java, use the low-level Java API. For objects smaller than 5 GB, use the single-operation copy described in [Copy an Object Using the AWS SDK for Java \(p. 212\)](#).

To copy an object using the low-level Java API, do the following:

- Initiate a multipart upload by executing the `AmazonS3Client.initiateMultipartUpload()` method.
- Save the upload ID from the response object that the `AmazonS3Client.initiateMultipartUpload()` method returns. You provide this upload ID for each part-upload operation.
- Copy all of the parts. For each part that you need to copy, create a new instance of the `CopyPartRequest` class. Provide the part information, including the source and destination bucket names, source and destination object keys, upload ID, locations of the first and last bytes of the part, and part number.
- Save the responses of the `AmazonS3Client.copyPart()` method calls. Each response includes the `ETag` value and part number for the uploaded part. You need this information to complete the multipart upload.
- Call the `AmazonS3Client.completeMultipartUpload()` method to complete the copy operation.

Example

The following example shows how to use the Amazon S3 low-level Java API to perform a multipart copy. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class LowLevelMultipartCopy {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
```

```
String sourceBucketName = "**** Source bucket name ****";
String sourceObjectKey = "**** Source object key ****";
String destBucketName = "**** Target bucket name ****";
String destObjectKey = "**** Target object key ****";

try {
    AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(clientRegion)
        .build();

    // Initiate the multipart upload.
    InitiateMultipartUploadRequest initRequest = new
InitiateMultipartUploadRequest(destBucketName, destObjectKey);
    InitiateMultipartUploadResult initResult =
s3Client.initiateMultipartUpload(initRequest);

    // Get the object size to track the end of the copy operation.
    GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);
    ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRequest);
    long objectSize = metadataResult.getContentLength();

    // Copy the object using 5 MB parts.
    long partSize = 5 * 1024 * 1024;
    long bytePosition = 0;
    int partNum = 1;
    List<CopyPartResult> copyResponses = new ArrayList<CopyPartResult>();
    while (bytePosition < objectSize) {
        // The last part might be smaller than partSize, so check to make sure
        // that lastByte isn't beyond the end of the object.
        long lastByte = Math.min(bytePosition + partSize - 1, objectSize - 1);

        // Copy this part.
        CopyPartRequest copyRequest = new CopyPartRequest()
            .withSourceBucketName(sourceBucketName)
            .withSourceKey(sourceObjectKey)
            .withDestinationBucketName(destBucketName)
            .withDestinationKey(destObjectKey)
            .withUploadId(initResult.getUploadId())
            .withFirstByte(bytePosition)
            .withLastByte(lastByte)
            .withPartNumber(partNum++);
        copyResponses.add(s3Client.copyPart(copyRequest));
        bytePosition += partSize;
    }

    // Complete the upload request to concatenate all uploaded parts and make the
    copied object available.
    CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest(
    destBucketName,
    destObjectKey,
    initResult.getUploadId(),
    getETags(copyResponses));
    s3Client.completeMultipartUpload(completeRequest);
    System.out.println("Multipart copy complete.");
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
```

```

    }

    // This is a helper function to construct a list of ETags.
    private static List<PartETag> getETags(List<CopyPartResult> responses) {
        List<PartETag> etags = new ArrayList<PartETag>();
        for (CopyPartResult response : responses) {
            etags.add(new PartETag(response.getPartNumber(), response.getETag()));
        }
        return etags;
    }
}

```

Copy an Amazon S3 Object Using the AWS SDK for .NET Multipart Upload API

The following C# example shows how to use the AWS SDK for .NET to copy an Amazon S3 object that is larger than 5 GB from one source location to another, such as from one bucket to another. To copy objects that are smaller than 5 GB, use the single-operation copy procedure described in [Copy an Amazon S3 Object in a Single Operation Using the AWS SDK for .NET \(p. 212\)](#). For more information about Amazon S3 multipart uploads, see [Multipart Upload Overview \(p. 175\)](#).

This example shows how to copy an Amazon S3 object that is larger than 5 GB from one S3 bucket to another using the AWS SDK for .NET multipart upload API. For information about SDK compatibility and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CopyObjectUsingMPUapiTest
    {
        private const string sourceBucket = "*** provide the name of the bucket with source object ***";
        private const string targetBucket = "*** provide the name of the bucket to copy the object to ***";
        private const string sourceObjectKey = "*** provide the name of object to copy ***";
        private const string targetObjectKey = "*** provide the name of the object copy ***";

        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            {
                s3Client = new AmazonS3Client(bucketRegion);
                Console.WriteLine("Copying an object");
                MPUCopyObjectAsync().Wait();
            }
            private static async Task MPUCopyObjectAsync()
            {
                // Create a list to store the upload part responses.
                List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();
                List<CopyPartResponse> copyResponses = new List<CopyPartResponse>();

                // Setup information required to initiate the multipart upload.
                InitiateMultipartUploadRequest initiateRequest =

```

```
new InitiateMultipartUploadRequest
{
    BucketName = targetBucket,
    Key = targetObjectKey
};

// Initiate the upload.
InitiateMultipartUploadResponse initResponse =
    await s3Client.InitiateMultipartUploadAsync(initiateRequest);

// Save the upload ID.
String uploadId = initResponse.UploadId;

try
{
    // Get the size of the object.
    GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
    {
        BucketName = sourceBucket,
        Key = sourceObjectKey
    };

    GetObjectMetadataResponse metadataResponse =
        await s3Client.GetObjectMetadataAsync(metadataRequest);
    long objectSize = metadataResponse.ContentLength; // Length in bytes.

    // Copy the parts.
    long partSize = 5 * (long)Math.Pow(2, 20); // Part size is 5 MB.

    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++)
    {
        CopyPartRequest copyRequest = new CopyPartRequest
        {
            DestinationBucket = targetBucket,
            DestinationKey = targetObjectKey,
            SourceBucket = sourceBucket,
            SourceKey = sourceObjectKey,
            UploadId = uploadId,
            FirstByte = bytePosition,
            LastByte = bytePosition + partSize - 1 >= objectSize ? objectSize -
1 : bytePosition + partSize - 1,
            PartNumber = i
        };

        copyResponses.Add(await s3Client.CopyPartAsync(copyRequest));

        bytePosition += partSize;
    }

    // Set up to complete the copy.
    CompleteMultipartUploadRequest completeRequest =
    new CompleteMultipartUploadRequest
    {
        BucketName = targetBucket,
        Key = targetObjectKey,
        UploadId = initResponse.UploadId
    };
    completeRequest.AddPartETags(copyResponses);

    // Complete the copy.
    CompleteMultipartUploadResponse completeUploadResponse =
        await s3Client.CompleteMultipartUploadAsync(completeRequest);
}
catch (AmazonS3Exception e)
{
}
```

```
        Console.WriteLine("Error encountered on server. Message:'{0}' when writing  
an object", e.Message);  
    }  
    catch (Exception e)  
    {  
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when  
writing an object", e.Message);  
    }  
    }  
}
```

More Info

[AWS SDK for .NET](#)

Copy Object Using the REST Multipart Upload API

The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API for multipart upload. For copying an existing object you use the Upload Part (Copy) API and specify the source object by adding the `x-amz-copy-source` request header in your request.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

You can use these APIs to make your own REST requests, or you can use one the SDKs we provide. For more information about the SDKs, see [API Support for Multipart Upload \(p. 179\)](#).

Listing Object Keys

Keys can be listed by prefix. By choosing a common prefix for the names of related keys and marking these keys with a special character that delimits hierarchy, you can use the list operation to select and browse keys hierarchically. This is similar to how files are stored in directories within a file system.

Amazon S3 exposes a list operation that lets you enumerate the keys contained in a bucket. Keys are selected for listing by bucket and prefix. For example, consider a bucket named "dictionary" that contains a key for every English word. You might make a call to list all the keys in that bucket that start with the letter "q". List results are always returned in UTF-8 binary order.

Both the SOAP and REST list operations return an XML document that contains the names of matching keys and information about the object identified by each key.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Groups of keys that share a prefix terminated by a special delimiter can be rolled up by that common prefix for the purposes of listing. This enables applications to organize and browse their keys hierarchically, much like how you would organize your files into directories in a file system. For example, to extend the dictionary bucket to contain more than just English words, you might form

keys by prefixing each word with its language and a delimiter, such as "French/logical". Using this naming scheme and the hierarchical listing feature, you could retrieve a list of only French words. You could also browse the top-level list of available languages without having to iterate through all the lexicographically intervening keys.

For more information on this aspect of listing, see [Listing Keys Hierarchically Using a Prefix and Delimiter](#) (p. 222).

List Implementation Efficiency

List performance is not substantially affected by the total number of keys in your bucket, nor by the presence or absence of the prefix, marker, maxkeys, or delimiter arguments.

Iterating Through Multi-Page Results

As buckets can contain a virtually unlimited number of keys, the complete results of a list query can be extremely large. To manage large result sets, the Amazon S3 API supports pagination to split them into multiple responses. Each list keys response returns a page of up to 1,000 keys with an indicator indicating if the response is truncated. You send a series of list keys requests until you have received all the keys. AWS SDK wrapper libraries provide the same pagination.

The following Java and .NET SDK examples show how to use pagination when listing keys in a bucket:

- [Listing Keys Using the AWS SDK for Java](#) (p. 223)
- [Listing Keys Using the AWS SDK for .NET](#) (p. 224)

Related Resources

- [Using the AWS SDKs, CLI, and Explorers](#) (p. 669)

Listing Keys Hierarchically Using a Prefix and Delimiter

The prefix and delimiter parameters limit the kind of results returned by a list operation. The prefix limits the results to only those keys that begin with the specified prefix. The delimiter causes a list operation to roll up all the keys that share a common prefix into a single summary list result.

The purpose of the prefix and delimiter parameters is to help you organize and then browse your keys hierarchically. To do this, first pick a delimiter for your bucket, such as slash (/), that doesn't occur in any of your anticipated key names. Next, construct your key names by concatenating all containing levels of the hierarchy, separating each level with the delimiter.

For example, if you were storing information about cities, you might naturally organize them by continent, then by country, then by province or state. Because these names don't usually contain punctuation, you might select slash (/) as the delimiter. The following examples use a slash (/) delimiter.

- Europe/France/Aquitaine/Bordeaux
- North America/Canada/Quebec/Montreal
- North America/USA/Washington/Bellevue
- North America/USA/Washington/Seattle

If you stored data for every city in the world in this manner, it would become awkward to manage a flat key namespace. By using `Prefix` and `Delimiter` with the list operation, you can use the hierarchy you've created to list your data. For example, to list all the states in USA, set `Delimiter='/'` and `Prefix='North America/USA/'`. To list all the provinces in Canada for which you have data, set `Delimiter='/'` and `Prefix='North America/Canada/'`.

A list request with a delimiter lets you browse your hierarchy at just one level, skipping over and summarizing the (possibly millions of) keys nested at deeper levels. For example, assume you have a bucket (`ExampleBucket`) the following keys.

`sample.jpg`

`photos/2006/January/sample.jpg`

`photos/2006/February/sample2.jpg`

`photos/2006/February/sample3.jpg`

`photos/2006/February/sample4.jpg`

The sample bucket has only the `sample.jpg` object at the root level. To list only the root level objects in the bucket you send a GET request on the bucket with `"/` delimiter character. In response, Amazon S3 returns the `sample.jpg` object key because it does not contain the `"/` delimiter character. All other keys contain the delimiter character. Amazon S3 groups these keys and return a single `CommonPrefixes` element with prefix value `photos/` that is a substring from the beginning of these keys to the first occurrence of the specified delimiter.

Example

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>ExampleBucket</Name>
  <Prefix></Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <Delimiter></Delimiter>
  <IsTruncated>>false</IsTruncated>
  <Contents>
    <Key>sample.jpg</Key>
    <LastModified>2011-07-24T19:39:30.000Z</LastModified>
    <ETag>"d1a7fb5eab1c16cb4f7cf341cf188c3d"</ETag>
    <Size>6</Size>
    <Owner>
      <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>
      <DisplayName>displayname</DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <CommonPrefixes>
    <Prefix>photos/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

Listing Keys Using the AWS SDK for Java

Example

The following example lists the object keys in a bucket. The example uses pagination to retrieve a set of object keys. If there are more keys to return after the first page, Amazon S3 includes a continuation token in the response. The example uses the continuation token in the subsequent request to fetch the next set of object keys.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListObjectsV2Request;
import com.amazonaws.services.s3.model.ListObjectsV2Result;
import com.amazonaws.services.s3.model.S3ObjectSummary;

import java.io.IOException;

public class ListKeys {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            System.out.println("Listing objects");

            // maxKeys is set to 2 to demonstrate the use of
            // ListObjectsV2Result.getNextContinuationToken()
            ListObjectsV2Request req = new
ListObjectsV2Request().withBucketName(bucketName).withMaxKeys(2);
            ListObjectsV2Result result;

            do {
                result = s3Client.listObjectsV2(req);

                for (S3ObjectSummary objectSummary : result.getObjectSummaries()) {
                    System.out.printf(" - %s (size: %d)\n", objectSummary.getKey(),
objectSummary.getSize());
                }
                // If there are more than maxKeys keys in the bucket, get a continuation
token
                // and list the next objects.
                String token = result.getNextContinuationToken();
                System.out.println("Next Continuation Token: " + token);
                req.setContinuationToken(token);
            } while (result.isTruncated());
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Listing Keys Using the AWS SDK for .NET

Example

The following C# example lists the object keys for a bucket. In the example, we use pagination to retrieve a set of object keys. If there are more keys to return, Amazon S3 includes a continuation token in the

response. The code uses the continuation token in the subsequent request to fetch the next set of object keys.

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ListObjectsTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;

        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            ListingObjectsAsync().Wait();
        }

        static async Task ListingObjectsAsync()
        {
            try
            {
                ListObjectsV2Request request = new ListObjectsV2Request
                {
                    BucketName = bucketName,
                    MaxKeys = 10
                };
                ListObjectsV2Response response;
                do
                {
                    response = await client.ListObjectsV2Async(request);

                    // Process the response.
                    foreach (S3Object entry in response.S3Objects)
                    {
                        Console.WriteLine("key = {0} size = {1}",
                            entry.Key, entry.Size);
                    }
                    Console.WriteLine("Next Continuation Token: {0}",
                        response.NextContinuationToken);
                    request.ContinuationToken = response.NextContinuationToken;
                } while (response.IsTruncated);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " +
                    amazonS3Exception.ToString());
                Console.ReadKey();
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
                Console.ReadKey();
            }
        }
    }
}
```

```
}  
}
```

Listing Keys Using the AWS SDK for PHP

This topic guides you through using classes from version 3 of the AWS SDK for PHP to list the object keys contained in an Amazon S3 bucket.

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

To list the object keys contained in a bucket using the AWS SDK for PHP you first must list the objects contained in the bucket and then extract the key from each of the listed objects. When listing objects in a bucket you have the option of using the low-level `Aws\S3\S3Client::listObjects()` method or the high-level `Aws\ResultPaginator` class.

The low-level `listObjects()` method maps to the underlying Amazon S3 REST API. Each `listObjects()` request returns a page of up to 1,000 objects. If you have more than 1,000 objects in the bucket, your response will be truncated and you will need to send another `listObjects()` request to retrieve the next set of 1,000 objects.

You can use the high-level `ListObjects` paginator to make your task of listing the objects contained in a bucket a bit easier. To use the `ListObjects` paginator to create a list of objects you execute the Amazon S3 client `getPaginator()` method that is inherited from `Aws/AwsClientInterface` class with the `ListObjects` command as the first argument and an array to contain the returned objects from the specified bucket as the second argument. When used as a `ListObjects` paginator the `getPaginator()` method returns all the objects contained in the specified bucket. There is no 1,000 object limit, so you don't need to worry if the response is truncated or not.

The following tasks guide you through using the PHP Amazon S3 client methods to list the objects contained in a bucket from which you can list the object keys.

Example of Listing Object Keys

The following PHP example demonstrates how to list the keys from a specified bucket. It shows how to use the high-level `getIterator()` method to list the objects in a bucket and then how to extract the key from each of the objects in the list. It also show how to use the low-level `listObjects()` method to list the objects in a bucket and then how to extract the key from each of the objects in the list returned. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 679\)](#).

```
require 'vendor/autoload.php';  
  
use Aws\S3\S3Client;  
use Aws\S3\Exception\S3Exception;  
  
$bucket = '*** Your Bucket Name ***';  
  
// Instantiate the client.  
$s3 = new S3Client([  
    'version' => 'latest',  
    'region'  => 'us-east-1'  
]);  
  
// Use the high-level iterators (returns ALL of your objects).  
try {  
    $results = $s3->getPaginator('ListObjects', [  
        'Bucket' => $bucket  
    ]);  
}
```

```
        foreach ($results as $result) {
            foreach ($result['Contents'] as $object) {
                echo $object['Key'] . PHP_EOL;
            }
        }
    } catch (S3Exception $e) {
        echo $e->getMessage() . PHP_EOL;
    }

    // Use the plain API (returns ONLY up to 1000 of your objects).
    try {
        $objects = $s3->listObjects([
            'Bucket' => $bucket
        ]);
        foreach ($objects['Contents'] as $object) {
            echo $object['Key'] . PHP_EOL;
        }
    } catch (S3Exception $e) {
        echo $e->getMessage() . PHP_EOL;
    }
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [Paginators](#)
- [AWS SDK for PHP Documentation](#)

Listing Keys Using the REST API

You can use the AWS SDK to list the object keys in a bucket. However, if your application requires it, you can send REST requests directly. You can send a GET request to return some or all of the objects in a bucket or you can use selection criteria to return a subset of the objects in a bucket. For more information, go to [GET Bucket \(List Objects\) Version 2](#).

Deleting Objects

Topics

- [Deleting Objects from a Version-Enabled Bucket \(p. 228\)](#)
- [Deleting Objects from an MFA-Enabled Bucket \(p. 228\)](#)
- [Related Resources \(p. 228\)](#)
- [Deleting One Object Per Request \(p. 228\)](#)
- [Deleting Multiple Objects Per Request \(p. 234\)](#)

You can delete one or more objects directly from Amazon S3. You have the following options when deleting an object:

- **Delete a single object**—Amazon S3 provides the DELETE API that you can use to delete one object in a single HTTP request.
- **Delete multiple objects**—Amazon S3 also provides the Multi-Object Delete API that you can use to delete up to 1000 objects in a single HTTP request.

When deleting objects from a bucket that is not version-enabled, you provide only the object key name, however, when deleting objects from a version-enabled bucket, you can optionally provide version ID of the object to delete a specific version of the object.

Deleting Objects from a Version-Enabled Bucket

If your bucket is version-enabled, then multiple versions of the same object can exist in the bucket. When working with version-enabled buckets, the delete API enables the following options:

- **Specify a non-versioned delete request**—That is, you specify only the object's key, and not the version ID. In this case, Amazon S3 creates a delete marker and returns its version ID in the response. This makes your object disappear from the bucket. For information about object versioning and the delete marker concept, see [Object Versioning \(p. 108\)](#).
- **Specify a versioned delete request**—That is, you specify both the key and also a version ID. In this case the following two outcomes are possible:
 - If the version ID maps to a specific object version, then Amazon S3 deletes the specific version of the object.
 - If the version ID maps to the delete marker of that object, Amazon S3 deletes the delete marker. This makes the object reappear in your bucket.

Deleting Objects from an MFA-Enabled Bucket

When deleting objects from a Multi Factor Authentication (MFA) enabled bucket, note the following:

- If you provide an invalid MFA token, the request always fails.
- If you have an MFA-enabled bucket, and you make a versioned delete request (you provide an object key and version ID), the request will fail if you don't provide a valid MFA token. In addition, when using the Multi-Object Delete API on an MFA-enabled bucket, if any of the deletes is a versioned delete request (that is, you specify object key and version ID), the entire request will fail if you don't provide an MFA token.

On the other hand, in the following cases the request succeeds:

- If you have an MFA-enabled bucket, and you make a non-versioned delete request (you are not deleting a versioned object), and you don't provide an MFA token, the delete succeeds.
- If you have a Multi-Object Delete request specifying only non-versioned objects to delete from an MFA-enabled bucket, and you don't provide an MFA token, the deletions succeed.

For information on MFA delete, see [MFA Delete \(p. 433\)](#).

Related Resources

- [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#)

Deleting One Object Per Request

Topics

- [Deleting an Object Using the AWS SDK for Java \(p. 229\)](#)
- [Deleting an Object Using the AWS SDK for .NET \(p. 231\)](#)
- [Deleting an Object Using the AWS SDK for PHP \(p. 233\)](#)
- [Deleting an Object Using the REST API \(p. 234\)](#)

To delete one object per request, use the `DELETE` API (see [DELETE Object](#)). To learn more about object deletion, see [Deleting Objects \(p. 227\)](#).

You can use either the REST API directly or the wrapper libraries provided by the AWS SDKs that simplify application development.

Deleting an Object Using the AWS SDK for Java

You can delete an object from a bucket. If you have versioning enabled on the bucket, you have the following options:

- Delete a specific object version by specifying a version ID.
- Delete an object without specifying a version ID, in which case S3 adds a delete marker to the object.

For more information about versioning, see [Object Versioning \(p. 108\)](#).

Example Example 1: Deleting an Object (Non-Versioned Bucket)

The following example deletes an object from a bucket. The example assumes that the bucket is not versioning-enabled and the object doesn't have any version IDs. In the delete request, you specify only the object key and not a version ID. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

import java.io.IOException;

public class DeleteObjectNonVersionedBucket {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String keyName = "**** Key name ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            s3Client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Example Example 2: Deleting an Object (Versioned Bucket)

The following example deletes an object from a versioned bucket. The example deletes a specific object version by specifying the object key name and version ID. The example does the following:

1. Adds a sample object to the bucket. Amazon S3 returns the version ID of the newly added object. The example uses this version ID in the delete request.
2. Deletes the object version by specifying both the object key name and a version ID. If there are no other versions of that object, Amazon S3 deletes the object entirely. Otherwise, Amazon S3 only deletes the specified version.

Note

You can get the version IDs of an object by sending a `ListVersions` request.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.DeleteVersionRequest;
import com.amazonaws.services.s3.model.PutObjectResult;

import java.io.IOException;

public class DeleteObjectVersionEnabledBucket {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String keyName = "**** Key name ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Check to ensure that the bucket is versioning-enabled.
            String bucketVersionStatus =
s3Client.getBucketVersioningConfiguration(bucketName).getStatus();
            if (!bucketVersionStatus.equals(BucketVersioningConfiguration.ENABLED)) {
                System.out.printf("Bucket %s is not versioning-enabled.", bucketName);
            } else {
                // Add an object.
                PutObjectResult putResult = s3Client.putObject(bucketName, keyName, "Sample
content for deletion example.");
                System.out.printf("Object %s added to bucket %s\n", keyName, bucketName);

                // Delete the version of the object that we just created.
                System.out.println("Deleting versioned object " + keyName);
                s3Client.deleteVersion(new DeleteVersionRequest(bucketName, keyName,
putResult.getVersionId()));
                System.out.printf("Object %s, version %s deleted\n", keyName,
putResult.getVersionId());
            }
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```



```
}
```

Deleting an Object Using the AWS SDK for .NET

When you delete an object from a non-versioned bucket, the object is removed. If you have versioning enabled on the bucket, you have the following options:

- Delete a specific version of an object by specifying a version ID.
- Delete an object without specifying a version ID. Amazon S3 adds a delete marker. For more information about delete markers, see [Object Versioning \(p. 108\)](#).

The following examples show how to delete an object from both versioned and non-versioned buckets. For more information about versioning, see [Object Versioning \(p. 108\)](#).

Example Deleting an Object from a Non-versioned Bucket

The following C# example deletes an object from a non-versioned bucket. The example assumes that the objects don't have version IDs, so you don't specify version IDs. You specify only the object key. For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DeleteObjectNonVersionedBucketTest
    {
        private const string bucketName = "**** bucket name ****";
        private const string keyName = "**** object key ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            DeleteObjectNonVersionedBucketAsync().Wait();
        }
        private static async Task DeleteObjectNonVersionedBucketAsync()
        {
            try
            {
                var deleteObjectRequest = new DeleteObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                };

                Console.WriteLine("Deleting an object");
                await client.DeleteObjectAsync(deleteObjectRequest);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when writing an object", e.Message);
            }
        }
    }
}
```

```
        catch (Exception e)
        {
            Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
        }
    }
}
```

Example Deleting an Object from a Versioned Bucket

The following C# example deletes an object from a versioned bucket. It deletes a specific version of the object by specifying the object key name and version ID.

The code performs the following tasks:

1. Enables versioning on a bucket that you specify (if versioning is already enabled, this has no effect).
2. Adds a sample object to the bucket. In response, Amazon S3 returns the version ID of the newly added object. The example uses this version ID in the delete request.
3. Deletes the sample object by specifying both the object key name and a version ID.

Note

You can also get the version ID of an object by sending a `ListVersions` request:

```
var listResponse = client.ListVersions(new ListVersionsRequest { BucketName =
    bucketName, Prefix = keyName });
```

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DeleteObjectVersion
    {
        private const string bucketName = "*** versioning-enabled bucket name ***";
        private const string keyName = "*** Object Key Name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            CreateAndDeleteObjectVersionAsync().Wait();
        }

        private static async Task CreateAndDeleteObjectVersionAsync()
        {
            try
            {
                // Add a sample object.
                string versionID = await PutAnObject(keyName);

                // Delete the object by specifying an object key and a version ID.
                DeleteObjectRequest request = new DeleteObjectRequest
```

```

        {
            BucketName = bucketName,
            Key = keyName,
            VersionId = versionID
        };
        Console.WriteLine("Deleting an object");
        await client.DeleteObjectAsync(request);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}

static async Task<string> PutAnObject(string objectKey)
{
    PutObjectRequest request = new PutObjectRequest
    {
        BucketName = bucketName,
        Key = objectKey,
        ContentBody = "This is the content body!"
    };
    PutObjectResponse response = await client.PutObjectAsync(request);
    return response.VersionId;
}
}
}

```

Deleting an Object Using the AWS SDK for PHP

This topic shows how to use classes from version 3 of the AWS SDK for PHP to delete an object from a non-versioned bucket. For information on deleting an object from a versioned bucket, see [Deleting an Object Using the REST API \(p. 234\)](#).

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example deletes an object from a bucket. Because this example shows how to delete objects from non-versioned buckets, it provides only the bucket name and object key (not a version ID) in the delete request. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

```

<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

// Delete an object from the bucket.

```

```
$s3->deleteObject([  
    'Bucket' => $bucket,  
    'Key'     => $keyname  
]);
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Deleting an Object Using the REST API

You can use the AWS SDKs to delete an object. However, if your application requires it, you can send REST requests directly. For more information, go to [DELETE Object](#) in the *Amazon Simple Storage Service API Reference*.

Deleting Multiple Objects Per Request

Topics

- [Deleting Multiple Objects Using the AWS SDK for Java \(p. 234\)](#)
- [Deleting Multiple Objects Using the AWS SDK for .NET \(p. 237\)](#)
- [Deleting Multiple Objects Using the AWS SDK for PHP \(p. 243\)](#)
- [Deleting Multiple Objects Using the REST API \(p. 245\)](#)

Amazon S3 provides the Multi-Object Delete API (see [Delete - Multi-Object Delete](#)), which enables you to delete multiple objects in a single request. The API supports two modes for the response: verbose and quiet. By default, the operation uses verbose mode. In verbose mode, the response includes the result of the deletion of each key that is specified in your request. In quiet mode, the response includes only keys for which the delete operation encountered an error. If all keys are successfully deleted when you're using quiet mode, Amazon S3 returns an empty response.

To learn more about object deletion, see [Deleting Objects \(p. 227\)](#).

You can use the REST API directly or use the AWS SDKs.

Deleting Multiple Objects Using the AWS SDK for Java

The AWS SDK for Java provides the `AmazonS3Client.deleteObjects()` method for deleting multiple objects. For each object that you want to delete, you specify the key name. If the bucket is versioning-enabled, you have the following options:

- Specify only the object's key name. Amazon S3 will add a delete marker to the object.
- Specify both the object's key name and a version ID to be deleted. Amazon S3 will delete the specified version of the object.

Example

The following example uses the Multi-Object Delete API to delete objects from a bucket that is not version-enabled. The example uploads sample objects to the bucket and then uses the `AmazonS3Client.deleteObjects()` method to delete the objects in a single request. In the `DeleteObjectsRequest`, the example specifies only the object key names because the objects do not have version IDs.

For more information about deleting objects, see [Deleting Objects \(p. 227\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

import java.io.IOException;

public class DeleteObjectNonVersionedBucket {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            s3Client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Example

The following example uses the Multi-Object Delete API to delete objects from a version-enabled bucket. It does the following:

1. Creates sample objects and then deletes them, specifying the key name and version ID for each object to delete. The operation deletes only the specified object versions.
2. Creates sample objects and then deletes them by specifying only the key names. Because the example doesn't specify version IDs, the operation adds a delete marker to each object, without deleting any specific object versions. After the delete markers are added, these objects will not appear in the AWS Management Console.
3. Remove the delete markers by specifying the object keys and version IDs of the delete markers. The operation deletes the delete markers, which results in the objects reappearing in the AWS Management Console.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
```

```
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.DeleteObjectsResult.DeletedObject;
import com.amazonaws.services.s3.model.PutObjectResult;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class DeleteMultipleObjectsVersionEnabledBucket {
    private static AmazonS3 S3_CLIENT;
    private static String VERSIONED_BUCKET_NAME;

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        VERSIONED_BUCKET_NAME = "*** Bucket name ***";

        try {
            S3_CLIENT = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Check to make sure that the bucket is versioning-enabled.
            String bucketVersionStatus =
                S3_CLIENT.getBucketVersioningConfiguration(VERSIONED_BUCKET_NAME).getStatus();
            if (!bucketVersionStatus.equals(BucketVersioningConfiguration.ENABLED)) {
                System.out.printf("Bucket %s is not versioning-enabled.",
                    VERSIONED_BUCKET_NAME);
            } else {
                // Upload and delete sample objects, using specific object versions.
                uploadAndDeleteObjectsWithVersions();

                // Upload and delete sample objects without specifying version IDs.
                // Amazon S3 creates a delete marker for each object rather than deleting
                // specific versions.
                DeleteObjectsResult unversionedDeleteResult =
                    uploadAndDeleteObjectsWithoutVersions();

                // Remove the delete markers placed on objects in the non-versioned create/
                delete method.
                multiObjectVersionedDeleteRemoveDeleteMarkers(unversionedDeleteResult);
            }
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }

    private static void uploadAndDeleteObjectsWithVersions() {
        System.out.println("Uploading and deleting objects with versions specified.");

        // Upload three sample objects.
        ArrayList<KeyVersion> keys = new ArrayList<KeyVersion>();
        for (int i = 0; i < 3; i++) {
            String keyName = "delete object without version ID example " + i;
            PutObjectResult putResult = S3_CLIENT.putObject(VERSIONED_BUCKET_NAME, keyName,
                "Object number " + i + " to be deleted.");
            // Gather the new object keys with version IDs.
            keys.add(new KeyVersion(keyName, putResult.getVersionId()));
        }
    }
}
```

```

    }

    // Delete the specified versions of the sample objects.
    DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest(VERSIONED_BUCKET_NAME)
        .withKeys(keys)
        .withQuiet(false);

    // Verify that the object versions were successfully deleted.
    DeleteObjectsResult delObjRes = S3_CLIENT.deleteObjects(multiObjectDeleteRequest);
    int successfulDeletes = delObjRes.getDeletedObjects().size();
    System.out.println(successfulDeletes + " objects successfully deleted");
}

private static DeleteObjectsResult uploadAndDeleteObjectsWithoutVersions() {
    System.out.println("Uploading and deleting objects with no versions specified.");

    // Upload three sample objects.
    ArrayList<KeyVersion> keys = new ArrayList<KeyVersion>();
    for (int i = 0; i < 3; i++) {
        String keyName = "delete object with version ID example " + i;
        S3_CLIENT.putObject(VERSIONED_BUCKET_NAME, keyName, "Object number " + i + " to
be deleted.");
        // Gather the new object keys without version IDs.
        keys.add(new KeyVersion(keyName));
    }

    // Delete the sample objects without specifying versions.
    DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest(VERSIONED_BUCKET_NAME).withKeys(keys)
        .withQuiet(false);

    // Verify that delete markers were successfully added to the objects.
    DeleteObjectsResult delObjRes = S3_CLIENT.deleteObjects(multiObjectDeleteRequest);
    int successfulDeletes = delObjRes.getDeletedObjects().size();
    System.out.println(successfulDeletes + " objects successfully marked for deletion
without versions.");
    return delObjRes;
}

private static void multiObjectVersionedDeleteRemoveDeleteMarkers(DeleteObjectsResult
response) {
    List<KeyVersion> keyList = new ArrayList<KeyVersion>();
    for (DeletedObject deletedObject : response.getDeletedObjects()) {
        // Note that the specified version ID is the version ID for the delete marker.
        keyList.add(new KeyVersion(deletedObject.getKey(),
deletedObject.getDeleteMarkerVersionId()));
    }
    // Create a request to delete the delete markers.
    DeleteObjectsRequest deleteRequest = new
DeleteObjectsRequest(VERSIONED_BUCKET_NAME).withKeys(keyList);

    // Delete the delete markers, leaving the objects intact in the bucket.
    DeleteObjectsResult delObjRes = S3_CLIENT.deleteObjects(deleteRequest);
    int successfulDeletes = delObjRes.getDeletedObjects().size();
    System.out.println(successfulDeletes + " delete markers successfully deleted");
}
}

```

Deleting Multiple Objects Using the AWS SDK for .NET

The AWS SDK for .NET provides a convenient method for deleting multiple objects: `DeleteObjects`. For each object that you want to delete, you specify the key name and the version of the object. If the

bucket is not versioning-enabled, you specify `null` for the version ID. If an exception occurs, review the `DeleteObjectsException` response to determine which objects were not deleted and why.

Example Deleting Multiple Objects from a Non-Versioning Bucket

The following C# example uses the multi-object delete API to delete objects from a bucket that is not version-enabled. The example uploads the sample objects to the bucket, and then uses the `DeleteObjects` method to delete the objects in a single request. In the `DeleteObjectsRequest`, the example specifies only the object key names because the version IDs are null.

For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DeleteMultipleObjectsNonVersionedBucketTest
    {
        private const string bucketName = "*** versioning-enabled bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            MultiObjectDeleteAsync().Wait();
        }

        static async Task MultiObjectDeleteAsync()
        {
            // Create sample objects (for subsequent deletion).
            var keysAndVersions = await PutObjectsAsync(3);

            // a. multi-object delete by specifying the key names and version IDs.
            DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keysAndVersions // This includes the object keys and null version
                IDs.
            };
            // You can add specific object key to the delete request using the .AddKey.
            // multiObjectDeleteRequest.AddKey("TickerReference.csv", null);
            try
            {
                DeleteObjectsResponse response = await
                s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
                response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException e)
            {
                PrintDeletionErrorStatus(e);
            }
        }

        private static void PrintDeletionErrorStatus(DeleteObjectsException e)
        {

```



```
// var errorResponse = e.ErrorResponse;
DeleteObjectsResponse errorResponse = e.Response;
Console.WriteLine("x {0}", errorResponse.DeletedObjects.Count);

Console.WriteLine("No. of objects successfully deleted = {0}",
errorResponse.DeletedObjects.Count);
Console.WriteLine("No. of objects failed to delete = {0}",
errorResponse.DeleteErrors.Count);

Console.WriteLine("Printing error data...");
foreach (DeleteError deleteError in errorResponse.DeleteErrors)
{
    Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key,
deleteError.Code, deleteError.Message);
}

static async Task<List<KeyVersion>> PutObjectsAsync(int number)
{
    List<KeyVersion> keys = new List<KeyVersion>();
    for (int i = 0; i < number; i++)
    {
        string key = "ExampleObject-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = key,
            ContentBody = "This is the content body!",
        };

        PutObjectResponse response = await s3Client.PutObjectAsync(request);
        KeyVersion keyVersion = new KeyVersion
        {
            Key = key,
            // For non-versioned bucket operations, we only need object key.
            // VersionId = response.VersionId
        };
        keys.Add(keyVersion);
    }
    return keys;
}
}
```

Example Multi-Object Deletion for a Version-Enabled Bucket

The following C# example uses the multi-object delete API to delete objects from a version-enabled bucket. The example performs the following actions:

1. Creates sample objects and deletes them by specifying the key name and version ID for each object. The operation deletes specific versions of the objects.
2. Creates sample objects and deletes them by specifying only the key names. Because the example doesn't specify version IDs, the operation only adds delete markers. It doesn't delete any specific versions of the objects. After deletion, these objects don't appear in the Amazon S3 console.
3. Deletes the delete markers by specifying the object keys and version IDs of the delete markers. When the operation deletes the delete markers, the objects reappear in the console.

For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
```

```
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DeleteMultipleObjVersionedBucketTest
    {
        private const string bucketName = "**** versioning-enabled bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            DeleteMultipleObjectsFromVersionedBucketAsync().Wait();
        }

        private static async Task DeleteMultipleObjectsFromVersionedBucketAsync()
        {
            // Delete objects (specifying object version in the request).
            await DeleteObjectVersionsAsync();

            // Delete objects (without specifying object version in the request).
            var deletedObjects = await DeleteObjectsAsync();

            // Additional exercise - remove the delete markers S3 returned in the preceding
            response.
            // This results in the objects reappearing in the bucket (you can
            // verify the appearance/disappearance of objects in the console).
            await RemoveDeleteMarkersAsync(deletedObjects);
        }

        private static async Task<List<DeletedObject>> DeleteObjectsAsync()
        {
            // Upload the sample objects.
            var keysAndVersions2 = await PutObjectsAsync(3);

            // Delete objects using only keys. Amazon S3 creates a delete marker and
            // returns its version ID in the response.
            List<DeletedObject> deletedObjects = await
            NonVersionedDeleteAsync(keysAndVersions2);
            return deletedObjects;
        }

        private static async Task DeleteObjectVersionsAsync()
        {
            // Upload the sample objects.
            var keysAndVersions1 = await PutObjectsAsync(3);

            // Delete the specific object versions.
            await VersionedDeleteAsync(keysAndVersions1);
        }

        private static void PrintDeletionReport(DeleteObjectsException e)
        {
            var errorResponse = e.Response;
            Console.WriteLine("No. of objects successfully deleted = {0}",
            errorResponse.DeletedObjects.Count);
            Console.WriteLine("No. of objects failed to delete = {0}",
            errorResponse.DeleteErrors.Count);
            Console.WriteLine("Printing error data...");
            foreach (var deleteError in errorResponse.DeleteErrors)

```

```

        {
            Console.WriteLine("Object Key: {0}\\t{1}\\t{2}", deleteError.Key,
deleteError.Code, deleteError.Message);
        }
    }

    static async Task VersionedDeleteAsync(List<KeyVersion> keys)
    {
        // a. Perform a multi-object delete by specifying the key names and version
IDs.
        var multiObjectDeleteRequest = new DeleteObjectsRequest
        {
            BucketName = bucketName,
            Objects = keys // This includes the object keys and specific version IDs.
        };
        try
        {
            Console.WriteLine("Executing VersionedDelete...");
            DeleteObjectsResponse response = await
s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);
            Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
        }
        catch (DeleteObjectsException e)
        {
            PrintDeletionReport(e);
        }
    }

    static async Task<List<DeletedObject>> NonVersionedDeleteAsync(List<KeyVersion>
keys)
    {
        // Create a request that includes only the object key names.
        DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest();
        multiObjectDeleteRequest.BucketName = bucketName;

        foreach (var key in keys)
        {
            multiObjectDeleteRequest.AddKey(key.Key);
        }
        // Execute DeleteObjects - Amazon S3 add delete marker for each object
// deletion. The objects disappear from your bucket.
// You can verify that using the Amazon S3 console.
        DeleteObjectsResponse response;
        try
        {
            Console.WriteLine("Executing NonVersionedDelete...");
            response = await s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);
            Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
        }
        catch (DeleteObjectsException e)
        {
            PrintDeletionReport(e);
            throw; // Some deletes failed. Investigate before continuing.
        }
        // This response contains the DeletedObjects list which we use to delete the
delete markers.
        return response.DeletedObjects;
    }

    private static async Task RemoveDeleteMarkersAsync(List<DeletedObject>
deletedObjects)
    {
        var keyVersionList = new List<KeyVersion>();
    }

```

```

        foreach (var deletedObject in deletedObjects)
        {
            KeyVersion keyVersion = new KeyVersion
            {
                Key = deletedObject.Key,
                VersionId = deletedObject.DeleteMarkerVersionId
            };
            keyVersionList.Add(keyVersion);
        }
        // Create another request to delete the delete markers.
        var multiObjectDeleteRequest = new DeleteObjectsRequest
        {
            BucketName = bucketName,
            Objects = keyVersionList
        };

        // Now, delete the delete marker to bring your objects back to the bucket.
        try
        {
            Console.WriteLine("Removing the delete markers .....");
            var deleteObjectResponse = await
s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);
            Console.WriteLine("Successfully deleted all the {0} delete markers",
                             deleteObjectResponse.DeletedObjects.Count);
        }
        catch (DeleteObjectsException e)
        {
            PrintDeletionReport(e);
        }
    }

    static async Task<List<KeyVersion>> PutObjectsAsync(int number)
    {
        var keys = new List<KeyVersion>();

        for (var i = 0; i < number; i++)
        {
            string key = "ObjectToDelete-" + new System.Random().Next();
            PutObjectRequest request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = key,
                ContentBody = "This is the content body!",
            };

            var response = await s3Client.PutObjectAsync(request);
            KeyVersion keyVersion = new KeyVersion
            {
                Key = key,
                VersionId = response.VersionId
            };

            keys.Add(keyVersion);
        }
        return keys;
    }
}

```

Deleting Multiple Objects Using the AWS SDK for PHP

This topic shows how to use classes from version 3 of the AWS SDK for PHP to delete multiple objects from versioned and non-versioned Amazon S3 buckets. For more information about versioning, see [Using Versioning \(p. 432\)](#).

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

Example Deleting Multiple Objects from a Non-Versioned Bucket

The following PHP example uses the `deleteObjects()` method to delete multiple objects from a bucket that is not version-enabled.

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

```
<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// 1. Create a few objects.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject([
        'Bucket' => $bucket,
        'Key'    => "key{$i}",
        'Body'   => "content {$i}",
    ]);
}

// 2. List the objects and get the keys.
$keys = $s3->listObjects([
    'Bucket' => $bucket
])->getPath('Contents/*/Key');

// 3. Delete the objects.
$s3->deleteObjects([
    'Bucket' => $bucket,
    'Delete' => [
        'Objects' => array_map(function ($key) {
            return ['Key' => $key];
        }, $keys)
    ],
]);
```

Example Deleting Multiple Objects from a Version-enabled Bucket

The following PHP example uses the `deleteObjects()` method to delete multiple objects from a version-enabled bucket.

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 679\)](#).

```
<?php
```

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

// 1. Enable object versioning for the bucket.
$s3->putBucketVersioning([
    'Bucket' => $bucket,
    'Status' => 'Enabled',
]);

// 2. Create a few versions of an object.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject([
        'Bucket' => $bucket,
        'Key' => $keyname,
        'Body' => "content {$i}",
    ]);
}

// 3. List the objects versions and get the keys and version IDs.
$versions = $s3->listObjectVersions(['Bucket' => $bucket])
    ->getPath('Versions');

// 4. Delete the object versions.
$s3->deleteObjects([
    'Bucket' => $bucket,
    'Delete' => [
        'Objects' => array_map(function ($version) {
            return [
                'Key' => $version['Key'],
                'VersionId' => $version['VersionId']
            ], $versions),
    ],
]);

echo "The following objects were deleted successfully:". PHP_EOL;
foreach ($result['Deleted'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}" . PHP_EOL;
}

echo PHP_EOL . "The following objects could not be deleted:" . PHP_EOL;
foreach ($result['Errors'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}" . PHP_EOL;
}

// 5. Suspend object versioning for the bucket.
$s3->putBucketVersioning([
    'Bucket' => $bucket,
    'Status' => 'Suspended',
]);
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Deleting Multiple Objects Using the REST API

You can use the AWS SDKs to delete multiple objects using the Multi-Object Delete API. However, if your application requires it, you can send REST requests directly. For more information, go to [Delete Multiple Objects](#) in the *Amazon Simple Storage Service API Reference*.

Selecting Content from Objects

With Amazon S3 Select, you can use simple structured query language (SQL) statements to filter the contents of Amazon S3 objects and retrieve just the subset of data that you need. By using Amazon S3 Select to filter this data, you can reduce the amount of data that Amazon S3 transfers, which reduces the cost and latency to retrieve this data.

Amazon S3 Select works on objects stored in CSV, JSON, or Apache Parquet format. It also works with objects that are compressed with GZIP or BZIP2 (for CSV and JSON objects only), and server-side encrypted objects. You can specify the format of the results as either CSV or JSON, and you can determine how the records in the result are delimited.

You pass SQL expressions to Amazon S3 in the request. Amazon S3 Select supports a subset of SQL. For more information about the SQL elements that are supported by Amazon S3 Select, see [SQL Reference for Amazon S3 Select and Glacier Select \(p. 714\)](#).

You can perform SQL queries using AWS SDKs, the SELECT Object Content REST API, the AWS Command Line Interface (AWS CLI), or the Amazon S3 console. The Amazon S3 console limits the amount of data returned to 40 MB. To retrieve more data, use the AWS CLI or the API.

Requirements and Limits

The following are requirements for using Amazon S3 Select:

- You must have `s3:GetObject` permission for the object you are querying.
- If the object you are querying is encrypted with a customer-provided encryption key (SSE-C), you must use `https`, and you must provide the encryption key in the request.

The following limits apply when using Amazon S3 Select:

- The maximum length of a SQL expression is 256 KB.
- The maximum length of a record in the result is 1 MB.

Additional limitations apply when using Amazon S3 Select with Parquet objects:

- Amazon S3 Select supports only columnar compression using GZIP or Snappy. Amazon S3 Select doesn't support whole-object compression for Parquet objects.
- Amazon S3 Select doesn't support Parquet output. You must specify the output format as CSV or JSON.
- The maximum uncompressed block size is 256 MB.
- The maximum number of columns is 100.
- You must use the data types specified in the object's schema.
- Selecting on a repeated field returns only the last value.

Constructing a Request

When you construct a request, you provide details of the object that is being queried using an `InputSerialization` object. You provide details of how the results are to be returned using an

`OutputSerialization` object. You also include the SQL expression that Amazon S3 uses to filter the request.

For more information about constructing an Amazon S3 Select request, see [SELECT Object Content](#) in the *Amazon Simple Storage Service API Reference*. You can also see one of the SDK code examples in the following sections.

Errors

Amazon S3 Select returns an error code and associated error message when an issue is encountered while attempting to execute a query. For a list of error codes and descriptions, see the [Special Errors](#) section of the *SELECT Object Content* page in the *Amazon Simple Storage Service API Reference*.

Topics

- [Related Resources](#) (p. 246)
- [Selecting Content from Objects Using the SDK for Java](#) (p. 246)
- [Selecting Content from Objects Using the REST API](#) (p. 248)
- [Selecting Content from Objects Using Other SDKs](#) (p. 248)

Related Resources

- [Using the AWS SDKs, CLI, and Explorers](#) (p. 669)

Selecting Content from Objects Using the SDK for Java

You use Amazon S3 Select to select contents of an object with Java using the `selectObjectContent` method, which on success returns the results of the SQL expression. The specified bucket and object key must exist, or an error results.

Example Example

The following Java code returns the value of the first column for each record that is stored in an object that contains data stored in CSV format. It also requests `Progress` and `Stats` messages to be returned. You must provide a valid bucket name and an object that contains data in CSV format.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
package com.amazonaws;

import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CSVInput;
import com.amazonaws.services.s3.model.CSVOutput;
import com.amazonaws.services.s3.model.CompressionType;
import com.amazonaws.services.s3.model.ExpressionType;
import com.amazonaws.services.s3.model.InputSerialization;
import com.amazonaws.services.s3.model.OutputSerialization;
import com.amazonaws.services.s3.model.SelectObjectContentEvent;
import com.amazonaws.services.s3.model.SelectObjectContentEventVisitor;
import com.amazonaws.services.s3.model.SelectObjectContentRequest;
import com.amazonaws.services.s3.model.SelectObjectContentResult;

import java.io.File;
import java.io.FileOutputStream;
```



```
import java.io.InputStream;
import java.io.OutputStream;
import java.util.concurrent.atomic.AtomicBoolean;

import static com.amazonaws.util.IOUtils.copy;

/**
 * This example shows how to query data from S3Select and consume the response in the form
 * of an
 * InputStream of records and write it to a file.
 */

public class RecordInputStreamExample {

    private static final String BUCKET_NAME = "${my-s3-bucket}";
    private static final String CSV_OBJECT_KEY = "${my-csv-object-key}";
    private static final String S3_SELECT_RESULTS_PATH = "${my-s3-select-results-path}";
    private static final String QUERY = "select s._1 from S3Object s";

    public static void main(String[] args) throws Exception {
        final AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

        SelectObjectContentRequest request = generateBaseCSVRequest(BUCKET_NAME,
        CSV_OBJECT_KEY, QUERY);
        final AtomicBoolean isResultComplete = new AtomicBoolean(false);

        try (OutputStream fileOutputStream = new FileOutputStream(new File
        (S3_SELECT_RESULTS_PATH));
            SelectObjectContentResult result = s3Client.selectObjectContent(request)) {
            InputStream resultInputStream = result.getPayload().getRecordsInputStream(
                new SelectObjectContentEventVisitor() {
                    @Override
                    public void visit(SelectObjectContentEvent.StatsEvent event)
                    {
                        System.out.println(
                            "Received Stats, Bytes Scanned: " +
                            event.getDetails().getBytesScanned()
                            + " Bytes Processed: " +
                            event.getDetails().getBytesProcessed());
                    }

                    /*
                     * An End Event informs that the request has finished successfully.
                     */
                    @Override
                    public void visit(SelectObjectContentEvent.EndEvent event)
                    {
                        isResultComplete.set(true);
                        System.out.println("Received End Event. Result is complete.");
                    }
                }
            );

            copy(resultInputStream, fileOutputStream);
        }

        /*
         * The End Event indicates all matching records have been transmitted.
         * If the End Event is not received, the results may be incomplete.
         */
        if (!isResultComplete.get()) {
            throw new Exception("S3 Select request was incomplete as End Event was not
            received.");
        }
    }
}
```

```
private static SelectObjectContentRequest generateBaseCSVRequest(String bucket, String
key, String query) {
    SelectObjectContentRequest request = new SelectObjectContentRequest();
    request.setBucketName(bucket);
    request.setKey(key);
    request.setExpression(query);
    request.setExpressionType(ExpressionType.SQL);

    InputSerialization inputSerialization = new InputSerialization();
    inputSerialization.setCsv(new CSVInput());
    inputSerialization.setCompressionType(CompressionType.NONE);
    request.setInputSerialization(inputSerialization);

    OutputSerialization outputSerialization = new OutputSerialization();
    outputSerialization.setCsv(new CSVOutput());
    request.setOutputSerialization(outputSerialization);

    return request;
}
}
```

Selecting Content from Objects Using the REST API

You can use the AWS SDK to select content from objects. However, if your application requires it, you can send REST requests directly. For more information about the request and response format, see [SELECT Object Content](#).

Selecting Content from Objects Using Other SDKs

You can select the contents of an object using Amazon S3 Select using other SDKs. For more information, see the following:

- Python: [Using the AWS SDK for Python \(Boto\)](#) (p. 681).

Restoring Archived Objects

Objects that you archive to the GLACIER or DEEP_ARCHIVE storage classes are not accessible in real time. You must first initiate a restore request, and then wait until a temporary copy of the object is available for the duration (number of days) that you specify in the request. For more information about how the GLACIER, DEEP_ARCHIVE, and other Amazon S3 storage classes compare, see [Amazon S3 Storage Classes](#) (p. 103).

Amazon S3 restores a temporary copy of the object only for the specified duration. After that, it deletes the restored object copy. You can modify the expiration period of a restored copy by reissuing a restore. In this case, Amazon S3 updates the expiration period relative to the current time.

Amazon S3 calculates the expiration time of the restored object copy by adding the number of days specified in the restoration request to the current time. It then rounds the resulting time to the next day at midnight Universal Coordinated Time (UTC). For example, suppose that an object was created on October 15, 2012 10:30 AM UTC, and the restoration period was specified as three days. In this case, the restored copy expires on October 19, 2012 00:00 UTC, at which time Amazon S3 deletes the object copy.

After you receive a temporary copy of the restored object, the object's storage class remains GLACIER or DEEP_ARCHIVE. (A [HEAD Object](#) or the [GET Object](#) API operations request returns GLACIER or DEEP_ARCHIVE as the storage class.)

The time it takes a restore job to finish depends on which archive storage class you use and which retrieval option you specify: [Expedited](#) (only available for GLACIER), [Standard](#), or [Bulk](#). You can be

notified when your restore is complete using Amazon S3 event notifications. For more information, see [Configuring Amazon S3 Event Notifications](#) (p. 530).

You can restore an object copy for any number of days. However you should restore objects only for the duration that you need because of the storage costs associated with the object copy. When you restore an archive, you pay for both the archive (at the GLACIER or DEEP_ARCHIVE rate) and a copy that you restored temporarily (Reduced Redundancy Storage (RRS) rate). For information about pricing, see [Amazon S3 Pricing](#).

When required, you can restore large segments of the data stored in the GLACIER and DEEP_ARCHIVE storage classes. For example, you might want to restore data for a secondary copy. However, if you need to restore a large amount of data, keep in mind that the GLACIER and DEEP_ARCHIVE storage classes are designed for 35 random restore requests per petabyte (PiB) stored per day.

For information about using lifecycle transitions to move objects to the GLACIER or DEEP_ARCHIVE storage classes, see [Transitioning to the GLACIER and DEEP_ARCHIVE Storage Classes \(Object Archival\)](#) (p. 123).

To restore more than one Amazon S3 object with a single request, you can use Amazon S3 batch operations. You provide Amazon S3 batch operations with a list of objects to operate on. Amazon S3 batch operations call the respective API to perform the specified operation. A single Amazon S3 batch operations job can perform the specified operation on billions of objects containing exabytes of data.

Amazon S3 batch operations track progress, send notifications, and store a detailed completion report of all actions, providing a fully managed, auditable, serverless experience. You can use Amazon S3 batch operations through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see [the section called “The Basics: Jobs”](#) (p. 468).

The following sections provide more information about restoring objects.

Topics

- [Archive Retrieval Options](#) (p. 249)
- [Upgrading the Speed of an In-Progress Restore](#) (p. 250)
- [Restore an Archived Object Using the Amazon S3 Console](#) (p. 250)
- [Restore an Archived Object Using the AWS SDK for Java](#) (p. 251)
- [Restore an Archived Object Using the AWS SDK for .NET](#) (p. 252)
- [Restore an Archived Object Using the REST API](#) (p. 253)

Archive Retrieval Options

The following are the available retrieval options when restoring an archived object:

- **Expedited** - Expedited retrievals allow you to quickly access your data stored in the GLACIER storage class when occasional urgent requests for a subset of archives are required. For all but the largest archived objects (250 MB+), data accessed using Expedited retrievals is typically made available within 1–5 minutes. Provisioned capacity ensures that retrieval capacity for Expedited retrievals is available when you need it. For more information, see [Provisioned Capacity](#) (p. 250). Expedited retrievals and provisioned capacity are not available for objects stored in the DEEP_ARCHIVE storage class.
- **Standard** - Standard retrievals allow you to access any of your archived objects within several hours. This is the default option for the GLACIER and DEEP_ARCHIVE retrieval requests that do not specify the retrieval option. Standard retrievals typically finish within 3–5 hours for objects stored in the GLACIER storage class. They typically finish within 12 hours for objects stored in the DEEP_ARCHIVE storage class.
- **Bulk** - Bulk retrievals are the lowest-cost retrieval option in Amazon S3 Glacier, enabling you to retrieve large amounts, even petabytes, of data inexpensively. Bulk retrievals typically finish within

5–12 hours for objects stored in the GLACIER storage class. They typically finish within 48 hours for objects stored in the DEEP_ARCHIVE storage class.

The following table summarizes the archival retrieval options.

To make an Expedited, Standard, or Bulk retrieval, set the `Tier` request element in the [POST Object restore](#) REST API request to the option you want, or the equivalent in the AWS CLI or AWS SDKs. If you purchased provisioned capacity, all Expedited retrievals are automatically served through your provisioned capacity.

You can restore an archived object programmatically or by using the Amazon S3 console. Amazon S3 processes only one restore request at a time per object. You can use both the console and the Amazon S3 API to check the restoration status and to find out when Amazon S3 will delete the restored copy.

Provisioned Capacity

Provisioned capacity ensures that your retrieval capacity for expedited retrievals is available when you need it. Each unit of capacity provides that at least three expedited retrievals can be performed every 5 minutes, and it provides up to 150 MB/s of retrieval throughput.

If your workload requires highly reliable and predictable access to a subset of your data in minutes, you should purchase provisioned retrieval capacity. Without provisioned capacity, Expedited retrievals might not be accepted during periods of high demand. If you require access to Expedited retrievals under all circumstances, we recommend that you purchase provisioned retrieval capacity.

You can purchase provisioned capacity using the Amazon S3 console, the Amazon S3 Glacier console, the [Purchase Provisioned Capacity](#) REST API, the AWS SDKs, or the AWS CLI. For provisioned capacity pricing information, see [Amazon S3 Pricing](#).

Expedited retrievals using provisioned capacity still incur request and retrieval charges, and are not available for the DEEP_ARCHIVE storage class.

Upgrading the Speed of an In-Progress Restore

Using Amazon S3 restore speed upgrade, you can change the restore speed to a faster speed while the restore is in progress. A restore speed upgrade overrides an in-progress restore with a faster restore tier. You cannot slow down an in-progress restore.

To upgrade the speed of an in-progress restoration, issue another restore request to the same object that sets a new `Tier` request element in the [POST Object restore](#) REST API, or the equivalent in the AWS CLI or AWS SDKs. When issuing a request to upgrade the restore tier, you must choose a tier that is faster than the tier that the in-progress restore is using. You must not change any other parameters, such as the `Days` request element.

You can be notified of the completion of the restore by using Amazon S3 event notifications. Restores are charged at the price of the upgraded tier. For information about restore pricing, see [Amazon S3 Pricing](#).

Restore an Archived Object Using the Amazon S3 Console

You can use the Amazon S3 console to restore a copy of an object that has been archived to Amazon S3 Glacier. For instructions on how to restore an archive using the AWS Management Console, see [How Do I Restore an S3 Object that has been Archived to Amazon S3 Glacier?](#) in the *Amazon Simple Storage Service Console User Guide*.

When you restore an archive, you are paying for both the archive and a copy you restored temporarily. For information about pricing, see [Amazon S3 Pricing](#).

Restore an Archived Object Using the AWS SDK for Java

Example

The following example shows how to restore an object archived to Amazon S3 Glacier using the AWS SDK for Java. The example initiates a restoration request for the specified archived object and checks its restoration status. For more information about restoring archived objects, see [Restoring Archived Objects \(p. 248\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.RestoreObjectRequest;

import java.io.IOException;

public class RestoreArchivedObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Object key ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Create and submit a request to restore an object from Glacier for two days.
            RestoreObjectRequest requestRestore = new RestoreObjectRequest(bucketName,
keyName, 2);
            s3Client.restoreObjectV2(requestRestore);

            // Check the restoration status of the object.
            ObjectMetadata response = s3Client.getObjectMetadata(bucketName, keyName);
            Boolean restoreFlag = response.getOngoingRestore();
            System.out.format("Restoration status: %s.\n",
                restoreFlag ? "in progress" : "not in progress (finished or failed)");
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Restore an Archived Object Using the AWS SDK for .NET

Example

The following C# example initiates a request to restore an archived object for 2 days. Amazon S3 maintains the restoration status in the object metadata. After initiating the request, the example retrieves the object metadata and checks the value of the `RestoreInProgress` property. For instructions on creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class RestoreArchivedObjectTest
    {
        private const string bucketName = "**** bucket name ****";
        private const string objectKey = "*** archived object key name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            RestoreObjectAsync(client, bucketName, objectKey).Wait();
        }

        static async Task RestoreObjectAsync(IAmazonS3 client, string bucketName, string
objectKey)
        {
            try
            {
                var restoreRequest = new RestoreObjectRequest
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Days = 2
                };
                RestoreObjectResponse response = await
client.RestoreObjectAsync(restoreRequest);

                // Check the status of the restoration.
                await CheckRestorationStatusAsync(client, bucketName, objectKey);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("An AmazonS3Exception was thrown. Exception: " +
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }

        static async Task CheckRestorationStatusAsync(IAmazonS3 client, string bucketName,
string objectKey)
        {
            GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
```

```
        {
            BucketName = bucketName,
            Key = objectKey
        };
        GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
        Console.WriteLine("restoration status: {0}", response.RestoreInProgress ? "in-
progress" : "finished or failed");
    }
}
```

Restore an Archived Object Using the REST API

Amazon S3 provides an API for you to initiate an archive restoration. For more information, see [POST Object restore](#) in the *Amazon Simple Storage Service API Reference*.

Querying Archived Objects

With the select type of [POST Object restore](#), you can perform filtering operations using simple Structured Query Language (SQL) statements directly on your data that is archived by Amazon S3 to Glacier. When you provide an SQL query for an archived object, select runs the query in place and writes the output results to an S3 bucket. You can run queries and custom analytics on your data that is stored in Glacier, without having to restore your entire object to Amazon S3.

When you perform select queries, Glacier provides three data access tiers—*expedited*, *standard*, and *bulk*. All of these tiers provide different data access times and costs, and you can choose any one of them depending on how quickly you want your data to be available. For more information, see [Data Access Tiers](#) (p. 255).

You can use the select type of restore with the AWS SDKs, the Glacier REST API, and the AWS Command Line Interface (AWS CLI).

Topics

- [Select Requirements and Limits](#) (p. 253)
- [How Do I Query Data Using Select?](#) (p. 254)
- [Error Handling](#) (p. 255)
- [Data Access Tiers](#) (p. 255)
- [More Info](#) (p. 256)

Select Requirements and Limits

The following are requirements for using select:

- Archive objects that are queried by select must be formatted as uncompressed comma-separated values (CSV).
- An S3 bucket for output. The AWS account that you use to initiate an Glacier select job must have write permissions for the S3 bucket. The Amazon S3 bucket must be in the same AWS Region as the bucket that contains the archived object that is being queried.
- The requesting AWS account must have permissions to perform the `s3:RestoreObject` and `s3:GetObject` actions. For more information about these permissions, see [Permissions Related to Bucket Subresource Operations](#) (p. 347).
- The archive must not be encrypted with SSE-C or client-side encryption.

The following limits apply when using select:

- There are no limits on the number of records that select can process. An input or output record must not exceed 1 MB; otherwise, the query fails. There is a limit of 1,048,576 columns per record.
- There is no limit on the size of your final result. However, your results are broken into multiple parts.
- An SQL expression is limited to 128 KB.

How Do I Query Data Using Select?

Using select, you can use SQL commands to query Glacier archive objects that are in encrypted uncompressed CSV format. With this restriction, you can perform simple query operations on your text-based data in Glacier. For example, you might look for a specific name or ID among a set of archived text files.

To query your Glacier data, create a select request using the [POST Object restore](#) operation. When performing a select request, you provide the SQL expression, the archive to query, and the location to store the results.

The following example expression returns all records from the archived object specified in [POST Object restore](#).

```
SELECT * FROM object
```

Glacier Select supports a subset of the ANSI SQL language. It supports common filtering SQL clauses like `SELECT`, `FROM`, and `WHERE`. It does not support `SUM`, `COUNT`, `GROUP BY`, `JOINS`, `DISTINCT`, `UNION`, `ORDER BY`, and `LIMIT`. For more information about support for SQL, see [SQL Reference for Amazon S3 Select and Glacier Select](#) in the *Amazon Simple Storage Service Developer Guide*.

Select Output

When you initiate a select request, you define an output location for the results of your select query. This location must be an Amazon S3 bucket in the same AWS Region as the bucket that contains the archived object that is being queried. The AWS account that initiates the job must have permissions to write to the S3 bucket.

You can specify the Amazon S3 storage class and encryption for the output objects stored in Amazon S3. Select supports SSE-KMS and SSE-S3 encryption. Select doesn't support SSE-C and client-side encryption. For more information about Amazon S3 storage classes and encryption, see [Amazon S3 Storage Classes \(p. 103\)](#) and [Protecting Data Using Server-Side Encryption \(p. 265\)](#).

Glacier Select results are stored in the S3 bucket using the prefix provided in the output location specified in [POST Object restore](#). From this information, select creates a unique prefix referring to the job ID. (Prefixes are used to group Amazon S3 objects together by beginning object names with a common string.) Under this unique prefix, there are two new prefixes created, `results` for results and `errors` for logs and errors. Upon completion of the job, a result manifest is written which contains the location of all results.

There is also a placeholder file named `job.txt` that is written to the output location. After it is written it is never updated. The placeholder file is used for the following:

- Validation of the write permission and majority of SQL syntax errors synchronously.
- Providing a static output about your select request that you can easily reference whenever you want.

For example, suppose that you make a select request with the output location for the results specified as `s3://example-bucket/my-prefix`, and the job response returns the job ID as `examplekne1209ualkdjh812elkassdu9012e`. After the select job finishes, you can see the following Amazon S3 objects in your bucket:


```
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/job.txt  
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/results/abc  
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/results/def  
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/results/ghi  
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/result_manifest.txt
```

The select query results are broken into multiple parts. In the example, select uses the prefix that you specified when setting the output location and appends the job ID and the `results` prefix. It then writes the results in three parts, with the object names ending in `abc`, `def`, and `ghi`. The result manifest contains all three files to allow programmatic retrieval. If the job fails with any error, then a file is visible under the error prefix and an `error_manifest.txt` is produced.

Presence of a `result_manifest.txt` file along with the absence of `error_manifest.txt` guarantees that the job finished successfully. There is no guarantee provided on how results are ordered.

Note

The length of an Amazon S3 object name, also referred to as the *key*, can be no more than 1,024 bytes. Glacier select reserves 128 bytes for prefixes. And, the length of your Amazon S3 location path cannot be more than 512 bytes. A request with a length greater than 512 bytes returns an exception, and the request is not accepted.

Error Handling

Select notifies you of two kinds of errors. The first set of errors is sent to you synchronously when you submit the query in [POST Object restore](#). These errors are sent to you as part of the HTTP response. Another set of errors can occur after the query has been accepted successfully, but they happen during query execution. In this case, the errors are written to the specified output location under the `errors` prefix.

Select stops executing the query after encountering an error. To execute the query successfully, you must resolve all errors. You can check the logs to identify which records caused a failure.

Because queries run in parallel across multiple compute nodes, the errors that you get are not in sequential order. For example, if your query fails with an error in row 6,234, it does not mean that all rows before row 6,234 were successfully processed. The next run of the query might show an error in a different row.

Data Access Tiers

You can specify one of the following data access tiers when querying an archived object:

- **Expedited** – Allows you to quickly access your data when occasional urgent requests for a subset of archives are required. For all but the largest archived object (250 MB+), data accessed using Expedited retrievals are typically made available within 1–5 minutes. There are two types of Expedited data access: On-Demand and Provisioned. On-Demand requests are similar to EC2 On-Demand instances and are available most of the time. Provisioned requests are guaranteed to be available when you need them. For more information, see [Provisioned Capacity \(p. 256\)](#).
- **Standard** – Allows you to access any of your archived objects within several hours. Standard retrievals typically finish within 3–5 hours. This is the default tier.
- **Bulk** – The lowest-cost data access option in Glacier, enabling you to retrieve large amounts, even petabytes, of data inexpensively in a day. Bulk access typically finishes within 5–12 hours.

To make an Expedited, Standard, or Bulk request, set the `Tier` request element in the [POST Object restore](#) REST API request to the option you want, or the equivalent in the AWS CLI or AWS SDKs. For Expedited access, there is no need to designate whether an expedited retrieval is On-Demand or Provisioned. If you purchased provisioned capacity, all Expedited retrievals are automatically served through your provisioned capacity. For information about tier pricing, see [Glacier Pricing](#).

Provisioned Capacity

Provisioned capacity guarantees that your retrieval capacity for expedited retrievals is available when you need it. Each unit of capacity ensures that at least three expedited retrievals can be performed every five minutes and provides up to 150 MB/s of retrieval throughput.

You should purchase provisioned retrieval capacity if your workload requires highly reliable and predictable access to a subset of your data in minutes. Without provisioned capacity, `Expedited` retrievals are accepted, except for rare situations of unusually high demand. However, if you require access to `Expedited` retrievals under all circumstances, you must purchase provisioned retrieval capacity. You can purchase provisioned capacity using the Amazon S3 console, the Glacier console, the [Purchase Provisioned Capacity](#) REST API, the AWS SDKs, or the AWS CLI. For provisioned capacity pricing information, see the [Glacier Pricing](#).

More Info

- [POST Object restore](#) in the *Amazon Simple Storage Service API Reference*
- [SQL Reference for Amazon S3 Select and Glacier Select](#) in the *Amazon Simple Storage Service Developer Guide*

Amazon S3 Analytics – Storage Class Analysis

By using Amazon S3 analytics *storage class analysis* you can analyze storage access patterns to help you decide when to transition the right data to the right storage class. This new Amazon S3 analytics feature observes data access patterns to help you determine when to transition less frequently accessed STANDARD storage to the STANDARD_IA (IA, for infrequent access) storage class. For more information about storage classes, see [Amazon S3 Storage Classes \(p. 103\)](#).

After storage class analysis observes the infrequent access patterns of a filtered set of data over a period of time, you can use the analysis results to help you improve your lifecycle policies. You can configure storage class analysis to analyze all the objects in a bucket. Or, you can configure filters to group objects together for analysis by common prefix (that is, objects that have names that begin with a common string), by object tags, or by both prefix and tags. You'll most likely find that filtering by object groups is the best way to benefit from storage class analysis.

Important

Storage class analysis does not give recommendations for transitions to the ONEZONE_IA or GLACIER storage classes.

You can have multiple storage class analysis filters per bucket, up to 1,000, and will receive a separate analysis for each filter. Multiple filter configurations allow you analyze specific groups of objects to improve your lifecycle policies that transition objects to STANDARD_IA.

Storage class analysis shows storage usage visualizations in the Amazon S3 console that are updated daily. The storage usage data can also be exported daily to a file in an S3 bucket. You can open the exported usage report file in a spreadsheet application or use it with the business intelligence tools of your choice such as Amazon QuickSight.

Topics

- [How Do I Set Up Storage Class Analysis? \(p. 257\)](#)
- [How Do I Use Storage Class Analysis? \(p. 258\)](#)
- [How Can I Export Storage Class Analysis Data? \(p. 260\)](#)
- [Amazon S3 Analytics REST APIs \(p. 262\)](#)

How Do I Set Up Storage Class Analysis?

You set up storage class analysis by configuring what object data you want to analyze. You can configure storage class analysis to do the following:

- **Analyze the entire contents of a bucket.**

You'll receive an analysis for all the objects in the bucket.

- **Analyze objects grouped together by prefix and tags.**

You can configure filters that group objects together for analysis by prefix, or by object tags, or by a combination of prefix and tags. You receive a separate analysis for each filter you configure. You can have multiple filter configurations per bucket, up to 1,000.

- **Export analysis data.**

When you configure storage class analysis for a bucket or filter, you can choose to have the analysis data exported to a file each day. The analysis for the day is added to the file to form a historic analysis log for the configured filter. The file is updated daily at the destination of your choice. When selecting data to export, you specify a destination bucket and optional destination prefix where the file is written.

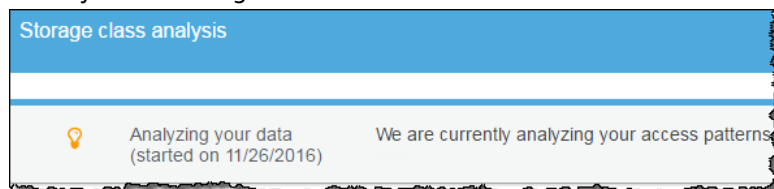
You can use the Amazon S3 console, the REST API, or the AWS CLI or AWS SDKs to configure storage class analysis.

- For information about how to configure storage class analysis in the Amazon S3 console, see [How Do I Configure Storage Class Analysis?](#).
- To use the Amazon S3 API, use the [PutBucketAnalyticsConfiguration](#) REST API, or the equivalent, from the AWS CLI or AWS SDKs.

How Do I Use Storage Class Analysis?

You use storage class analysis to observe your data access patterns over time to gather information to help you improve the lifecycle management of your STANDARD_IA storage. After you configure a filter, you'll start seeing data analysis based on the filter in the Amazon S3 console in 24 to 48 hours. However, storage class analysis observes the access patterns of a filtered data set for 30 days or longer to gather information for analysis before giving a result. The analysis continues to run after the initial result and updates the result as the access patterns change

When you first configure a filter the Amazon S3 console shows a message similar to the following.



Storage class analysis observes the access patterns of a filtered object data set for 30 days or longer to gather enough information for the analysis. After storage class analysis has gathered sufficient information, you'll see a message in the Amazon S3 console similar to the following.



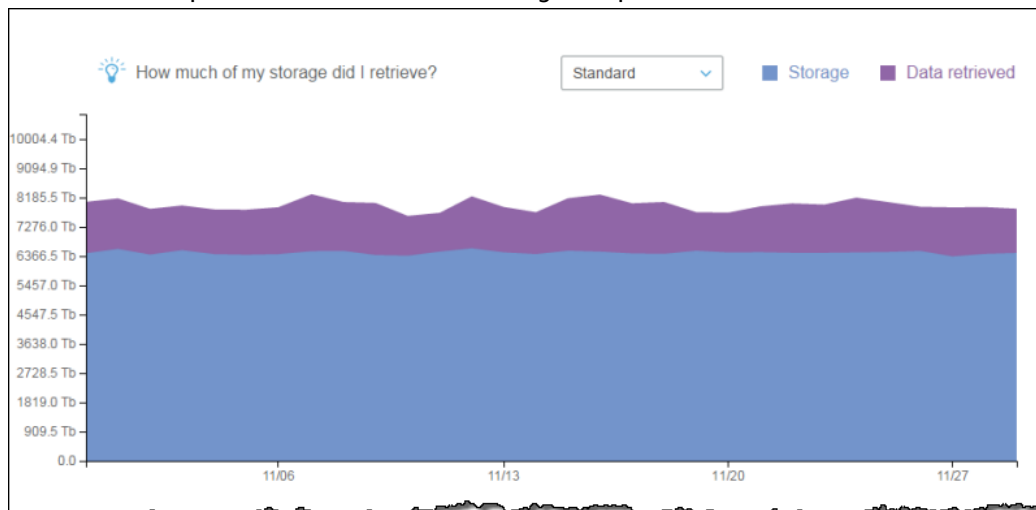
When performing the analysis for infrequently accessed objects storage class analysis looks at the filtered set of objects grouped together based on age since they were uploaded to Amazon S3. Storage class analysis determines if the age group is infrequently accessed by looking at the following factors for the filtered data set:

- Objects in the STANDARD storage class that are larger than 128K.
- How much average total storage you have per age group.
- Average number of bytes transferred out (not frequency) per age group.
- Analytics export data only includes requests with data relevant to storage class analysis. This might cause differences in the number of requests, and the total upload and request bytes compared to what are shown in storage metrics or tracked by your own internal systems.

- Failed GET and PUT requests are not counted for the analysis. However, you will see failed requests in storage metrics.

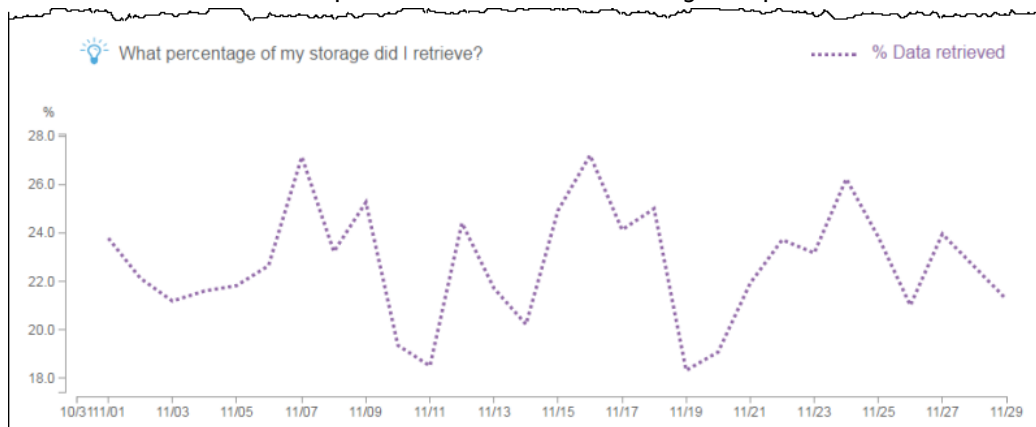
How Much of My Storage did I Retrieve?

The Amazon S3 console graphs how much of the storage in the filtered data set has been retrieved for the observation period as shown in the following example.



What Percentage of My Storage did I Retrieve?

The Amazon S3 console also graphs what percentage of the storage in the filtered data set has been retrieved for the observation period as shown in the following example.



As stated earlier in this topic, when you are performing the analysis for infrequently accessed objects, storage class analysis looks at the filtered set of objects grouped together based on the age since they were uploaded to Amazon S3. The storage class analysis uses the following predefined object age groups:

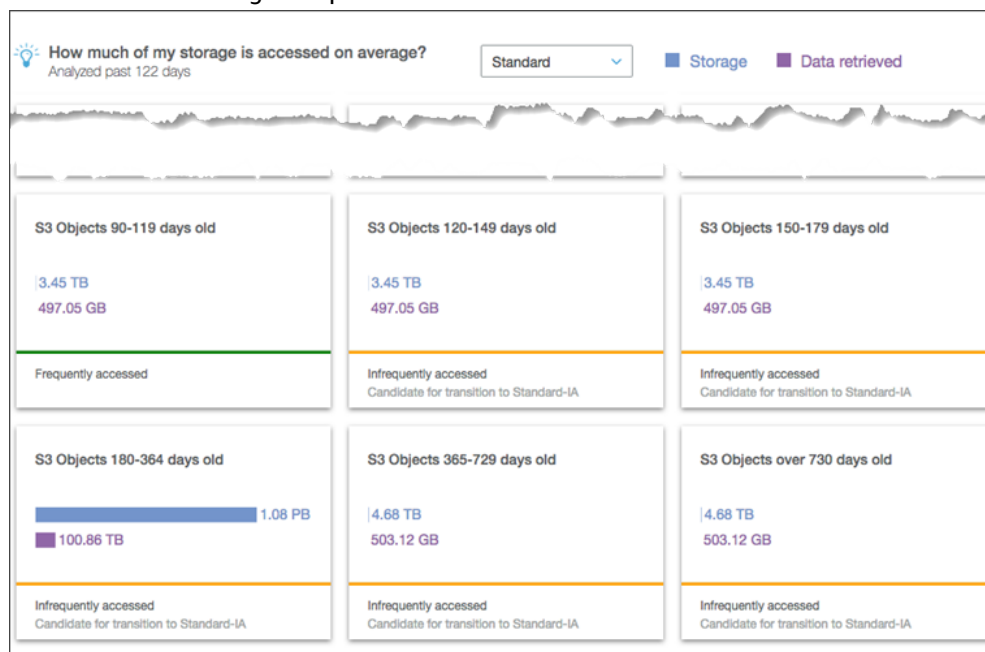
- Amazon S3 Objects less than 15 days old
- Amazon S3 Objects 15-29 days old
- Amazon S3 Objects 30-44 days old
- Amazon S3 Objects 45-59 days old
- Amazon S3 Objects 60-74 days old

- Amazon S3 Objects 75-89 days old
- Amazon S3 Objects 90-119 days old
- Amazon S3 Objects 120-149 days old
- Amazon S3 Objects 150-179 days old
- Amazon S3 Objects 180-364 days old
- Amazon S3 Objects 365-729 days old
- Amazon S3 Objects 730 days and older

Usually it takes about 30 days of observing access patterns to gather enough information for an analysis result. It might take longer than 30 days, depending on the unique access pattern of your data. However, after you configure a filter you'll start seeing data analysis based on the filter in the Amazon S3 console in 24 to 48 hours. You can see analysis on a daily basis of object access broken down by object age group in the Amazon S3 console.

How Much of My Storage is Infrequently Accessed?

The Amazon S3 console shows the access patterns grouped by the predefined object age groups as shown in the following example.



The **Frequently accessed** or **Infrequently accessed** text shown at the bottom of each age group is based on the same logic as the lifecycle policy recommendation being prepared. After a recommended age for a lifecycle policy is ready (RecommendedObjectAge), all of the age tiers younger than that recommended age are marked as infrequently accessed, regardless of the current cumulative access ratio. This text is meant as a visual aid to help you in the lifecycle creation process.

How Can I Export Storage Class Analysis Data?

You can choose to have storage class analysis export analysis reports to a comma-separated values (CSV) flat file. Reports are updated daily and are based on the object age group filters you configure. When using the Amazon S3 console you can choose the export report option when you create a filter. When

selecting data export you specify a destination bucket and optional destination prefix where the file is written. You can export the data to a destination bucket in a different account. The destination bucket must be in the same region as the bucket that you configure to be analyzed.

You must create a bucket policy on the destination bucket to grant permissions to Amazon S3 to verify what AWS account owns the bucket and to write objects to the bucket in the defined location. For an example policy, see [Granting Permissions for Amazon S3 Inventory and Amazon S3 Analytics \(p. 377\)](#).

After you configure storage class analysis reports, you start getting the exported report daily after 24 hours. After that, Amazon S3 continues monitoring and providing daily exports.

You can open the CSV file in a spreadsheet application or import the file into other applications like [Amazon QuickSight](#). For information on using Amazon S3 files with Amazon QuickSight, see [Create a Data Set Using Amazon S3 Files](#) in the *Amazon QuickSight User Guide*.

Data in the exported file is sorted by date within object age group as shown in following examples. If the storage class is STANDARD the row also contains data for the columns `ObjectAgeForSIATransition` and `RecommendedObjectAgeForSIATransition`.

Date	ConfigId	Filter	StorageClass	ObjectAge	ObjectCount	DataInloaded_MB	Storage_MB	DataRetrieved_MB	GetRequestCount	CumulativeAccessRatio	ObjectAgeForSIATransition	RecommendedObjectAgeForSIATransition
11/26/17	SalesMaterial	SalesMaterial	STANDARD	000-014	39376.428	3610.516	100409232	106131482	17724.24	1.056989281		
11/25/17	SalesMaterial	SalesMaterial	STANDARD	000-014	37904.412	3411.772	90017538.84	100602072	18068.4	1.11758301		
11/24/17	SalesMaterial	SalesMaterial	STANDARD	000-014	39744.432	3478.02	87888239.2	108298204.6	18584.64	1.232226355		
11/23/17	SalesMaterial	SalesMaterial	STANDARD	000-014	40480.44	3544.268	91903507.12	111984477.9	18928.8	1.218500593		
11/22/17	SalesMaterial	SalesMaterial	STANDARD	000-014	40112.436	3544.268	94450993.4	109851751.9	17724.24	1.12250433		
11/21/17	SalesMaterial	SalesMaterial	STANDARD	000-014	40480.44	3345.524	103598373.7	115342424	18412.56	1.11361359		
11/20/17	SalesMaterial	SalesMaterial	STANDARD	000-014	39008.424	3411.772	92668826.16	114416707.1	18756.72	1.234683893		
11/19/17	SalesMaterial	SalesMaterial	STANDARD	000-014	40480.44	3444.896	90254194.28	116396152.9	18068.4	1.28964813		
11/18/17	SalesMaterial	SalesMaterial	STANDARD	000-014	39744.432	3444.896	88724282.88	110218385.6	17724.24	1.242257271		
11/17/17	SalesMaterial	SalesMaterial	STANDARD	000-014	37904.412	3444.896	89554277.32	102845839	18584.64	1.148419061		
11/16/17	SalesMaterial	SalesMaterial	STANDARD	000-014	40112.436	3444.896	91362383.92	116521794.3	18928.8	1.275381802		
11/15/17	SalesMaterial	SalesMaterial	STANDARD	000-014	40112.436	3378.648	87790610.56	113237336.7	18756.72	1.290739184		
11/14/17	SalesMaterial	SalesMaterial	STANDARD	000-014	39744.432	3478.02	96131832.8	110526562.8	17896.32	1.462739473		

11/25/17	SalesMaterial	SalesMaterial	STANDARD	015-029	56856.618		5117.658	135026308.3	150903108.1	27102.6	1.11758301	
11/24/17	SalesMaterial	SalesMaterial	STANDARD	015-029	59616.648		5217.03	131832358.8	162447307	27876.96	1.232226355	
11/23/17	SalesMaterial	SalesMaterial	STANDARD	015-029	60720.66		5316.402	137855260.7	167976716.9	28393.2	1.218500593	
11/22/17	SalesMaterial	SalesMaterial	STANDARD	015-029	60168.654		5316.402	141607640.1	158777627.8	26586.36	1.121250433	
11/21/17	SalesMaterial	SalesMaterial	STANDARD	015-029	60720.66		5018.286	155397557.6	173013635.9	27618.84	1.11361359	
11/20/17	SalesMaterial	SalesMaterial	STANDARD	015-029	58512.636		5117.658	139003239.2	171625060.6	28135.08	1.234683893	
11/19/17	SalesMaterial	SalesMaterial	STANDARD	015-029	60720.66		5167.344	135381291.4	174594229.3	27102.6	1.28964813	
11/18/17	SalesMaterial	SalesMaterial	STANDARD	015-029	59616.648		5167.344	133086424.3	165327578.3	26586.36	1.242257271	
11/17/17	SalesMaterial	SalesMaterial	STANDARD	015-029	56856.618		5167.344	134331416	154268758.5	27876.96	1.148419061	
11/16/17	SalesMaterial	SalesMaterial	STANDARD	015-029	60168.654		5167.344	137043425.9	174782691.5	28393.2	1.275381802	
11/15/17	SalesMaterial	SalesMaterial	STANDARD	015-029	60168.654		5067.972	131595915.8	169856005	28135.08	1.290739184	
11/14/17	SalesMaterial	SalesMaterial	STANDARD	015-029	59616.648		5217.03	144197749.2	165789844.1	26844.48	1.149739473	
11/13/17	SalesMaterial	SalesMaterial	STANDARD	015-029	59616.648		5167.344	133083383.3	156404070.8	27618.84	1.181358667	

At the end of the report the object age group is ALL. The ALL rows contain cumulative totals for all the age groups for that day as shown in the following example.

11/25/17	SalesMaterial	SalesMaterial	STANDARD	ALL	777408.45	89229.16	3237567090	2066259214	363519	0.034	090-119	090-119
11/24/17	SalesMaterial	SalesMaterial	STANDARD	ALL	787528.56	69726.02	3222158279	2031486122	362228.4	0.034	090-119	090-119
11/23/17	SalesMaterial	SalesMaterial	STANDARD	ALL	775568.43	69643.21	3224228797	2143607314	365670	0.034	090-119	090-119
11/22/17	SalesMaterial	SalesMaterial	STANDARD	ALL	785688.54	70471.31	3268098456	2093598765	357926.4	0.034	090-119	090-119
11/21/17	SalesMaterial	SalesMaterial	STANDARD	ALL	772808.4	68069.82	3260462574	2152402776	366960.6	0.034	090-119	090-119
11/20/17	SalesMaterial	SalesMaterial	STANDARD	ALL	775568.43	69311.97	3235392872	2135278200	363088.8	0.034	090-119	090-119
11/19/17	SalesMaterial	SalesMaterial	STANDARD	ALL	793048.62	68566.68	3191517795	2156592855	361368	0.034	090-119	090-119
11/18/17	SalesMaterial	SalesMaterial	STANDARD	ALL	784768.53	69808.83	3241761620	2158880455	357926.4	0.034	090-119	090-119
11/17/17	SalesMaterial	SalesMaterial	STANDARD	ALL	782008.5	71050.98	3143115658	2130892653	372983.4	0.034	090-119	090-119
11/16/17	SalesMaterial	SalesMaterial	STANDARD	ALL	773728.41	70636.93	3237654725	2145622446	369111.6	0.034	090-119	090-119
11/15/17	SalesMaterial	SalesMaterial	STANDARD	ALL	773728.41	69477.59	3131717743	2091869246	369541.8	0.034	090-119	090-119
11/14/17	SalesMaterial	SalesMaterial	STANDARD	ALL	789368.58	70388.5	3169870122	2171336445	358356.6	0.034	090-119	090-119
11/13/17	SalesMaterial	SalesMaterial	STANDARD	ALL	767288.34	69643.21	3199668074	2128513319	367390.8	0.034	090-119	090-119
11/12/17	SalesMaterial	SalesMaterial	STANDARD	ALL	767288.34	68649.49	3135649104	2041228760	358356.6	0.034	090-119	090-119
11/11/17	SalesMaterial	SalesMaterial	STANDARD	ALL	774648.42	68897.92	3132710428	2154020987	356635.8	0.034	090-119	090-119
11/10/17	SalesMaterial	SalesMaterial	STANDARD	ALL	774648.42	69311.97	3181716041	2175053920	369541.8	0.034	090-119	090-119
11/9/17	SalesMaterial	SalesMaterial	STANDARD	ALL	771888.39	69643.21	3118609765	2197544222	364809.6	0.034	090-119	090-119
11/8/17	SalesMaterial	SalesMaterial	STANDARD	ALL	781088.49	70885.36	3136378996	2162511633	360507.6	0.034	090-119	090-119
11/7/17	SalesMaterial	SalesMaterial	STANDARD	ALL	783848.52	68649.49	3166312643	2019414755	355345.2	0.034	090-119	090-119
11/6/17	SalesMaterial	SalesMaterial	STANDARD	ALL	771888.39	69146.35	3087018554	2210161640	363519	0.034	090-119	090-119
11/5/17	SalesMaterial	SalesMaterial	STANDARD	ALL	770968.38	68235.44	3098188075	2177492429	360077.4	0.034	090-119	090-119
11/4/17	SalesMaterial	SalesMaterial	STANDARD	ALL	783848.52	69974.45	3115637244	2208645234	367390.8	0.034	090-119	090-119
11/3/17	SalesMaterial	SalesMaterial	STANDARD	ALL	770048.37	69311.97	3125766832	2195447440	354054.6	0.034	090-119	090-119
11/2/17	SalesMaterial	SalesMaterial	STANDARD	ALL	772808.4	72280.8	314641159	2237000000	360507.6	0.034	090-119	090-119

The next section describes the columns used in the report.

Exported File Layout

The following table describe the layout of the exported file.

Amazon S3 Analytics REST APIs

The following are the REST operations used for storage inventory.

- [DELETE Bucket analytics configuration](#)
- [GET Bucket analytics configuration](#)
- [List Bucket Analytics Configuration](#)
- [PUT Bucket analytics configuration](#)

Amazon S3 Security

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon S3, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation will help you understand how to apply the shared responsibility model when using Amazon S3. The following topics show you how to configure Amazon S3 to meet your security and compliance objectives. You'll also learn how to use other AWS services that can help you monitor and secure your Amazon S3 resources.

Topics

- [Data Protection in Amazon S3 \(p. 263\)](#)
- [Identity and Access Management in Amazon S3 \(p. 301\)](#)
- [Logging and Monitoring in Amazon S3 \(p. 421\)](#)
- [Compliance Validation for Amazon S3 \(p. 422\)](#)
- [Resilience in Amazon S3 \(p. 431\)](#)
- [Infrastructure Security in Amazon S3 \(p. 461\)](#)
- [Configuration and Vulnerability Analysis in Amazon S3 \(p. 462\)](#)
- [Security Best Practices for Amazon S3 \(p. 463\)](#)

Data Protection in Amazon S3

Amazon S3 provides a highly durable storage infrastructure designed for mission-critical and primary data storage. Objects are redundantly stored on multiple devices across multiple facilities in an Amazon S3 Region. To help better ensure data durability, Amazon S3 `PUT` and `PUT Object Copy` operations synchronously store your data across multiple facilities. After the objects are stored, Amazon S3 maintains their durability by quickly detecting and repairing any lost redundancy.

Amazon S3 standard storage offers the following features:

- Backed with the [Amazon S3 Service Level Agreement](#)
- Designed to provide 99.999999999% durability and 99.99% availability of objects over a given year

- Designed to sustain the concurrent loss of data in two facilities

Amazon S3 further protects your data using versioning. You can use versioning to preserve, retrieve, and restore every version of every object that is stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. By default, requests retrieve the most recently written version. You can retrieve older versions of an object by specifying a version of the object in a request.

The following security best practices also address data protection in Amazon S3:

- [Implement server-side encryption](#)
- [Enforce encryption of data in transit](#)
- [Consider using Amazon Macie with Amazon S3](#)
- [Identify and audit all your Amazon S3 buckets](#)
- [Monitor AWS security advisories](#)

Internetwork Traffic Privacy

This topic describes how Amazon S3 secures connections from the service to other locations.

Traffic Between Service and On-Premises Clients and Applications

You have two connectivity options between your private network and AWS:

- An AWS Site-to-Site VPN connection. For more information, see [What is AWS Site-to-Site VPN?](#)
- An AWS Direct Connect connection. For more information, see [What is AWS Direct Connect?](#)

Access to Amazon S3 via the network is through AWS published APIs. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2 or above. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Most modern systems such as Java 7 and later support these modes. Additionally, you must sign requests using an access key ID and a secret access key that are associated with an IAM principal, or you can use the [AWS Security Token Service \(STS\)](#) to generate temporary security credentials to sign requests.

Traffic Between AWS Resources in the Same Region

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon S3 is a logical entity within a VPC that allows connectivity only to Amazon S3. The Amazon VPC routes requests to Amazon S3 and routes responses back to the VPC. For more information, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For example bucket policies that you can use to control S3 bucket access from VPC endpoints, see [Example Bucket Policies for VPC Endpoints for Amazon S3](#) (p. 378).

Protecting Data Using Encryption

Data protection refers to protecting data while in-transit (as it travels to and from Amazon S3) and at rest (while it is stored on disks in Amazon S3 data centers). You can protect data in transit using Secure Sockets Layer (SSL) or client-side encryption. You have the following options for protecting data at rest in Amazon S3:

- **Server-Side Encryption** – Request Amazon S3 to encrypt your object before saving it on disks in its data centers and then decrypt it when you download the objects.
- **Client-Side Encryption** – Encrypt data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and related tools.

Protecting Data Using Server-Side Encryption

Server-side encryption is about data encryption at rest—that is, Amazon S3 encrypts your data at the object level as it writes it to disks in its data centers and decrypts it for you when you access it. As long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects. For example, if you share your objects using a presigned URL, that URL works the same way for both encrypted and unencrypted objects.

Note

You can't apply different types of server-side encryption to the same object simultaneously.

You have three mutually exclusive options depending on how you choose to manage the encryption keys:

- **Use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)** – Each object is encrypted with a unique key. As an additional safeguard, it encrypts the key itself with a master key that it regularly rotates. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256), to encrypt your data. For more information, see [Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys \(SSE-S3\)](#) (p. 270).
- **Use Server-Side Encryption with Keys Stored in AWS KMS (SSE-KMS)** – Similar to SSE-S3, but with some additional benefits along with some additional charges for using this service. There are separate permissions for the use of an envelope key (that is, a key that protects your data's encryption key) that provides added protection against unauthorized access of your objects in Amazon S3. SSE-KMS also provides you with an audit trail of when your key was used and by whom. Additionally, you have the option to create and manage encryption keys yourself, or use a default key that is unique to you, the service you're using, and the Region you're working in. For more information, see [Protecting Data Using Server-Side Encryption with keys stored in AWS KMS\(SSE-KMS\)](#) (p. 265).
- **Use Server-Side Encryption with Customer-Provided Keys (SSE-C)** – You manage the encryption keys and Amazon S3 manages the encryption, as it writes to disks, and decryption, when you access your objects. For more information, see [Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\)](#) (p. 279).

Note

When you list objects in your bucket, the list API returns a list of all objects, regardless of whether they are encrypted.

Protecting Data Using Server-Side Encryption with keys stored in AWS KMS(SSE-KMS)

Server-side encryption is about protecting data at rest. AWS Key Management Service (AWS KMS) is a service that combines secure, highly available hardware and software to provide a key management system scaled for the cloud. AWS KMS uses customer master keys (CMKs) to encrypt your Amazon S3 objects. You use AWS KMS via the [AWS Management Console](#) or [AWS KMS APIs](#) to centrally create encryption keys, define the policies that control how keys can be used, and audit key usage to prove they are being used correctly. You can use these keys to protect your data in Amazon S3 buckets.

The first time you add an SSE-KMS–encrypted object to a bucket in a Region, a default CMK is created for you automatically. This key is used for SSE-KMS encryption unless you select a CMK that you created separately using AWS Key Management Service. Creating your own CMK gives you more flexibility, including the ability to create, rotate, disable, and define access controls, and to audit the encryption keys used to protect your data.

For more information, see [What is AWS Key Management Service?](#) in the *AWS Key Management Service Developer Guide*. If you use AWS KMS, there are additional charges for using AWS KMS keys. For more information, see [AWS Key Management Service Pricing](#).

Note

- If you are uploading or accessing objects encrypted by SSE-KMS, you need to use AWS Signature Version 4 for added security. For more information on how to do this using an AWS SDK, see [Specifying Signature Version in Request Authentication](#).
- When using SSE-KMS encryption with an S3 bucket, the KMS key must be in the same Region as the bucket.

The highlights of SSE-KMS are:

- You can choose to create and manage encryption keys yourself, or you can choose to use your default service key uniquely generated on a customer by service by Region level.
- The ETag in the response is not the MD5 of the object data.
- The data keys used to encrypt your data are also encrypted and stored alongside the data they protect.
- Auditable master keys can be created, rotated, and disabled from the AWS KMS console.
- The security controls in AWS KMS can help you meet encryption-related compliance requirements.

Amazon S3 supports bucket policies that you can use if you require server-side encryption for all objects that are stored in your bucket. For example, the following bucket policy denies upload object (`s3:PutObject`) permission to everyone if the request does not include the `x-amz-server-side-encryption` header requesting server-side encryption with SSE-KMS.

```
{
  "Version": "2012-10-17",
  "Id": "PutObjPolicy",
  "Statement": [{
    "Sid": "DenyUnEncryptedObjectUploads",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::YourBucket/*",
    "Condition": {
      "StringNotEquals": {
        "s3:x-amz-server-side-encryption": "aws:kms"
      }
    }
  }]
}
```

Amazon S3 also supports the `s3:x-amz-server-side-encryption-aws-kms-key-id` condition key, which you can use to require a specific KMS key for object encryption. The KMS key you specify in the policy must use the `"arn:aws:kms:region:acct-id:key/key-id"` format.

Note

When you upload an object, you can specify the KMS key using the `x-amz-server-side-encryption-aws-kms-key-id` header. If the header is not present in the request, Amazon S3 assumes the default KMS key. Regardless, the KMS key ID that Amazon S3 uses for object encryption must match the KMS key ID in the policy, otherwise Amazon S3 denies the request.

Important

All GET and PUT requests for an object protected by AWS KMS will fail if they are not made via SSL or if they are not made using SigV4.

SSE-KMS encrypts only the object data. Any object metadata is not encrypted.

Using AWS Key Management Service in the Amazon S3 Console

For more information about using the Amazon S3 console with encryption keys stored in AWS KMS, see [How Do I Upload Files and Folders to an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

API Support for AWS Key Management Service in Amazon S3

The object creation REST APIs (see [Specifying the AWS Key Management Service in Amazon S3 Using the REST API \(p. 270\)](#)) provide a request header, `x-amz-server-side-encryption` that you can use to request SSE-KMS with the value of `aws:kms`. There's also `x-amz-server-side-encryption-aws-kms-key-id`, which specifies the ID of the AWS KMS master encryption key that was used for the object. The Amazon S3 API also supports encryption context, with the `x-amz-server-side-encryption-context` header.

The encryption context can be any value that you want, provided that the header adheres to the Base64-encoded JSON format. However, because the encryption context is not encrypted and because it is logged if AWS CloudTrail logging is turned on, the encryption context should not include sensitive information. We further recommend that your context describe the data being encrypted or decrypted so that you can better understand the CloudTrail events produced by AWS KMS. For more information, see [Encryption Context](#) in the *AWS Key Management Service Developer Guide*.

Also, Amazon S3 may append a predefined key of `aws:s3:arn` with the value equal to the object's ARN for the encryption context that you provide. This only happens if the key `aws:s3:arn` is not already in the encryption context that you provided, in which case this predefined key is appended when Amazon S3 processes your Put requests. If this `aws:s3:arn` key is already present in your encryption context, the key is not appended a second time to your encryption context.

Having this predefined key as a part of your encryption context means that you can track relevant requests in CloudTrail, so you'll always be able to see which Amazon S3 object's ARN was used with which encryption key. In addition, this predefined key as a part of your encryption context guarantees that the encryption context is not identical between different Amazon S3 objects, which provides additional security for your objects. Your full encryption context will be validated to have the value equal to the object's ARN.

The following Amazon S3 APIs support these request headers.

- PUT operation — When uploading data using the PUT API (see [PUT Object](#)), you can specify these request headers.
- Initiate Multipart Upload — When uploading large objects using the multipart upload API, you can specify these headers. You specify these headers in the initiate request (see [Initiate Multipart Upload](#)).
- POST operation — When using a POST operation to upload an object (see [POST Object](#)), instead of the request headers, you provide the same information in the form fields.
- COPY operation — When you copy an object (see [PUT Object - Copy](#)), you have both a source object and a target object. When you pass SSE-KMS headers with the COPY operation, they will be applied only to the target object.

The AWS SDKs also provide wrapper APIs for you to request SSE-KMS with Amazon S3.

Specifying the AWS Key Management Service in Amazon S3 Using the AWS SDKs

When using AWS SDKs, you can request Amazon S3 to use AWS Key Management Service (AWS KMS)–managed encryption keys. This section provides examples of using the AWS SDKs for Java and .NET. For information about other SDKs, go to [Sample Code and Libraries](#).

AWS SDK for Java

This section explains various Amazon S3 operations using the AWS SDK for Java and how you use the AWS KMS–managed encryption keys.

Put Operation

When uploading an object using the AWS SDK for Java, you can request Amazon S3 to use an AWS KMS–managed encryption key by adding the `SSEAwsKeyManagementParams` property as shown in the following request:

```
PutObjectRequest putRequest = new PutObjectRequest(bucketName,
    keyName, file).withSSEAwsKeyManagementParams(new SSEAwsKeyManagementParams());
```

In this case, Amazon S3 uses the default master key (see [Protecting Data Using Server-Side Encryption with keys stored in KMS\(SSE-KMS\) \(p. 265\)](#)). You can optionally create your own key and specify that in the request.

```
PutObjectRequest putRequest = new PutObjectRequest(bucketName,
    keyName, file).withSSEAwsKeyManagementParams(new SSEAwsKeyManagementParams(keyID));
```

For more information about creating keys, see [Programming the AWS KMS API](#) in the *AWS Key Management Service Developer Guide*.

For working code examples of uploading an object, see the following topics. You will need to update those code examples and provide encryption information as shown in the preceding code fragment.

- For uploading an object in a single operation, see [Upload an Object Using the AWS SDK for Java \(p. 170\)](#).
- For a multipart upload, see the following topics:
 - Using high-level multipart upload API, see [Upload a File \(p. 182\)](#).
 - If you are using the low-level multipart upload API, see [Upload a File \(p. 186\)](#).

Copy Operation

When copying objects, you add the same request properties (`ServerSideEncryptionMethod` and `ServerSideEncryptionKeyManagementServiceKeyId`) to request Amazon S3 to use an AWS KMS–managed encryption key. For more information about copying objects, see [Copying Objects \(p. 210\)](#).

Presigned URLs

When creating a presigned URL for an object encrypted using an AWS KMS–managed encryption key, you must explicitly specify Signature Version 4:

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setSignerOverride("AWSS3V4SignerType");
AmazonS3Client s3client = new AmazonS3Client(
    new ProfileCredentialsProvider(), clientConfiguration);
...
```

For a code example, see [Generate a presigned Object URL Using the AWS SDK for Java \(p. 167\)](#).

AWS SDK for .NET

This section explains various Amazon S3 operations using the AWS SDK for .NET and how you use the AWS KMS–managed encryption keys.

Put Operation

When uploading an object using the AWS SDK for .NET, you can request Amazon S3 to use an AWS KMS–managed encryption key by adding the `ServerSideEncryptionMethod` property as shown in the following request.

```
PutObjectRequest putRequest = new PutObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    // other properties.
    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AWSEKMS
};
```

In this case, Amazon S3 uses the default master key (see [Protecting Data Using Server-Side Encryption with keys stored in AWS KMS\(SSE-KMS\) \(p. 265\)](#)). You can optionally create your own key and specify that in the request.

```
PutObjectRequest putRequest1 = new PutObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    // other properties.
    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AWSEKMS,
    ServerSideEncryptionKeyManagementServiceKeyId = keyId
};
```

For more information about creating keys, see [Programming the AWS KMS API](#) in the *AWS Key Management Service Developer Guide*.

For working code examples of uploading an object, see the following topics. You will need to update these code examples and provide encryption information as shown in the preceding code fragment.

- For uploading an object in a single operation, see [Upload an Object Using the AWS SDK for .NET \(p. 171\)](#).
- For multipart upload see the following topics:
 - Using high-level multipart upload API, see [Upload a File to an S3 Bucket Using the AWS SDK for .NET \(High-Level API\) \(p. 191\)](#).
 - Using low-level multipart upload API, see [Upload a File to an S3 Bucket Using the AWS SDK for .NET \(Low-Level API\) \(p. 197\)](#).

Copy Operation

When copying objects, you add the same request properties (`ServerSideEncryptionMethod` and `ServerSideEncryptionKeyManagementServiceKeyId`) to request Amazon S3 to use an AWS KMS–managed encryption key. For more information about copying objects, see [Copying Objects \(p. 210\)](#).

Presigned URLs

When creating a presigned URL for an object encrypted using an AWS KMS–managed encryption key, you must explicitly specify Signature Version 4:

```
AWSConfigs.S3Config.UseSignatureVersion4 = true;
```

For a code example, see [Generate a Presigned Object URL Using AWS SDK for .NET \(p. 168\)](#).

Specifying the AWS Key Management Service in Amazon S3 Using the REST API

At the time of object creation—that is, when you are uploading a new object or making a copy of an existing object—you can specify the use of server-side encryption with AWS KMS-managed encryption keys (SSE-KMS) to encrypt your data by adding the `x-amz-server-side-encryption` header to the request. Set the value of the header to the encryption algorithm `aws:kms`. Amazon S3 confirms that your object is stored using SSE-KMS by returning the response header `x-amz-server-side-encryption`.

The following REST upload APIs accept the `x-amz-server-side-encryption` request header.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)

When uploading large objects using the multipart upload API, you can specify SSE-KMS by adding the `x-amz-server-side-encryption` header to the Initiate Multipart Upload request with the value of `aws:kms`. When copying an existing object, regardless of whether the source object is encrypted or not, the destination object is not encrypted unless you explicitly request server-side encryption.

The response headers of the following REST APIs return the `x-amz-server-side-encryption` header when an object is stored using server-side encryption.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part - Copy](#)
- [Complete Multipart Upload](#)
- [Get Object](#)
- [Head Object](#)

Note

Encryption request headers should not be sent for `GET` requests and `HEAD` requests if your object uses SSE-KMS or you'll get an HTTP 400 BadRequest error.

Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3)

Server-side encryption protects data at rest. Amazon S3 encrypts each object with a unique key. As an additional safeguard, it encrypts the key itself with a master key that it rotates regularly. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256), to encrypt your data.

If you need server-side encryption for all of the objects that are stored in a bucket, use a bucket policy. For example, the following bucket policy denies permissions to upload an object unless the request includes the `x-amz-server-side-encryption` header to request server-side encryption:

```
{
  "Version": "2012-10-17",
  "Id": "PutObjPolicy",
  "Statement": [
    {
```



```

    "Sid": "DenyIncorrectEncryptionHeader",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::YourBucket/*",
    "Condition": {
      "StringNotEquals": {
        "s3:x-amz-server-side-encryption": "AES256"
      }
    }
  },
  {
    "Sid": "DenyUnEncryptedObjectUploads",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::YourBucket/*",
    "Condition": {
      "Null": {
        "s3:x-amz-server-side-encryption": "true"
      }
    }
  }
]
}

```

Note

- Server-side encryption encrypts only the object data, not object metadata.
- You need the `kms:Decrypt` permission when you upload or download an Amazon S3 object encrypted with an AWS KMS key, and that is in addition to `kms:ReEncrypt`, `kms:GenerateDataKey`, and `kms:DescribeKey`. For more information, see [Failures uploading a large file to Amazon S3 with encryption using an AWS KMS key](#).

API Support for Server-Side Encryption

To request server-side encryption using the object creation REST APIs, provide the `x-amz-server-side-encryption` request header. For information about the REST APIs, see [Specifying Server-Side Encryption Using the REST API](#) (p. 278).

The following Amazon S3 APIs support this header:

- PUT operations—Specify the request header when uploading data using the PUT API. For more information, see [PUT Object](#).
- Initiate Multipart Upload—Specify the header in the initiate request when uploading large objects using the multipart upload API. For more information, see [Initiate Multipart Upload](#).
- COPY operations—When you copy an object, you have both a source object and a target object. For more information, see [PUT Object - Copy](#).

Note

When using a POST operation to upload an object, instead of providing the request header, you provide the same information in the form fields. For more information, see [POST Object](#).

The AWS SDKs also provide wrapper APIs that you can use to request server-side encryption. You can also use the AWS Management Console to upload objects and request server-side encryption.

Specifying Server-Side Encryption Using the AWS SDK for Java

When you use the AWS SDK for Java to upload an object, you can use server-side encryption to encrypt it. To request server-side encryption, use the `ObjectMetadata` property of the `PutObjectRequest` to

set the `x-amz-server-side-encryption` request header. When you call the `putObject()` method of the `AmazonS3Client`, Amazon S3 encrypts and saves the data.

You can also request server-side encryption when uploading objects with the multipart upload API:

- When using the high-level multipart upload API, you use the `TransferManager` methods to apply server-side encryption to objects as you upload them. You can use any of the upload methods that take `ObjectMetadata` as a parameter. For more information, see [Using the AWS Java SDK for Multipart Upload \(High-Level API\)](#) (p. 182).
- When using the low-level multipart upload API, you specify server-side encryption when you initiate the multipart upload. You add the `ObjectMetadata` property by calling the `InitiateMultipartUploadRequest.setObjectMetadata()` method. For more information, see [Upload a File](#) (p. 186).

You can't directly change the encryption state of an object (encrypting an unencrypted object or decrypting an encrypted object). To change an object's encryption state, you make a copy of the object, specifying the desired encryption state for the copy, and then delete the original object. Amazon S3 encrypts the copied object only if you explicitly request server-side encryption. To request encryption of the copied object through the Java API, use the `ObjectMetadata` property to specify server-side encryption in the `CopyObjectRequest`.

Example Example

The following example shows how to set server-side encryption using the AWS SDK for Java. It shows how to perform the following tasks:

- Upload a new object using server-side encryption.
- Change an object's encryption state (in this example, encrypting a previously unencrypted object) by making a copy of the object.
- Check the encryption state of the object.

For more information about server-side encryption, see [Specifying Server-Side Encryption Using the REST API](#) (p. 278). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.internal.SSEResultBase;
import com.amazonaws.services.s3.model.*;

import java.io.ByteArrayInputStream;

public class SpecifyServerSideEncryption {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String keyNameToEncrypt = "**** Key name for an object to upload and encrypt ****";
        String keyNameToCopyAndEncrypt = "**** Key name for an unencrypted object to be
encrypted by copying ****";
        String copiedObjectKeyName = "**** Key name for the encrypted copy of the
unencrypted object ****";

        try {
```

```

        AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
            .withRegion(clientRegion)
            .withCredentials(new ProfileCredentialsProvider())
            .build();

        // Upload an object and encrypt it with SSE.
        uploadObjectWithSSEEncryption(s3Client, bucketName, keyNameToEncrypt);

        // Upload a new unencrypted object, then change its encryption state
        // to encrypted by making a copy.
        changeSSEEncryptionStatusByCopying(s3Client,
            bucketName,
            keyNameToCopyAndEncrypt,
            copiedObjectKeyName);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}

private static void uploadObjectWithSSEEncryption(AmazonS3 s3Client, String bucketName,
String keyName) {
    String objectContent = "Test object encrypted with SSE";
    byte[] objectBytes = objectContent.getBytes();

    // Specify server-side encryption.
    ObjectMetadata objectMetadata = new ObjectMetadata();
    objectMetadata.setContentLength(objectBytes.length);
    objectMetadata.setSSEAlgorithm(ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);
    PutObjectRequest putRequest = new PutObjectRequest(bucketName,
        keyName,
        new ByteArrayInputStream(objectBytes),
        objectMetadata);

    // Upload the object and check its encryption status.
    PutObjectResult putResult = s3Client.putObject(putRequest);
    System.out.println("Object \"" + keyName + "\" uploaded with SSE.");
    printEncryptionStatus(putResult);
}

private static void changeSSEEncryptionStatusByCopying(AmazonS3 s3Client,
String bucketName,
String sourceKey,
String destKey) {
    // Upload a new, unencrypted object.
    PutObjectResult putResult = s3Client.putObject(bucketName, sourceKey, "Object
example to encrypt by copying");
    System.out.println("Unencrypted object \"" + sourceKey + "\" uploaded.");
    printEncryptionStatus(putResult);

    // Make a copy of the object and use server-side encryption when storing the copy.
    CopyObjectRequest request = new CopyObjectRequest(bucketName,
        sourceKey,
        bucketName,
        destKey);
    ObjectMetadata objectMetadata = new ObjectMetadata();
    objectMetadata.setSSEAlgorithm(ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);
    request.setNewObjectMetadata(objectMetadata);

    // Perform the copy operation and display the copy's encryption status.
    CopyObjectResult response = s3Client.copyObject(request);

```

```

        System.out.println("Object \"" + destKey + "\" uploaded with SSE.");
        printEncryptionStatus(response);

        // Delete the original, unencrypted object, leaving only the encrypted copy in
        Amazon S3.
        s3Client.deleteObject(bucketName, sourceKey);
        System.out.println("Unencrypted object \"" + sourceKey + "\" deleted.");
    }

    private static void printEncryptionStatus(SSEResultBase response) {
        String encryptionStatus = response.getSSEAlgorithm();
        if (encryptionStatus == null) {
            encryptionStatus = "Not encrypted with SSE";
        }
        System.out.println("Object encryption status is: " + encryptionStatus);
    }
}

```

Specifying Server-Side Encryption Using the AWS SDK for .NET

When you upload an object, you can direct Amazon S3 to encrypt it. To change the encryption state of an existing object, you make a copy of the object and delete the source object. By default, the copy operation encrypts the target only if you explicitly request server-side encryption of the target object. To specify server-side encryption in the `CopyObjectRequest`, add the following:

```
ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256
```

For a working sample of how to copy an object, see [Copy an Amazon S3 Object in a Single Operation Using the AWS SDK for .NET \(p. 212\)](#).

The following example uploads an object. In the request, the example directs Amazon S3 to encrypt the object. The example then retrieves object metadata and verifies the encryption method that was used. For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class SpecifyServerSideEncryptionTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** key name for object created ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            WritingAnObjectAsync().Wait();
        }

        static async Task WritingAnObjectAsync()
        {
            try
            {

```

```

var putRequest = new PutObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    ContentBody = "sample text",
    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256
};

var putResponse = await client.PutObjectAsync(putRequest);

// Determine the encryption state of an object.
GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
{
    BucketName = bucketName,
    Key = keyName
};
GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
ServerSideEncryptionMethod objectEncryption =
response.ServerSideEncryptionMethod;

    Console.WriteLine("Encryption method used: {0}",
objectEncryption.ToString());
}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
}
}
}
}

```

Specifying Server-Side Encryption Using the AWS SDK for PHP

This topic shows how to use classes from version 3 of the AWS SDK for PHP to add server-side encryption to objects that you upload to Amazon Simple Storage Service (Amazon S3). It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 678\)](#) and have the AWS SDK for PHP properly installed.

To upload an object to Amazon S3, use the [Aws\S3\S3Client::putObject\(\)](#) method. To add the `x-amz-server-side-encryption` request header to your upload request, specify the `ServerSideEncryption` parameter with the value `AES256`, as shown in the following code example. For information about server-side encryption requests, see [Specifying Server-Side Encryption Using the REST API \(p. 278\)](#).

```

require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// $filepath should be an absolute path to a file on disk.
$filepath = '*** Your File Path ***';

$s3 = new S3Client([
    'version' => 'latest',

```

```
'region' => 'us-east-1'
]);

// Upload a file with server-side encryption.
$result = $s3->putObject([
    'Bucket' => $bucket,
    'Key' => $keyname,
    'SourceFile' => $filepath,
    'ServerSideEncryption' => 'AES256',
]);
```

In response, Amazon S3 returns the `x-amz-server-side-encryption` header with the value of the encryption algorithm that was used to encrypt your object's data.

When you upload large objects using the multipart upload API, you can specify server-side encryption for the objects that you are uploading, as follows:

- When using the low-level multipart upload API, specify server-side encryption when you call the [Aws\S3\S3Client::createMultipartUpload\(\)](#) method. To add the `x-amz-server-side-encryption` request header to your request, specify the array parameter's `ServerSideEncryption` key with the value `AES256`. For more information about the low-level multipart upload API, see [Using the AWS PHP SDK for Multipart Upload \(Low-Level API\)](#) (p. 203).
- When using the high-level multipart upload API, specify server-side encryption using the `ServerSideEncryption` parameter of the [CreateMultipartUpload](#) method. For an example of using the `setOption()` method with the high-level multipart upload API, see [Using the AWS PHP SDK for Multipart Upload](#) (p. 201).

Determining Encryption Algorithm Used

To determine the encryption state of an existing object, retrieve the object metadata by calling the [Aws\S3\S3Client::headObject\(\)](#) method as shown in the following PHP code example.

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

// Check which server-side encryption algorithm is used.
$result = $s3->headObject([
    'Bucket' => $bucket,
    'Key' => $keyname,
]);
echo $result['ServerSideEncryption'];
```

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, make a copy of the object using the [Aws\S3\S3Client::copyObject\(\)](#) method and delete the source object. By default, `copyObject()` does not encrypt the target unless you explicitly request server-side encryption of the destination object using the `ServerSideEncryption` parameter with the value `AES256`. The following PHP code example makes a copy of an object and adds server-side encryption to the copied object.

```
require 'vendor/autoload.php';
```

```
use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';

$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

// Copy an object and add server-side encryption.
$s3->copyObject([
    'Bucket' => $targetBucket,
    'Key' => $targetKeyname,
    'CopySource' => "{$sourceBucket}/{$sourceKeyname}",
    'ServerSideEncryption' => 'AES256',
]);
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Specifying Server-Side Encryption Using the AWS SDK for Ruby

When using the AWS SDK for Ruby to upload an object, you can specify that the object be stored encrypted at rest with server-side encryption (SSE). When you read the object back, it is automatically decrypted.

The following AWS SDK for Ruby – Version 3 example demonstrates how to specify that a file uploaded to Amazon S3 be encrypted at rest.

```
# The following example demonstrates how to specify that a file uploaded to Amazon S3 be
# encrypted at rest.
require 'aws-sdk-s3'

regionName = 'us-west-2'
bucketName = 'my-bucket'
key = 'key'
filePath = 'local/path/to/file'
encryptionType = 'AES256'

s3 = Aws::S3::Resource.new(region: regionName)
obj = s3.bucket(bucketName).object(key)
obj.upload_file(filePath, :server_side_encryption => encryptionType)
```

For an example that shows how to upload an object without SSE, see [Upload an Object Using the AWS SDK for Ruby \(p. 173\)](#).

Determining the Encryption Algorithm Used

The following code example demonstrates how to determine the encryption state of an existing object.

```
# Determine server-side encryption of an object.
require 'aws-sdk-s3'

regionName = 'us-west-2'
```

```
bucketName='bucket-name'  
key = 'key'  
  
s3 = Aws::S3::Resource.new(region:regionName)  
enc = s3.bucket(bucketName).object(key).server_side_encryption  
enc_state = (enc != nil) ? enc : "not set"  
puts "Encryption state is #{enc_state}."
```

If server-side encryption is not used for the object that is stored in Amazon S3, the method returns null.

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, make a copy of the object and delete the source object. By default, the copy methods do not encrypt the target unless you explicitly request server-side encryption. You can request the encryption of the target object by specifying the `server_side_encryption` value in the options hash argument as shown in the following Ruby code example. The code example demonstrates how to copy an object and encrypt the copy.

```
require 'aws-sdk-s3'  
  
regionName = 'us-west-2'  
encryptionType = 'AES256'  
  
s3 = Aws::S3::Resource.new(region:regionName)  
bucket1 = s3.bucket('source-bucket-name')  
bucket2 = s3.bucket('target-bucket-name')  
obj1 = bucket1.object('Bucket1Key')  
obj2 = bucket2.object('Bucket2Key')  
  
obj1.copy_to(obj2, :server_side_encryption => encryptionType)
```

For a sample of how to copy an object without encryption, see [Copy an Object Using the AWS SDK for Ruby](#) (p. 215).

Specifying Server-Side Encryption Using the REST API

At the time of object creation—that is, when you are uploading a new object or making a copy of an existing object—you can specify if you want Amazon S3 to encrypt your data by adding the `x-amz-server-side-encryption` header to the request. Set the value of the header to the encryption algorithm `AES256` that Amazon S3 supports. Amazon S3 confirms that your object is stored using server-side encryption by returning the response header `x-amz-server-side-encryption`.

The following REST upload APIs accept the `x-amz-server-side-encryption` request header.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)

When uploading large objects using the multipart upload API, you can specify server-side encryption by adding the `x-amz-server-side-encryption` header to the Initiate Multipart Upload request. When you are copying an existing object, regardless of whether the source object is encrypted or not, the destination object is not encrypted unless you explicitly request server-side encryption.

The response headers of the following REST APIs return the `x-amz-server-side-encryption` header when an object is stored using server-side encryption.

- [PUT Object](#)
- [PUT Object - Copy](#)

- [POST Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part - Copy](#)
- [Complete Multipart Upload](#)
- [Get Object](#)
- [Head Object](#)

Note

Encryption request headers should not be sent for `GET` requests and `HEAD` requests if your object uses SSE-S3 or you'll get an HTTP 400 BadRequest error.

[Specifying Server-Side Encryption Using the AWS Management Console](#)

When uploading an object using the AWS Management Console, you can specify server-side encryption. For an example of how to upload an object, see [Uploading S3 Objects](#).

When you copy an object using the AWS Management Console, the console copies the object as is. That is, if the copy source is encrypted, the target object is encrypted. The console also allows you to add encryption to an object. For more information, see [How Do I Add Encryption to an S3 Object?](#).

[More Info](#)

- [Amazon S3 Default Encryption for S3 Buckets \(p. 66\)](#)

[Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\)](#)

Server-side encryption is about protecting data at rest. Using server-side encryption with customer-provided encryption keys (SSE-C) allows you to set your own encryption keys. With the encryption key you provide as part of your request, Amazon S3 manages both the encryption, as it writes to disks, and decryption, when you access your objects. Therefore, you don't need to maintain any code to perform data encryption and decryption. The only thing you do is manage the encryption keys you provide.

When you upload an object, Amazon S3 uses the encryption key you provide to apply AES-256 encryption to your data and removes the encryption key from memory.

Important

Amazon S3 does not store the encryption key you provide. Instead, it stores a randomly salted HMAC value of the encryption key to validate future requests. The salted HMAC value cannot be used to derive the value of the encryption key or to decrypt the contents of the encrypted object. That means if you lose the encryption key, you lose the object.

Note

Server-side encryption encrypts only the object data, not object metadata.

When you retrieve an object, you must provide the same encryption key as part of your request. Amazon S3 first verifies that the encryption key you provided matches, and then decrypts the object before returning the object data to you.

The highlights of SSE-C are:

- You must use HTTPS.

Important

Amazon S3 rejects any requests made over HTTP when using SSE-C. For security considerations, we recommend that you consider any key you send erroneously using HTTP to be compromised. You should discard the key, and rotate as appropriate.

- The ETag in the response is not the MD5 of the object data.
- You manage a mapping of which encryption key was used to encrypt which object. Amazon S3 does not store encryption keys. You are responsible for tracking which encryption key you provided for which object.
- If your bucket is versioning-enabled, each object version you upload using this feature can have its own encryption key. You are responsible for tracking which encryption key was used for which object version.
- Because you manage encryption keys on the client side, you manage any additional safeguards, such as key rotation, on the client side.

Warning

If you lose the encryption key, any GET request for an object without its encryption key fails, and you lose the object.

Using SSE-C

When using server-side encryption with customer-provided encryption keys (SSE-C), you must provide encryption key information using the following request headers.

Name	Description
x-amz-server-side-encryption-customer-algorithm	Use this header to specify the encryption algorithm. The header value must be "AES256".
x-amz-server-side-encryption-customer-key	Use this header to provide the 256-bit, base64-encoded encryption key for Amazon S3 to use to encrypt or decrypt your data.
x-amz-server-side-encryption-customer-key-MD5	Use this header to provide the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321 . Amazon S3 uses this header for a message integrity check to ensure that the encryption key was transmitted without error.

You can use AWS SDK wrapper libraries to add these headers to your request. If you need to, you can make the Amazon S3 REST API calls directly in your application.

Note

You cannot use the Amazon S3 console to upload an object and request SSE-C. You also cannot use the console to update (for example, change the storage class or add metadata) an existing object stored using SSE-C.

The following Amazon S3 APIs support these headers.

- GET operation — When retrieving objects using the GET API (see [GET Object](#)), you can specify the request headers. Torrents are not supported for objects encrypted using SSE-C.
- HEAD operation — To retrieve object metadata using the HEAD API (see [HEAD Object](#)), you can specify these request headers.
- PUT operation — When uploading data using the PUT API (see [PUT Object](#)), you can specify these request headers.
- Multipart Upload — When uploading large objects using the multipart upload API, you can specify these headers. You specify these headers in the initiate request (see [Initiate Multipart Upload](#)) and each subsequent part upload request ([Upload Part](#)). For each part upload request, the encryption information must be the same as what you provided in the initiate multipart upload request.

- POST operation — When using a POST operation to upload an object (see [POST Object](#)), instead of the request headers, you provide the same information in the form fields.
- Copy operation — When you copy an object (see [PUT Object - Copy](#)), you have both a source object and a target object. Accordingly, you have the following to consider:
 - If you want the target object encrypted using server-side encryption with AWS managed keys, you must provide the `x-amz-server-side-encryption` request header.
 - If you want the target object encrypted using SSE-C, you must provide encryption information using the three headers described in the preceding table.
 - If the source object is encrypted using SSE-C, you must provide encryption key information using the following headers so that Amazon S3 can decrypt the object for copying.

Name	Description
<code>x-amz-copy-source-server-side-encryption-customer-algorithm</code>	Include this header to specify the algorithm Amazon S3 should use to decrypt the source object. This value must be <code>AES256</code> .
<code>x-amz-copy-source-server-side-encryption-customer-key</code>	Include this header to provide the base64-encoded encryption key for Amazon S3 to use to decrypt the source object. This encryption key must be the one that you provided Amazon S3 when you created the source object. Otherwise, Amazon S3 cannot decrypt the object.
<code>x-amz-copy-source-server-side-encryption-customer-key-MD5</code>	Include this header to provide the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321 .

Presigned URL and SSE-C

You can generate a presigned URL that can be used for operations such as upload a new object, retrieve an existing object, or object metadata. Presigned URLs support the SSE-C as follows:

- When creating a presigned URL, you must specify the algorithm using the `x-amz-server-side-encryption-customer-algorithm` in the signature calculation.
- When using the presigned URL to upload a new object, retrieve an existing object, or retrieve only object metadata, you must provide all the encryption headers in your client application.

Note

For non-SSE-C objects, you can generate a presigned URL and directly paste that into a browser, for example to access the data.

However, this is not true for SSE-C objects because in addition to the presigned URL, you also need to include HTTP headers that are specific to SSE-C objects. Therefore, you can use the presigned URL for SSE-C objects only programmatically.

For more information, see the following topics:

- [Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the AWS SDK for Java \(p. 282\)](#)
- [Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the AWS SDK for .NET \(p. 286\)](#)
- [Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the REST API \(p. 293\)](#)

Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the AWS SDK for Java

The following example shows how to request server-side encryption with customer-provided keys (SSE-C) for objects. The example performs the following operations. Each operation shows how to specify SSE-C-related headers in the request:

- **Put object**—Uploads an object and requests server-side encryption using a customer-provided encryption key.
- **Get object**—Downloads the object uploaded in the previous step. In the request, you provide the same encryption information that you provided when you uploaded the object. Amazon S3 needs this information to decrypt the object so that it can return it to you.
- **Get object metadata**—Retrieves the object's metadata. You provide the same encryption information used when the object was created.
- **Copy object**—Makes a copy of the previously uploaded object. Because the source object is stored using SSE-C, you must provide its encryption information in your copy request. By default, Amazon S3 encrypts the copy of the object only if you explicitly request it. This example directs Amazon S3 to store an encrypted copy of the object using a new SSE-C key.

Note

This example shows how to upload an object in a single operation. When using the Multipart Upload API to upload large objects, you provide encryption information in the same way shown in this example. For examples of multipart uploads that use the AWS SDK for Java, see [Using the AWS Java SDK for Multipart Upload \(High-Level API\)](#) (p. 182) and [Using the AWS Java SDK for a Multipart Upload \(Low-Level API\)](#) (p. 186).

To add the required encryption information, you include an `SSECustomerKey` in your request. For more information about the `SSECustomerKey` class, see [Using SSE-C](#) (p. 280).

For information about SSE-C, see [Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\)](#) (p. 279). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import javax.crypto.KeyGenerator;
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

public class ServerSideEncryptionUsingClientSideEncryptionKey {
    private static SSECustomerKey SSE_KEY;
    private static AmazonS3 S3_CLIENT;
    private static KeyGenerator KEY_GENERATOR;

    public static void main(String[] args) throws IOException, NoSuchAlgorithmException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
```

```
String keyName = "*** Key name ***";
String uploadFileName = "*** File path ***";
String targetKeyName = "*** Target key name ***";

// Create an encryption key.
KEY_GENERATOR = KeyGenerator.getInstance("AES");
KEY_GENERATOR.init(256, new SecureRandom());
SSE_KEY = new SSECustomerKey(KEY_GENERATOR.generateKey());

try {
    S3_CLIENT = AmazonS3ClientBuilder.standard()
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(clientRegion)
        .build();

    // Upload an object.
    uploadObject(bucketName, keyName, new File(uploadFileName));

    // Download the object.
    downloadObject(bucketName, keyName);

    // Verify that the object is properly encrypted by attempting to retrieve it
    // using the encryption key.
    retrieveObjectMetadata(bucketName, keyName);

    // Copy the object into a new object that also uses SSE-C.
    copyObject(bucketName, keyName, targetKeyName);
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}

private static void uploadObject(String bucketName, String keyName, File file) {
    PutObjectRequest putRequest = new PutObjectRequest(bucketName, keyName,
file).withSSECustomerKey(SSE_KEY);
    S3_CLIENT.putObject(putRequest);
    System.out.println("Object uploaded");
}

private static void downloadObject(String bucketName, String keyName) throws
IOException {
    GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName,
keyName).withSSECustomerKey(SSE_KEY);
    S3Object object = S3_CLIENT.getObject(getObjectRequest);

    System.out.println("Object content: ");
    displayTextInputStream(object.getObjectContent());
}

private static void retrieveObjectMetadata(String bucketName, String keyName) {
    GetObjectMetadataRequest getMetadataRequest = new
GetObjectMetadataRequest(bucketName, keyName)
        .withSSECustomerKey(SSE_KEY);
    ObjectMetadata objectMetadata = S3_CLIENT.getObjectMetadata(getMetadataRequest);
    System.out.println("Metadata retrieved. Object size: " +
objectMetadata.getContentLength());
}

private static void copyObject(String bucketName, String keyName, String targetKeyName)
throws NoSuchAlgorithmException {
```

```
// Create a new encryption key for target so that the target is saved using SSE-C.
SSECustomerKey newSSEKey = new SSECustomerKey(KEY_GENERATOR.generateKey());

CopyObjectRequest copyRequest = new CopyObjectRequest(bucketName, keyName,
bucketName, targetKeyName)
    .withSourceSSECustomerKey(SSE_KEY)
    .withDestinationSSECustomerKey(newSSEKey);

S3_CLIENT.copyObject(copyRequest);
System.out.println("Object copied");
}

private static void displayTextInputStream(S3ObjectInputStream input) throws
IOException {
    // Read one line at a time from the input stream and display each line.
    BufferedReader reader = new BufferedReader(new InputStreamReader(input));
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
    System.out.println();
}
}
```

Other Amazon S3 Operations with SSE-C Using the AWS SDK for Java

The example in the preceding section shows how to request server-side encryption with customer-provided keys (SSE-C) in the PUT, GET, Head, and Copy operations. This section describes other APIs that support SSE-C.

To upload large objects, you can use multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 175\)](#)). You can use either high-level or low-level APIs to upload large objects. These APIs support encryption-related headers in the request.

- When using the high-level `TransferManager` API, you provide the encryption-specific headers in the `PutObjectRequest` (see [Using the AWS Java SDK for Multipart Upload \(High-Level API\) \(p. 182\)](#)).
- When using the low-level API, you provide encryption-related information in the `InitiateMultipartUploadRequest`, followed by identical encryption information in each `UploadPartRequest`. You do not need to provide any encryption-specific headers in your `CompleteMultipartUploadRequest`. For examples, see [Using the AWS Java SDK for a Multipart Upload \(Low-Level API\) \(p. 186\)](#).

The following example uses `TransferManager` to create objects and shows how to provide SSE-C related information. The example does the following:

- Creates an object using the `TransferManager.upload()` method. In the `PutObjectRequest` instance, you provide encryption key information to request. Amazon S3 encrypts the object using the customer-provided encryption key.
- Makes a copy of the object by calling the `TransferManager.copy()` method. The example directs Amazon S3 to encrypt the object copy using a new `SSECustomerKey`. Because the source object is encrypted using SSE-C, the `CopyObjectRequest` also provides the encryption key of the source object so that Amazon S3 can decrypt the object before copying it.

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
```

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CopyObjectRequest;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.SSECustomerKey;
import com.amazonaws.services.s3.transfer.Copy;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import javax.crypto.KeyGenerator;
import java.io.File;
import java.security.SecureRandom;

public class ServerSideEncryptionCopyObjectUsingHlwithSSEC {

    public static void main(String[] args) throws Exception {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String fileToUpload = "**** File path ****";
        String keyName = "**** New object key name ****";
        String targetKeyName = "**** Key name for object copy ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();
            TransferManager tm = TransferManagerBuilder.standard()
                .withS3Client(s3Client)
                .build();

            // Create an object from a file.
            PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, keyName,
                new File(fileToUpload));

            // Create an encryption key.
            KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
            keyGenerator.init(256, new SecureRandom());
            SSECustomerKey sseCustomerEncryptionKey = new
            SSECustomerKey(keyGenerator.generateKey());

            // Upload the object. TransferManager uploads asynchronously, so this call
            returns immediately.
            putObjectRequest.setSSECustomerKey(sseCustomerEncryptionKey);
            Upload upload = tm.upload(putObjectRequest);

            // Optionally, wait for the upload to finish before continuing.
            upload.waitForCompletion();
            System.out.println("Object created.");

            // Copy the object and store the copy using SSE-C with a new key.
            CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucketName,
            keyName, bucketName, targetKeyName);
            SSECustomerKey sseTargetObjectEncryptionKey = new
            SSECustomerKey(keyGenerator.generateKey());
            copyObjectRequest.setSourceSSECustomerKey(sseCustomerEncryptionKey);
            copyObjectRequest.setDestinationSSECustomerKey(sseTargetObjectEncryptionKey);

            // Copy the object. TransferManager copies asynchronously, so this call returns
            immediately.
            Copy copy = tm.copy(copyObjectRequest);

            // Optionally, wait for the upload to finish before continuing.
```

```

        copy.WaitForCompletion();
        System.out.println("Copy complete.");
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
}

```

Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the AWS SDK for .NET

The following C# example shows how server-side encryption with customer-provided keys (SSE-C) works. The example performs the following operations. Each operation shows how to specify SSE-C-related headers in the request.

- **Put object**—Uploads an object and requests server-side encryption using customer-provided encryption keys.
- **Get object**—Downloads the object that was uploaded in the previous step. The request provides the same encryption information that was provided when the object was uploaded. Amazon S3 needs this information to decrypt the object and return it to you.
- **Get object metadata**—Provides the same encryption information used when the object was created to retrieve the object's metadata.
- **Copy object**—Makes a copy of the uploaded object. Because the source object is stored using SSE-C, the copy request must provide encryption information. By default, Amazon S3 does not encrypt a copy of an object. The code directs Amazon S3 to encrypt the copied object using SSE-C by providing encryption-related information for the target. It also stores the target.

Note

For examples of uploading large objects using the multipart upload API, see [Using the AWS SDK for .NET for Multipart Upload \(High-Level API\)](#) (p. 191) and [Using the AWS SDK for .NET for Multipart Upload \(Low-Level API\)](#) (p. 197).

For information about SSE-C, see [Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\)](#) (p. 279)). For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples](#) (p. 678).

Example

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class SSEClientEncryptionKeyObjectOperationsTest
    {
        private const string bucketName = "**** bucket name ****";
        private const string keyName = "**** key name for new object created ****";
    }
}

```



```

private const string copyTargetKeyName = "**** key name for object copy ****";
// Specify your bucket region (an example region is shown).
private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
private static IAmazonS3 client;

public static void Main()
{
    client = new AmazonS3Client(bucketRegion);
    ObjectOpsUsingClientEncryptionKeyAsync().Wait();
}
private static async Task ObjectOpsUsingClientEncryptionKeyAsync()
{
    try
    {
        // Create an encryption key.
        Aes aesEncryption = Aes.Create();
        aesEncryption.KeySize = 256;
        aesEncryption.GenerateKey();
        string base64Key = Convert.ToBase64String(aesEncryption.Key);

        // 1. Upload the object.
        PutObjectRequest putObjectRequest = await UploadObjectAsync(base64Key);
        // 2. Download the object and verify that its contents matches what you
uploaded.
        await DownloadObjectAsync(base64Key, putObjectRequest);
        // 3. Get object metadata and verify that the object uses AES-256
encryption.
        await GetObjectMetadataAsync(base64Key);
        // 4. Copy both the source and target objects using server-side encryption
with
        // a customer-provided encryption key.
        await CopyObjectAsync(aesEncryption, base64Key);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}

private static async Task<PutObjectRequest> UploadObjectAsync(string base64Key)
{
    PutObjectRequest putObjectRequest = new PutObjectRequest
    {
        BucketName = bucketName,
        Key = keyName,
        ContentBody = "sample text",
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };
    PutObjectResponse putObjectResponse = await
client.PutObjectAsync(putObjectRequest);
    return putObjectRequest;
}
private static async Task DownloadObjectAsync(string base64Key, PutObjectRequest
putObjectRequest)
{
    GetObjectRequest getObjectRequest = new GetObjectRequest
    {
        BucketName = bucketName,

```

```

        Key = keyName,
        // Provide encryption information for the object stored in Amazon S3.
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };

    using (GetObjectResponse getResponse = await
client.GetObjectAsync(getObjectRequest))
    using (StreamReader reader = new StreamReader(getResponse.ResponseStream))
    {
        string content = reader.ReadToEnd();
        if (String.Compare(putObjectRequest.ContentBody, content) == 0)
            Console.WriteLine("Object content is same as we uploaded");
        else
            Console.WriteLine("Error...Object content is not same.");

        if (getResponse.ServerSideEncryptionCustomerMethod ==
ServerSideEncryptionCustomerMethod.AES256)
            Console.WriteLine("Object encryption method is AES256, same as we
set");
        else
            Console.WriteLine("Error...Object encryption method is not the same as
AES256 we set");

        // Assert.AreEqual(putObjectRequest.ContentBody, content);
        // Assert.AreEqual(ServerSideEncryptionCustomerMethod.AES256,
getResponse.ServerSideEncryptionCustomerMethod);
    }
}
private static async Task GetObjectMetadataAsync(string base64Key)
{
    GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest
    {
        BucketName = bucketName,
        Key = keyName,

        // The object stored in Amazon S3 is encrypted, so provide the necessary
encryption information.
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };

    GetObjectMetadataResponse getObjectMetadataResponse = await
client.GetObjectMetadataAsync(getObjectMetadataRequest);
    Console.WriteLine("The object metadata show encryption method used is: {0}",
getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
    // Assert.AreEqual(ServerSideEncryptionCustomerMethod.AES256,
getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
}
private static async Task CopyObjectAsync(Aes aesEncryption, string base64Key)
{
    aesEncryption.GenerateKey();
    string copyBase64Key = Convert.ToBase64String(aesEncryption.Key);

    CopyObjectRequest copyRequest = new CopyObjectRequest
    {
        SourceBucket = bucketName,
        SourceKey = keyName,
        DestinationBucket = bucketName,
        DestinationKey = copyTargetKeyName,
        // Information about the source object's encryption.
        CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,

```

```
CopySourceServerSideEncryptionCustomerProvidedKey = base64Key,
// Information about the target object's encryption.
ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
ServerSideEncryptionCustomerProvidedKey = copyBase64Key
    };
    await client.CopyObjectAsync(copyRequest);
}
}
```

Other Amazon S3 Operations and SSE-C

The example in the preceding section shows how to request server-side encryption with customer-provided key (SSE-C) in the PUT, GET, Head, and Copy operations. This section describes other Amazon S3 APIs that support SSE-C.

To upload large objects, you can use multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 175\)](#)). AWS SDK for .NET provides both high-level or low-level APIs to upload large objects. These APIs support encryption-related headers in the request.

- When using high-level Transfer-Utility API, you provide the encryption-specific headers in the `TransferUtilityUploadRequest` as shown. For code examples, see [Using the AWS SDK for .NET for Multipart Upload \(High-Level API\) \(p. 191\)](#).

```
TransferUtilityUploadRequest request = new TransferUtilityUploadRequest()
{
    FilePath = filePath,
    BucketName = existingBucketName,
    Key = keyName,
    // Provide encryption information.
    ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMethod.AES256,
    ServerSideEncryptionCustomerProvidedKey = base64Key,
};
```

- When using the low-level API, you provide encryption-related information in the initiate multipart upload request, followed by identical encryption information in the subsequent upload part requests. You do not need to provide any encryption-specific headers in your complete multipart upload request. For examples, see [Using the AWS SDK for .NET for Multipart Upload \(Low-Level API\) \(p. 197\)](#).

The following is a low-level multipart upload example that makes a copy of an existing large object. In the example, the object to be copied is stored in Amazon S3 using SSE-C, and you want to save the target object also using SSE-C. In the example, you do the following:

- Initiate a multipart upload request by providing an encryption key and related information.
- Provide source and target object encryption keys and related information in the `CopyPartRequest`.
- Obtain the size of the source object to be copied by retrieving the object metadata.
- Upload the objects in 5 MB parts.

Example

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
```

```
{
    class SSECLowLevelMPUCopyObjectTest
    {
        private const string existingBucketName = "*** bucket name ***";
        private const string sourceKeyName      = "*** source object key name ***";
        private const string targetKeyName      = "*** key name for the target object
***";
        private const string filePath           = @"*** file path ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            CopyObjClientEncryptionKeyAsync().Wait();
        }

        private static async Task CopyObjClientEncryptionKeyAsync()
        {
            Aes aesEncryption = Aes.Create();
            aesEncryption.KeySize = 256;
            aesEncryption.GenerateKey();
            string base64Key = Convert.ToBase64String(aesEncryption.Key);

            await CreateSampleObjUsingClientEncryptionKeyAsync(base64Key, s3Client);

            await CopyObjectAsync(s3Client, base64Key);
        }
        private static async Task CopyObjectAsync(IAmazonS3 s3Client, string base64Key)
        {
            List<CopyPartResponse> uploadResponses = new List<CopyPartResponse>();

            // 1. Initialize.
            InitiateMultipartUploadRequest initiateRequest = new
            InitiateMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = targetKeyName,
                ServerSideEncryptionCustomerMethod =
                ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            InitiateMultipartUploadResponse initResponse =
            await s3Client.InitiateMultipartUploadAsync(initiateRequest);

            // 2. Upload Parts.
            long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB
            long firstByte = 0;
            long lastByte = partSize;

            try
            {
                // First find source object size. Because object is stored encrypted with
                // customer provided key you need to provide encryption information in
                your request.
                GetObjectMetadataRequest getObjectMetadataRequest = new
                GetObjectMetadataRequest()
                {
                    BucketName = existingBucketName,
                    Key = sourceKeyName,
                    ServerSideEncryptionCustomerMethod =
                    ServerSideEncryptionCustomerMethod.AES256,
                    ServerSideEncryptionCustomerProvidedKey = base64Key // " * **source
                    object encryption key ***"
                };
            }
        }
    }
}
```

```

        GetObjectMetadataResponse getObjectMetadataResponse = await
s3Client.GetObjectMetadataAsync(getObjectMetadataRequest);

        long filePosition = 0;
        for (int i = 1; filePosition < getObjectMetadataResponse.ContentLength; i
++)
        {
            CopyPartRequest copyPartRequest = new CopyPartRequest
            {
                UploadId = initResponse.UploadId,
                // Source.
                SourceBucket = existingBucketName,
                SourceKey = sourceKeyName,
                // Source object is stored using SSE-C. Provide encryption
information.
                CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                CopySourceServerSideEncryptionCustomerProvidedKey =
base64Key, // "***source object encryption key ***",
                FirstByte = firstByte,
                // If the last part is smaller then our normal part size then use
the remaining size.
                LastByte = lastByte > getObjectMetadataResponse.ContentLength ?
                    getObjectMetadataResponse.ContentLength - 1 : lastByte,

                // Target.
                DestinationBucket = existingBucketName,
                DestinationKey = targetKeyName,
                PartNumber = i,
                // Encryption information for the target object.
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key
            };
            uploadResponses.Add(await s3Client.CopyPartAsync(copyPartRequest));
            filePosition += partSize;
            firstByte += partSize;
            lastByte += partSize;
        }

        // Step 3: complete.
        CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = targetKeyName,
            UploadId = initResponse.UploadId,
        };
        completeRequest.AddPartETags(uploadResponses);

        CompleteMultipartUploadResponse completeUploadResponse =
            await s3Client.CompleteMultipartUploadAsync(completeRequest);
    }
    catch (Exception exception)
    {
        Console.WriteLine("Exception occurred: {0}", exception.Message);
        AbortMultipartUploadRequest abortMPURRequest = new
AbortMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = targetKeyName,
            UploadId = initResponse.UploadId
        };
        s3Client.AbortMultipartUpload(abortMPURRequest);
    }
}

```

```

    }
    private static async Task CreateSampleObjUsingClientEncryptionKeyAsync(string
base64Key, IAmazonS3 s3Client)
    {
        // List to store upload part responses.
        List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

        // 1. Initialize.
        InitiateMultipartUploadRequest initiateRequest = new
InitiateMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = sourceKeyName,
            ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            ServerSideEncryptionCustomerProvidedKey = base64Key
        };

        InitiateMultipartUploadResponse initResponse =
            await s3Client.InitiateMultipartUploadAsync(initiateRequest);

        // 2. Upload Parts.
        long contentLength = new FileInfo(filePath).Length;
        long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

        try
        {
            long filePosition = 0;
            for (int i = 1; filePosition < contentLength; i++)
            {
                UploadPartRequest uploadRequest = new UploadPartRequest
                {
                    BucketName = existingBucketName,
                    Key = sourceKeyName,
                    UploadId = initResponse.UploadId,
                    PartNumber = i,
                    PartSize = partSize,
                    FilePosition = filePosition,
                    FilePath = filePath,
                    ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                    ServerSideEncryptionCustomerProvidedKey = base64Key
                };

                // Upload part and add response to our list.
                uploadResponses.Add(await s3Client.UploadPartAsync(uploadRequest));

                filePosition += partSize;
            }

            // Step 3: complete.
            CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = sourceKeyName,
                UploadId = initResponse.UploadId,
                //PartETags = new List<PartETag>(uploadResponses)

            };
            completeRequest.AddPartETags(uploadResponses);

            CompleteMultipartUploadResponse completeUploadResponse =
                await s3Client.CompleteMultipartUploadAsync(completeRequest);
        }
    }

```

```
        catch (Exception exception)
        {
            Console.WriteLine("Exception occurred: {0}", exception.Message);
            AbortMultipartUploadRequest abortMPURequest = new
            AbortMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = sourceKeyName,
                UploadId = initResponse.UploadId
            };
            await s3Client.AbortMultipartUploadAsync(abortMPURequest);
        }
    }
}
```

Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the REST API

The following Amazon S3 REST APIs support headers related to server-side encryption with customer-provided encryption keys. For more information about these headers, see [Using SSE-C \(p. 280\)](#).

- [GET Object](#)
- [HEAD Object](#)
- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part - Copy](#)

Protecting Data Using Client-Side Encryption

Client-side encryption is the act of encrypting data before sending it to Amazon S3. To enable client-side encryption, you have the following options:

- Use a master key stored in AWS KMS.
- Use a master key you store within your application.

The following AWS SDKs support client-side encryption:

- [AWS SDK for .NET](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for PHP](#)
- [AWS SDK for Ruby](#)
- [AWS SDK for C++](#)

Option 1: Using a Master Key stored in AWS KMS

- **When uploading an object**—Using the Customer Master Key (CMK) ID, the client first sends a request to the AWS Key Management Service (AWS KMS) for a key that it can use to encrypt your object data. AWS KMS returns two versions of a randomly generated data encryption key:

- A plaintext version that the client uses to encrypt the object data
- A cipher blob of the same data encryption key that the client uploads to Amazon S3 as object metadata

Note

The client obtains a unique data encryption key for each object that it uploads.

- **When downloading an object**—The client downloads the encrypted object from Amazon S3 along with the cipher blob version of the data encryption key stored as object metadata. The client then sends the cipher blob to AWS KMS to get the plaintext version of the key so that it can decrypt the object data.

For more information about AWS KMS, see [What is the AWS Key Management Service?](#) in the *AWS Key Management Service Developer Guide*.

Example

The following example uploads an object to Amazon S3 using AWS KMS with the AWS SDK for Java. The example uses a AWS-managed customer master key (CMK) to encrypt data on the client side before uploading it to Amazon S3. If you already have a CMK, you can use that by specifying the value of the `kms_cmk_id` variable in the sample code. If you don't have a CMK, or you need another one, you can generate one through the Java API. The example shows how to generate a CMK.

For more information about key material, see [Importing Key Material in AWS Key Management Service \(AWS KMS\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.regions.RegionUtils;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.CreateKeyResult;
import com.amazonaws.services.s3.AmazonS3Encryption;
import com.amazonaws.services.s3.AmazonS3EncryptionClientBuilder;
import com.amazonaws.services.s3.model.CryptoConfiguration;
import com.amazonaws.services.s3.model.KMSEncryptionMaterialsProvider;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

import java.io.ByteArrayOutputStream;
import java.io.IOException;

public class UploadObjectKMSKey {

    public static void main(String[] args) throws IOException {
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Object key name ***";
        Regions clientRegion = Regions.DEFAULT_REGION;
        String kms_cmk_id = "*** AWS KMS customer master key ID ***";
        int readChunkSize = 4096;

        try {
            // Optional: If you don't have a KMS key (or need another one),
            // create one. This example creates a key with AWS-created
            // key material.
            AWSKMS kmsClient = AWSKMSClientBuilder.standard()
                .withRegion(clientRegion)
                .build();
```



```

        CreateKeyResult keyResult = kmsClient.createKey();
        kms_cmk_id = keyResult.getKeyMetadata().getKeyId();

        // Create the encryption client.
        KMSEncryptionMaterialsProvider materialProvider = new
KMSEncryptionMaterialsProvider(kms_cmk_id);
        CryptoConfiguration cryptoConfig = new CryptoConfiguration()
            .withAwsKmsRegion(RegionUtils.getRegion(clientRegion.toString()));
        AmazonS3Encryption encryptionClient =
AmazonS3EncryptionClientBuilder.standard()
            .withCredentials(new ProfileCredentialsProvider())
            .withEncryptionMaterials(materialProvider)
            .withCryptoConfiguration(cryptoConfig)
            .withRegion(clientRegion).build();

        // Upload an object using the encryption client.
        String origContent = "S3 Encrypted Object Using KMS-Managed Customer Master
Key.";
        int origContentLength = origContent.length();
        encryptionClient.putObject(bucketName, keyName, origContent);

        // Download the object. The downloaded object is still encrypted.
        S3Object downloadedObject = encryptionClient.getObject(bucketName, keyName);
        S3ObjectInputStream input = downloadedObject.getObjectContent();

        // Decrypt and read the object and close the input stream.
        byte[] readBuffer = new byte[readChunkSize];
        ByteArrayOutputStream baos = new ByteArrayOutputStream(readChunkSize);
        int bytesRead = 0;
        int decryptedContentLength = 0;

        while ((bytesRead = input.read(readBuffer)) != -1) {
            baos.write(readBuffer, 0, bytesRead);
            decryptedContentLength += bytesRead;
        }
        input.close();

        // Verify that the original and decrypted contents are the same size.
        System.out.println("Original content length: " + origContentLength);
        System.out.println("Decrypted content length: " + decryptedContentLength);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
}
}

```

Option 2: Using a Master Key Stored Within Your Application

This section shows how to use a master key stored within your application for client-side data encryption.

Important

Your client-side master keys and your unencrypted data are never sent to AWS. It's important that you safely manage your encryption keys. If you lose them, you can't decrypt your data.

This is how it works:

- **When uploading an object**—You provide a client-side master key to the Amazon S3 encryption client. The client uses the master key only to encrypt the data encryption key that it generates randomly. The process works like this:
 1. The Amazon S3 encryption client generates a one-time-use symmetric key (also known as a data encryption key or data key) locally. It uses the data key to encrypt the data of a single Amazon S3 object. The client generates a separate data key for each object.
 2. The client encrypts the data encryption key using the master key that you provide. The client uploads the encrypted data key and its material description as part of the object metadata. The client uses the material description to determine which client-side master key to use for decryption.
 3. The client uploads the encrypted data to Amazon S3 and saves the encrypted data key as object metadata (`x-amz-meta-x-amz-key`) in Amazon S3.
- **When downloading an object**—The client downloads the encrypted object from Amazon S3. Using the material description from the object's metadata, the client determines which master key to use to decrypt the data key. The client uses that master key to decrypt the data key and then uses the data key to decrypt the object.

The client-side master key that you provide can be either a symmetric key or a public/private key pair. The following examples show how to use both types of keys.

For more information, see [Client-Side Data Encryption with the AWS SDK for Java and Amazon S3](#).

Note

If you get a cipher-encryption error message when you use the encryption API for the first time, your version of the JDK may have a Java Cryptography Extension (JCE) jurisdiction policy file that limits the maximum key length for encryption and decryption transformations to 128 bits. The AWS SDK requires a maximum key length of 256 bits. To check your maximum key length, use the `getMaxAllowedKeyLength()` method of the `javax.crypto.Cipher` class. To remove the key-length restriction, install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files at the [Java SE download page](#).

Example

The following example shows how to do these tasks:

- Generate a 256-bit AES key
- Save and load the AES key to and from the file system
- Use the AES key to encrypt data on the client side before sending it to Amazon S3
- Use the AES key to decrypt data received from Amazon S3
- Verify that the decrypted data is the same as the original data

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3EncryptionClientBuilder;
import com.amazonaws.services.s3.model.*;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.security.InvalidKeyException;
```

```
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.X509EncodedKeySpec;

public class S3ClientSideEncryptionSymMasterKey {

    public static void main(String[] args) throws Exception {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String objectKeyName = "**** Object key name ****";
        String masterKeyDir = System.getProperty("java.io.tmpdir");
        String masterKeyName = "secret.key";

        // Generate a symmetric 256-bit AES key.
        KeyGenerator symKeyGenerator = KeyGenerator.getInstance("AES");
        symKeyGenerator.init(256);
        SecretKey symKey = symKeyGenerator.generateKey();

        // To see how it works, save and load the key to and from the file system.
        saveSymmetricKey(masterKeyDir, masterKeyName, symKey);
        symKey = loadSymmetricAESKey(masterKeyDir, masterKeyName, "AES");

        try {
            // Create the Amazon S3 encryption client.
            EncryptionMaterials encryptionMaterials = new EncryptionMaterials(symKey);
            AmazonS3 s3EncryptionClient = AmazonS3EncryptionClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withEncryptionMaterials(new
StaticEncryptionMaterialsProvider(encryptionMaterials))
                .withRegion(clientRegion)
                .build();

            // Upload a new object. The encryption client automatically encrypts it.
            byte[] plaintext = "S3 Object Encrypted Using Client-Side Symmetric Master
Key.".getBytes();
            s3EncryptionClient.putObject(new PutObjectRequest(bucketName,
                objectKeyName,
                new ByteArrayInputStream(plaintext),
                new ObjectMetadata()));

            // Download and decrypt the object.
            S3Object downloadedObject = s3EncryptionClient.getObject(bucketName,
objectKeyName);
            byte[] decrypted =
com.amazonaws.util.IOUtils.toByteArray(downloadedObject.getObjectContent());

            // Verify that the data that you downloaded is the same as the original data.
            System.out.println("Plaintext: " + new String(plaintext));
            System.out.println("Decrypted text: " + new String(decrypted));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }

    private static void saveSymmetricKey(String masterKeyDir, String masterKeyName,
SecretKey secretKey) throws IOException {
        X509EncodedKeySpec x509EncodedKeySpec = new
X509EncodedKeySpec(secretKey.getEncoded());
        FileOutputStream keyOutputStream = new FileOutputStream(masterKeyDir +
File.separator + masterKeyName);
```

```

        keyOutputStream.write(x509EncodedKeySpec.getEncoded());
        keyOutputStream.close();
    }

    private static SecretKey loadSymmetricAESKey(String masterKeyDir, String masterKeyName,
        String algorithm)
        throws IOException, NoSuchAlgorithmException, InvalidKeySpecException,
        InvalidKeyException {
        // Read the key from the specified file.
        File keyFile = new File(masterKeyDir + File.separator + masterKeyName);
        FileInputStream keyInputStream = new FileInputStream(keyFile);
        byte[] encodedPrivateKey = new byte[(int) keyFile.length()];
        keyInputStream.read(encodedPrivateKey);
        keyInputStream.close();

        // Reconstruct and return the master key.
        return new SecretKeySpec(encodedPrivateKey, "AES");
    }
}

```

The following example shows how to do these tasks:

- Generate a 1024-bit RSA key pair.
- Save and load the RSA keys to and from the file system.
- Use the RSA keys to encrypt data on the client side before sending it to Amazon S3.
- Use the RSA keys to decrypt data received from Amazon S3.
- Verify that the decrypted data is the same as the original data.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3EncryptionClientBuilder;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.util.IOUtils;

import java.io.*;
import java.security.*;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

public class S3ClientSideEncryptionAsymmetricMasterKey {

    public static void main(String[] args) throws Exception {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String objectKeyName = "**** Key name ****";
        String rsaKeyDir = System.getProperty("java.io.tmpdir");
        String publicKeyName = "public.key";
        String privateKeyName = "private.key";

        // Generate a 1024-bit RSA key pair.
        KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance("RSA");
        keyGenerator.initialize(1024, new SecureRandom());
    }
}

```

```

    KeyPair origKeyPair = keyGenerator.generateKeyPair();

    // To see how it works, save and load the key pair to and from the file system.
    saveKeyPair(rsaKeyDir, publicKeyName, privateKeyName, origKeyPair);
    KeyPair keyPair = loadKeyPair(rsaKeyDir, publicKeyName, privateKeyName, "RSA");

    try {
        // Create the encryption client.
        EncryptionMaterials encryptionMaterials = new EncryptionMaterials(keyPair);
        AmazonS3 s3EncryptionClient = AmazonS3EncryptionClientBuilder.standard()
            .withCredentials(new ProfileCredentialsProvider())
            .withEncryptionMaterials(new
StaticEncryptionMaterialsProvider(encryptionMaterials))
            .withRegion(clientRegion)
            .build();

        // Create a new object.
        byte[] plaintext = "S3 Object Encrypted Using Client-Side Asymmetric Master
Key.".getBytes();
        S3Object object = new S3Object();
        object.setKey(objectKeyName);
        object.setObjectContent(new ByteArrayInputStream(plaintext));
        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setContentLength(plaintext.length);

        // Upload the object. The encryption client automatically encrypts it.
        PutObjectRequest putRequest = new PutObjectRequest(bucketName,
            object.getKey(),
            object.getObjectContent(),
            metadata);
        s3EncryptionClient.putObject(putRequest);

        // Download and decrypt the object.
        S3Object downloadedObject = s3EncryptionClient.getObject(bucketName,
object.getKey());
        byte[] decrypted = IOUtils.toByteArray(downloadedObject.getObjectContent());

        // Verify that the data that you downloaded is the same as the original data.
        System.out.println("Plaintext: " + new String(plaintext));
        System.out.println("Decrypted text: " + new String(decrypted));
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}

private static void saveKeyPair(String dir,
                                String publicKeyName,
                                String privateKeyName,
                                KeyPair keyPair) throws IOException {
    PrivateKey privateKey = keyPair.getPrivate();
    PublicKey publicKey = keyPair.getPublic();

    // Write the public key to the specified file.
    X509EncodedKeySpec x509EncodedKeySpec = new
X509EncodedKeySpec(publicKey.getEncoded());
    FileOutputStream publicKeyOutputStream = new FileOutputStream(dir + File.separator
+ publicKeyName);
    publicKeyOutputStream.write(x509EncodedKeySpec.getEncoded());
    publicKeyOutputStream.close();
}

```

```
        // Write the private key to the specified file.
        PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new
PKCS8EncodedKeySpec(privateKey.getEncoded());
        FileOutputStream privateKeyOutputStream = new FileOutputStream(dir + File.separator
+ privateKeyName);
        privateKeyOutputStream.write(pkcs8EncodedKeySpec.getEncoded());
        privateKeyOutputStream.close();
    }

    private static KeyPair loadKeyPair(String dir,
                                      String publicKeyName,
                                      String privateKeyName,
                                      String algorithm)
        throws IOException, NoSuchAlgorithmException, InvalidKeySpecException {
        // Read the public key from the specified file.
        File publicKeyFile = new File(dir + File.separator + publicKeyName);
        FileInputStream publicKeyInputStream = new FileInputStream(publicKeyFile);
        byte[] encodedPublicKey = new byte[(int) publicKeyFile.length()];
        publicKeyInputStream.read(encodedPublicKey);
        publicKeyInputStream.close();

        // Read the private key from the specified file.
        File privateKeyFile = new File(dir + File.separator + privateKeyName);
        FileInputStream privateKeyInputStream = new FileInputStream(privateKeyFile);
        byte[] encodedPrivateKey = new byte[(int) privateKeyFile.length()];
        privateKeyInputStream.read(encodedPrivateKey);
        privateKeyInputStream.close();

        // Convert the keys into a key pair.
        KeyFactory keyFactory = KeyFactory.getInstance(algorithm);
        X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(encodedPublicKey);
        PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);

        PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(encodedPrivateKey);
        PrivateKey privateKey = keyFactory.generatePrivate(privateKeySpec);

        return new KeyPair(publicKey, privateKey);
    }
}
```

Identity and Access Management in Amazon S3

By default, all Amazon S3 resources—buckets, objects, and related subresources (for example, lifecycle configuration and website configuration)—are private: only the resource owner, an AWS account that created it, can access the resource. The resource owner can optionally grant access permissions to others by writing an access policy.

Amazon S3 offers access policy options broadly categorized as resource-based policies and user policies. Access policies you attach to your resources (buckets and objects) are referred to as resource-based policies. For example, bucket policies and access control lists (ACLs) are resource-based policies. You can also attach access policies to users in your account. These are called user policies. You may choose to use resource-based policies, user policies, or some combination of these to manage permissions to your Amazon S3 resources. The introductory topics provide general guidelines for managing permissions.

We recommend you first review the access control overview topics. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources](#) (p. 301). Then for more information about specific access policy options, see the following topics:

- [Using Bucket Policies and User Policies](#) (p. 341)
- [Managing Access with ACLs](#) (p. 403)
- [Using Amazon S3 Block Public Access](#) (p. 414)

Introduction to Managing Access Permissions to Your Amazon S3 Resources

Topics

- [Overview of Managing Access](#) (p. 302)
- [How Amazon S3 Authorizes a Request](#) (p. 307)
- [Guidelines for Using the Available Access Policy Options](#) (p. 312)
- [Example Walkthroughs: Managing Access to Your Amazon S3 Resources](#) (p. 315)

The topics in this section provide an overview of managing access permissions to your Amazon S3 resources and provides guidelines for when to use which access control method. The topics also provides introductory example walkthroughs. We recommend you review these topics in order.

Several security best practices also address access control, including:

- [Ensure Amazon S3 buckets are not publicly accessible](#)
- [Implement least privilege access](#)
- [Use IAM roles](#)
- [Enable MFA \(Multi-Factor Authentication\) Delete](#)
- [Identify and audit all your Amazon S3 buckets](#)
- [Monitor AWS security advisories](#)

Overview of Managing Access

When granting permissions, you decide who is getting them, which Amazon S3 resources they are getting permissions for, and specific actions you want to allow on those resources.

Topics

- [Amazon S3 Resources: Buckets and Objects \(p. 302\)](#)
- [Amazon S3 Bucket and Object Ownership \(p. 302\)](#)
- [Resource Operations \(p. 303\)](#)
- [Managing Access to Resources \(Access Policy Options\) \(p. 303\)](#)
- [Which Access Control Method Should I Use? \(p. 306\)](#)
- [More Info \(p. 306\)](#)

Amazon S3 Resources: Buckets and Objects

In Amazon Web Services (AWS), a resource is an entity that you can work with. In Amazon S3, buckets and objects are the resources, and both have associated subresources. For example, bucket subresources include the following:

- `lifecycle` – Stores lifecycle configuration information (see [Object Lifecycle Management \(p. 119\)](#)).
- `website` – Stores website configuration information if you configure your bucket for website hosting (see [Hosting a Static Website on Amazon S3 \(p. 503\)](#)).
- `versioning` – Stores versioning configuration (see [PUT Bucket versioning](#)).
- `policy` and `acl` (access control list) – Store access permission information for the bucket.
- `cors` (cross-origin resource sharing) – Supports configuring your bucket to allow cross-origin requests (see [Cross-Origin Resource Sharing \(CORS\) \(p. 151\)](#)).
- `logging` – Enables you to request Amazon S3 to save bucket access logs.

Object subresources include the following:

- `acl` – Stores a list of access permissions on the object. This topic discusses how to use this subresource to manage object permissions (see [Managing Access with ACLs \(p. 403\)](#)).
- `restore` – Supports temporarily restoring an archived object (see [POST Object restore](#)). An object in the Glacier storage class is an archived object. To access the object, you must first initiate a restore request, which restores a copy of the archived object. In the request, you specify the number of days that you want the restored copy to exist. For more information about archiving objects, see [Object Lifecycle Management \(p. 119\)](#).

Amazon S3 Bucket and Object Ownership

Buckets and objects are Amazon S3 resources. By default, only the resource owner can access these resources. The resource owner refers to the AWS account that creates the resource. For example:

- The AWS account that you use to create buckets and upload objects owns those resources.
- If you upload an object using AWS Identity and Access Management (IAM) user or role credentials, the AWS account that the user or role belongs to owns the object.

- A bucket owner can grant cross-account permissions to another AWS account (or users in another account) to upload objects. In this case, the AWS account that uploads objects owns those objects. The bucket owner does not have permissions on the objects that other accounts own, with the following exceptions:
 - The bucket owner pays the bills. The bucket owner can deny access to any objects, or delete any objects in the bucket, regardless of who owns them.
 - The bucket owner can archive any objects or restore archived objects regardless of who owns them. Archival refers to the storage class used to store the objects. For more information, see [Object Lifecycle Management](#) (p. 119).

Ownership and Request Authentication

All requests to a bucket are either authenticated or unauthenticated. Authenticated requests must include a signature value that authenticates the request sender, unauthenticated requests do not. For more information on request authentication, see [Making Requests](#) (p. 10).

A bucket owner can allow unauthenticated requests. For example, unauthenticated [PUT Object](#) requests are allowed when a bucket has a public bucket policy, or when a bucket ACL grants `WRITE` or `FULL_CONTROL` access to the All Users group or the anonymous user specifically. For more information about public bucket policies and public ACLs, see [The Meaning of "Public"](#) (p. 417).

All unauthenticated requests are made by the anonymous user. This user is represented in access control lists (ACLs) by the specific canonical user ID `65a011a29cdf8ec533ec3d1ccaae921c`. If an object is uploaded to a bucket through an unauthenticated request, the anonymous user owns the object. The default object ACL grants `FULL_CONTROL` to the anonymous user as the object's owner. Therefore, Amazon S3 allows unauthenticated requests to retrieve the object or modify its ACL.

To prevent objects from being modified by the anonymous user, we recommend that you do not implement bucket policies that allow anonymous public writes to your bucket or use ACLs that allow the anonymous user write access to your bucket. You can enforce this recommended behavior by using Amazon S3 Block Public Access.

For more information about blocking public access, see [Using Amazon S3 Block Public Access](#) (p. 414). For more information about ACLs, see [Access Control List \(ACL\) Overview](#) (p. 403).

Important

AWS recommends that you don't use the AWS account root user credentials to make authenticated requests. Instead, create an IAM user and grant that user full access. We refer to these users as administrator users. You can use the administrator user credentials, instead of AWS account root user credentials, to interact with AWS and perform tasks, such as create a bucket, create users, and grant them permissions. For more information, see [AWS Account Root User Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in the *IAM User Guide*.

Resource Operations

Amazon S3 provides a set of operations to work with the Amazon S3 resources. For a list of available operations, go to [Operations on Buckets](#) and [Operations on Objects](#) in the *Amazon Simple Storage Service API Reference*.

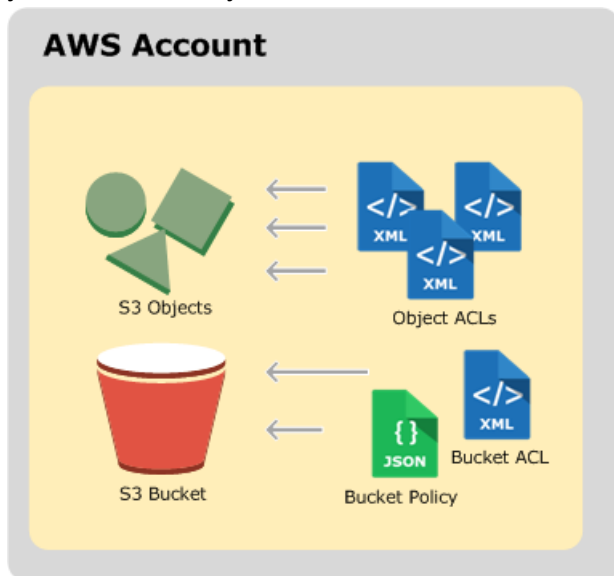
Managing Access to Resources (Access Policy Options)

Managing access refers to granting others (AWS accounts and users) permission to perform the resource operations by writing an access policy. For example, you can grant `PUT Object` permission to a user in an AWS account so the user can upload objects to your bucket. In addition to granting permissions to individual users and accounts, you can grant permissions to everyone (also referred as anonymous access) or to all authenticated users (users with AWS credentials). For example, if you configure your

bucket as a website, you may want to make objects public by granting the `GET` object permission to everyone.

Access policy describes who has access to what. You can associate an access policy with a resource (bucket and object) or a user. Accordingly, you can categorize the available Amazon S3 access policies as follows:

- **Resource-based policies** – Bucket policies and access control lists (ACLs) are resource-based because you attach them to your Amazon S3 resources.



- **ACL** – Each bucket and object has an ACL associated with it. An ACL is a list of grants identifying grantee and permission granted. You use ACLs to grant basic read/write permissions to other AWS accounts. ACLs use an Amazon S3-specific XML schema.

The following is an example bucket ACL. The grant in the ACL shows a bucket owner as having full control permission.

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="Canonical User">
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

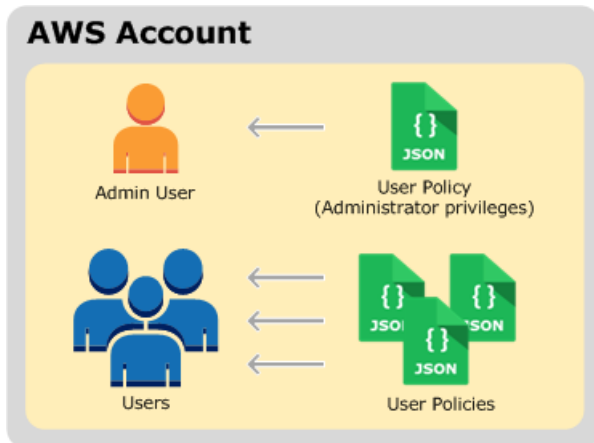
Both bucket and object ACLs use the same XML schema.

- **Bucket policy** – For your bucket, you can add a bucket policy to grant other AWS accounts or IAM users permissions for the bucket and the objects in it. Any object permissions apply only to the objects that the bucket owner creates. Bucket policies supplement, and in many cases, replace ACL-based access policies.

The following is an example bucket policy. You express bucket policy (and user policy) using a JSON file. The policy grants anonymous read permission on all objects in a bucket. The bucket policy has one statement, which allows the `s3:GetObject` action (read permission) on objects in a bucket named `examplebucket`. By specifying the `principal` with a wild card (*), the policy grants anonymous access, and should be used carefully. For example, the following bucket policy would make objects publicly accessible.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Resource": ["arn:aws:s3:::examplebucket/*"]
    }
  ]
}
```

- **User policies** – You can use IAM to manage access to your Amazon S3 resources. You can create IAM users, groups, and roles in your account and attach access policies to them granting them access to AWS resources, including Amazon S3.



For more information about IAM, see the [AWS Identity and Access Management \(IAM\)](#) product detail page.

The following is an example of a user policy. You cannot grant anonymous permissions in an IAM user policy, because the policy is attached to a user. The example policy allows the associated user that it's attached to perform six different Amazon S3 actions on a bucket and the objects in it. You can attach this policy to a specific IAM user, group, or role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStatement1",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject",
        "s3:GetBucketLocation"
      ]
    }
  ],
}
```

```
        "Resource": [
            "arn:aws:s3:::examplebucket/*",
            "arn:aws:s3:::examplebucket"
        ],
    },
    {
        "Sid": "ExampleStatement2",
        "Effect": "Allow",
        "Action": "s3:ListAllMyBuckets",
        "Resource": "*"
    }
]
```

When Amazon S3 receives a request, it must evaluate all the access policies to determine whether to authorize or deny the request. For more information about how Amazon S3 evaluates these policies, see [How Amazon S3 Authorizes a Request \(p. 307\)](#).

Which Access Control Method Should I Use?

With the options available to write an access policy, the following questions arise:

- When should I use which access control method? For example, to grant bucket permissions, should I use a bucket policy or bucket ACL? I own a bucket and the objects in the bucket. Should I use a resource-based access policy or an IAM user policy? If I use a resource-based access policy, should I use a bucket policy or an object ACL to manage object permissions?
- I own a bucket, but I don't own all of the objects in it. How are access permissions managed for the objects that somebody else owns?
- If I grant access by using a combination of these access policy options, how does Amazon S3 determine if a user has permission to perform a requested operation?

The following sections explain these access control alternatives, how Amazon S3 evaluates access control mechanisms, and when to use which access control method. They also provide example walkthroughs.

[How Amazon S3 Authorizes a Request \(p. 307\)](#)

[Guidelines for Using the Available Access Policy Options \(p. 312\)](#)

[Example Walkthroughs: Managing Access to Your Amazon S3 Resources \(p. 315\)](#)

More Info

We recommend that you first review the introductory topics that explain the options available for you to manage access to your Amazon S3 resources. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 301\)](#). You can then use the following topics for more information about specific access policy options.

- [Using Bucket Policies and User Policies \(p. 341\)](#)
- [Managing Access with ACLs \(p. 403\)](#)

How Amazon S3 Authorizes a Request

Topics

- [Related Topics \(p. 308\)](#)
- [How Amazon S3 Authorizes a Request for a Bucket Operation \(p. 308\)](#)
- [How Amazon S3 Authorizes a Request for an Object Operation \(p. 310\)](#)

When Amazon S3 receives a request—for example, a bucket or an object operation—it first verifies that the requester has the necessary permissions. Amazon S3 evaluates all the relevant access policies, user policies, and resource-based policies (bucket policy, bucket ACL, object ACL) in deciding whether to authorize the request. The following are some of the example scenarios:

- If the requester is an IAM principal, Amazon S3 must determine if the parent AWS account to which the principal belongs has granted the principal necessary permission to perform the operation. In addition, if the request is for a bucket operation, such as a request to list the bucket content, Amazon S3 must verify that the bucket owner has granted permission for the requester to perform the operation.

Note

To perform a specific operation on a resource, an IAM principal needs permission from both the parent AWS account to which it belongs and the AWS account that owns the resource.

- If the request is for an operation on an object that the bucket owner does not own, in addition to making sure the requester has permissions from the object owner, Amazon S3 must also check the bucket policy to ensure the bucket owner has not set explicit deny on the object.

Note

A bucket owner (who pays the bill) can explicitly deny access to objects in the bucket regardless of who owns it. The bucket owner can also delete any object in the bucket

In order to determine whether the requester has permission to perform the specific operation, Amazon S3 does the following, in order, when it receives a request:

1. Converts all the relevant access policies (user policy, bucket policy, ACLs) at run time into a set of policies for evaluation.
2. Evaluates the resulting set of policies in the following steps. In each step, Amazon S3 evaluates a subset of policies in a specific context, based on the context authority.
 - a. **User context** – In the user context, the parent account to which the user belongs is the context authority.

Amazon S3 evaluates a subset of policies owned by the parent account. This subset includes the user policy that the parent attaches to the user. If the parent also owns the resource in the request (bucket, object), Amazon S3 also evaluates the corresponding resource policies (bucket policy, bucket ACL, and object ACL) at the same time.

A user must have permission from the parent account to perform the operation.

This step applies only if the request is made by a user in an AWS account. If the request is made using root credentials of an AWS account, Amazon S3 skips this step.

- b. **Bucket context** – In the bucket context, Amazon S3 evaluates policies owned by the AWS account that owns the bucket.

If the request is for a bucket operation, the requester must have permission from the bucket owner. If the request is for an object, Amazon S3 evaluates all the policies owned by the bucket owner to check if the bucket owner has not explicitly denied access to the object. If there is an explicit deny set, Amazon S3 does not authorize the request.

- c. **Object context** – If the request is for an object, Amazon S3 evaluates the subset of policies owned by the object owner.

The following sections describe in detail and provide examples:

- [How Amazon S3 Authorizes a Request for a Bucket Operation](#) (p. 308)
- [How Amazon S3 Authorizes a Request for an Object Operation](#) (p. 310)

Related Topics

We recommend you first review the introductory topics that explain the options for managing access to your Amazon S3 resources. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources](#) (p. 301). You can then use the following topics for more information about specific access policy options.

- [Using Bucket Policies and User Policies](#) (p. 341)
- [Managing Access with ACLs](#) (p. 403)

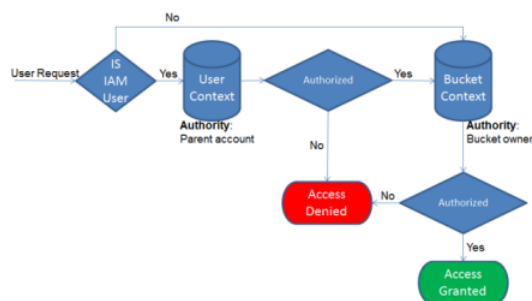
How Amazon S3 Authorizes a Request for a Bucket Operation

When Amazon S3 receives a request for a bucket operation, Amazon S3 converts all the relevant permissions—resource-based permissions (bucket policy, bucket access control list (ACL)) and IAM user policy if the request is from an IAM principal—into a set of policies to evaluate at run time. It then evaluates the resulting set of policies in a series of steps according to a specific context—user context or bucket context.

1. **User context** – If the requester is an IAM principal, the principal must have permission from the parent AWS account to which it belongs. In this step, Amazon S3 evaluates a subset of policies owned by the parent account (also referred to as the context authority). This subset of policies includes the user policy that the parent account attaches to the principal. If the parent also owns the resource in the request (in this case, the bucket), Amazon S3 also evaluates the corresponding resource policies (bucket policy and bucket ACL) at the same time. Whenever a request for a bucket operation is made, the server access logs record the canonical ID of the requester. For more information, see [Amazon S3 Server Access Logging](#) (p. 647).
2. **Bucket context** – The requester must have permissions from the bucket owner to perform a specific bucket operation. In this step, Amazon S3 evaluates a subset of policies owned by the AWS account that owns the bucket.

The bucket owner can grant permission by using a bucket policy or bucket ACL. Note that, if the AWS account that owns the bucket is also the parent account of an IAM principal, then it can configure bucket permissions in a user policy.

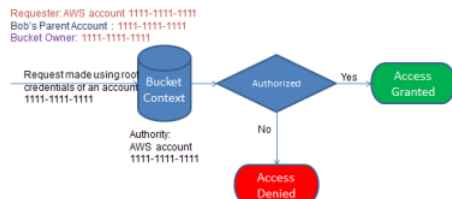
The following is a graphical illustration of the context-based evaluation for bucket operation.



The following examples illustrate the evaluation logic.

Example 1: Bucket Operation Requested by Bucket Owner

In this example, the bucket owner sends a request for a bucket operation using the root credentials of the AWS account.

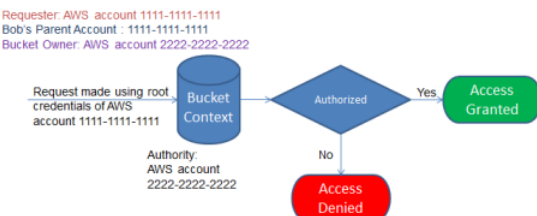


Amazon S3 performs the context evaluation as follows:

1. Because the request is made by using root credentials of an AWS account, the user context is not evaluated .
2. In the bucket context, Amazon S3 reviews the bucket policy to determine if the requester has permission to perform the operation. Amazon S3 authorizes the request.

Example 2: Bucket Operation Requested by an AWS Account That Is Not the Bucket Owner

In this example, a request is made using root credentials of AWS account 1111-1111-1111 for a bucket operation owned by AWS account 2222-2222-2222. No IAM users are involved in this request.

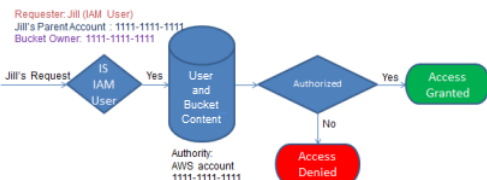


In this case, Amazon S3 evaluates the context as follows:

1. Because the request is made using root credentials of an AWS account, the user context is not evaluated.
2. In the bucket context, Amazon S3 examines the bucket policy. If the bucket owner (AWS account 2222-2222-2222) has not authorized AWS account 1111-1111-1111 to perform the requested operation, Amazon S3 denies the request. Otherwise, Amazon S3 grants the request and performs the operation.

Example 3: Bucket Operation Requested by an IAM Principal Whose Parent AWS Account Is Also the Bucket Owner

In the example, the request is sent by Jill, an IAM user in AWS account 1111-1111-1111, which also owns the bucket.



Amazon S3 performs the following context evaluation:

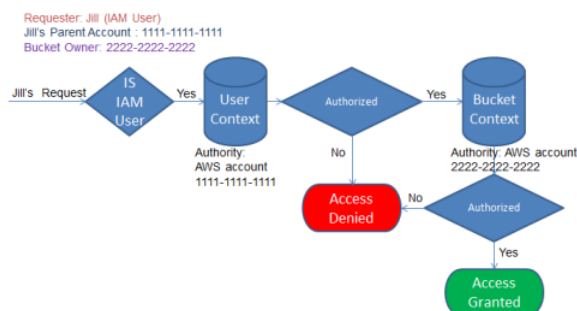
1. Because the request is from an IAM principal, in the user context, Amazon S3 evaluates all policies that belong to the parent AWS account to determine if Jill has permission to perform the operation.

In this example, parent AWS account 1111-1111-1111, to which the principal belongs, is also the bucket owner. As a result, in addition to the user policy, Amazon S3 also evaluates the bucket policy and bucket ACL in the same context, because they belong to the same account.

2. Because Amazon S3 evaluated the bucket policy and bucket ACL as part of the user context, it does not evaluate the bucket context.

Example 4: Bucket Operation Requested by an IAM Principal Whose Parent AWS Account Is Not the Bucket Owner

In this example, the request is sent by Jill, an IAM user whose parent AWS account is 1111-1111-1111, but the bucket is owned by another AWS account, 2222-2222-2222.



Jill will need permissions from both the parent AWS account and the bucket owner. Amazon S3 evaluates the context as follows:

1. Because the request is from an IAM principal, Amazon S3 evaluates the user context by reviewing the policies authored by the account to verify that Jill has the necessary permissions. If Jill has permission, then Amazon S3 moves on to evaluate the bucket context; if not, it denies the request.
2. In the bucket context, Amazon S3 verifies that bucket owner 2222-2222-2222 has granted Jill (or her parent AWS account) permission to perform the requested operation. If she has that permission, Amazon S3 grants the request and performs the operation; otherwise, Amazon S3 denies the request.

How Amazon S3 Authorizes a Request for an Object Operation

When Amazon S3 receives a request for an object operation, it converts all the relevant permissions—resource-based permissions (object access control list (ACL), bucket policy, bucket ACL) and IAM user policies—into a set of policies to be evaluated at run time. It then evaluates the resulting set of policies in a series of steps. In each step, it evaluates a subset of policies in three specific contexts—user context, bucket context, and object context.

1. **User context** – If the requester is an IAM principal, the principal must have permission from the parent AWS account to which it belongs. In this step, Amazon S3 evaluates a subset of policies owned by the parent account (also referred as the context authority). This subset of policies includes the user policy that the parent attaches to the principal. If the parent also owns the resource in the request (bucket, object), Amazon S3 evaluates the corresponding resource policies (bucket policy, bucket ACL, and object ACL) at the same time.

Note

If the parent AWS account owns the resource (bucket or object), it can grant resource permissions to its IAM principal by using either the user policy or the resource policy.

2. **Bucket context** – In this context, Amazon S3 evaluates policies owned by the AWS account that owns the bucket.

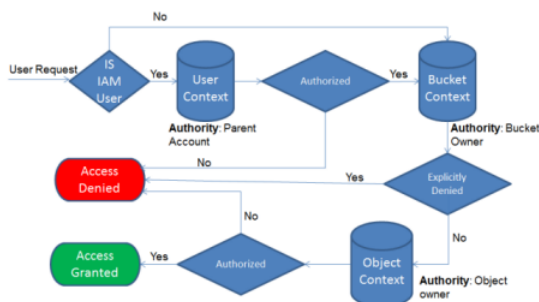
If the AWS account that owns the object in the request is not same as the bucket owner, in the bucket context Amazon S3 checks the policies if the bucket owner has explicitly denied access to the object. If there is an explicit deny set on the object, Amazon S3 does not authorize the request.

3. **Object context** – The requester must have permissions from the object owner to perform a specific object operation. In this step, Amazon S3 evaluates the object ACL.

Note

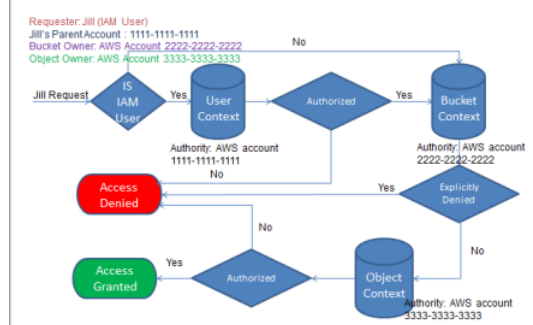
If bucket and object owners are the same, access to the object can be granted in the bucket policy, which is evaluated at the bucket context. If the owners are different, the object owners must use an object ACL to grant permissions. If the AWS account that owns the object is also the parent account to which the IAM principal belongs, it can configure object permissions in a user policy, which is evaluated at the user context. For more information about using these access policy alternatives, see [Guidelines for Using the Available Access Policy Options](#) (p. 312).

The following is an illustration of the context-based evaluation for an object operation.



Example 1: Object Operation Request

In this example, IAM user Jill, whose parent AWS account is 1111-1111-1111, sends an object operation request (for example, Get object) for an object owned by AWS account 3333-3333-3333 in a bucket owned by AWS account 2222-2222-2222.



Jill will need permission from the parent AWS account, the bucket owner, and the object owner. Amazon S3 evaluates the context as follows:

1. Because the request is from an IAM principal, Amazon S3 evaluates the user context to verify that the parent AWS account 1111-1111-1111 has given Jill permission to perform the requested operation. If she has that permission, Amazon S3 evaluates the bucket context. Otherwise, Amazon S3 denies the request.

2. In the bucket context, the bucket owner, AWS account 2222-2222-2222, is the context authority. Amazon S3 evaluates the bucket policy to determine if the bucket owner has explicitly denied Jill access to the object.
3. In the object context, the context authority is AWS account 3333-3333-3333, the object owner. Amazon S3 evaluates the object ACL to determine if Jill has permission to access the object. If she does, Amazon S3 authorizes the request.

Guidelines for Using the Available Access Policy Options

Amazon S3 supports resource-based policies and user policies to manage access to your Amazon S3 resources (see [Managing Access to Resources \(Access Policy Options\)](#) (p. 303)). Resource-based policies include bucket policies, bucket ACLs, and object ACLs. This section describes specific scenarios for using resource-based access policies to manage access to your Amazon S3 resources.

When to Use an ACL-based Access Policy (Bucket and Object ACLs)

Both buckets and objects have associated ACLs that you can use to grant permissions. The following sections describe scenarios for using object ACLs and bucket ACLs.

When to Use an Object ACL

In addition to an object ACL, there are other ways an object owner can manage object permissions. For example:

- If the AWS account that owns the object also owns the bucket, then it can write a bucket policy to manage the object permissions.
- If the AWS account that owns the object wants to grant permission to a user in its account, it can use a user policy.

So when do you use object ACLs to manage object permissions? The following are the scenarios when you use object ACLs to manage object permissions.

- **An object ACL is the only way to manage access to objects not owned by the bucket owner** – An AWS account that owns the bucket can grant another AWS account permission to upload objects. The bucket owner does not own these objects. The AWS account that created the object must grant permissions using object ACLs.

Note

A bucket owner cannot grant permissions on objects it does not own. For example, a bucket policy granting object permissions applies only to objects owned by the bucket owner. However, the bucket owner, who pays the bills, can write a bucket policy to deny access to any objects in the bucket, regardless of who owns it. The bucket owner can also delete any objects in the bucket.

- **Permissions vary by object and you need to manage permissions at the object level** – You can write a single policy statement granting an AWS account read permission on millions of objects with a specific key name prefix. For example, grant read permission on objects starting with key name prefix "logs". However, if your access permissions vary by object, granting permissions to individual objects using a bucket policy may not be practical. Also the bucket policies are limited to 20 KB in size.

In this case, you may find using object ACLs a suitable alternative. Although, even an object ACL is also limited to a maximum of 100 grants (see [Access Control List \(ACL\) Overview](#) (p. 403)).

- **Object ACLs control only object-level permissions** – There is a single bucket policy for the entire bucket, but object ACLs are specified per object.

An AWS account that owns a bucket can grant another AWS account permission to manage access policy. It allows that account to change anything in the policy. To better manage permissions, you

may choose not to give such a broad permission, and instead grant only the READ-ACP and WRITE-ACP permissions on a subset of objects. This limits the account to manage permissions only on specific objects by updating individual object ACLs.

When to Use a Bucket ACL

The only recommended use case for the bucket ACL is to grant write permission to the Amazon S3 Log Delivery group to write access log objects to your bucket (see [Amazon S3 Server Access Logging \(p. 647\)](#)). If you want Amazon S3 to deliver access logs to your bucket, you will need to grant write permission on the bucket to the Log Delivery group. The only way you can grant necessary permissions to the Log Delivery group is via a bucket ACL, as shown in the following bucket ACL fragment.

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    ...
  </Owner>
  <AccessControlList>
    <Grant>
      ...
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

When to Use a Bucket Policy

If an AWS account that owns a bucket wants to grant permission to users in its account, it can use either a bucket policy or a user policy. But in the following scenarios, you will need to use a bucket policy.

- **You want to manage cross-account permissions for all Amazon S3 permissions** – You can use ACLs to grant cross-account permissions to other accounts, but ACLs support only a finite set of permission ([What Permissions Can I Grant? \(p. 405\)](#)), these don't include all Amazon S3 permissions. For example, you cannot grant permissions on bucket subresources (see [Identity and Access Management in Amazon S3 \(p. 301\)](#)) using an ACL.

Although both bucket and user policies support granting permission for all Amazon S3 operations (see [Specifying Permissions in a Policy \(p. 345\)](#)), the user policies are for managing permissions for users in your account. For cross-account permissions to other AWS accounts or users in another account, you must use a bucket policy.

When to Use a User Policy

In general, you can use either a user policy or a bucket policy to manage permissions. You may choose to manage permissions by creating users and managing permissions individually by attaching policies to users (or user groups), or you may find that resource-based policies, such as a bucket policy, work better for your scenario.

Note that AWS Identity and Access Management (IAM) enables you to create multiple users within your AWS account and manage their permissions via user policies. An IAM user must have permissions from the parent account to which it belongs, and from the AWS account that owns the resource the user wants to access. The permissions can be granted as follows:

- **Permission from the parent account** – The parent account can grant permissions to its user by attaching a user policy.
- **Permission from the resource owner** – The resource owner can grant permission to either the IAM user (using a bucket policy) or the parent account (using a bucket policy, bucket ACL, or object ACL).

This is akin to a child who wants to play with a toy that belongs to someone else. In this case, the child must get permission from a parent to play with the toy and permission from the toy owner.

Permission Delegation

If an AWS account owns a resource, it can grant those permissions to another AWS account. That account can then delegate those permissions, or a subset of them, to users in the account. This is referred to as permission delegation. But an account that receives permissions from another account cannot delegate permission cross-account to another AWS account.

Related Topics

We recommend you first review all introductory topics that explain how you manage access to your Amazon S3 resources and related guidelines. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 301\)](#). You can then use the following topics for more information about specific access policy options.

- [Using Bucket Policies and User Policies \(p. 341\)](#)
- [Managing Access with ACLs \(p. 403\)](#)

Example Walkthroughs: Managing Access to Your Amazon S3 Resources

This topic provides the following introductory walkthrough examples for granting access to Amazon S3 resources. These examples use the AWS Management Console to create resources (buckets, objects, users) and grant them permissions. The examples then show you how to verify permissions using the command line tools, so you don't have to write any code. We provide commands using both the AWS Command Line Interface (CLI) and the AWS Tools for Windows PowerShell.

- [Example 1: Bucket Owner Granting Its Users Bucket Permissions \(p. 318\)](#)

The IAM users you create in your account have no permissions by default. In this exercise, you grant a user permission to perform bucket and object operations.

- [Example 2: Bucket Owner Granting Cross-Account Bucket Permissions \(p. 322\)](#)

In this exercise, a bucket owner, Account A, grants cross-account permissions to another AWS account, Account B. Account B then delegates those permissions to users in its account.

- **Managing object permissions when the object and bucket owners are not the same**

The example scenarios in this case are about a bucket owner granting object permissions to others, but not all objects in the bucket are owned by the bucket owner. What permissions does the bucket owner need, and how can it delegate those permissions?

The AWS account that creates a bucket is called the bucket owner. The owner can grant other AWS accounts permission to upload objects, and the AWS accounts that create objects own them. The bucket owner has no permissions on those objects created by other AWS accounts. If the bucket owner writes a bucket policy granting access to objects, the policy does not apply to objects that are owned by other accounts.

In this case, the object owner must first grant permissions to the bucket owner using an object ACL. The bucket owner can then delegate those object permissions to others, to users in its own account, or to another AWS account, as illustrated by the following examples.

- [Example 3: Bucket Owner Granting Its Users Permissions to Objects It Does Not Own \(p. 327\)](#)

In this exercise, the bucket owner first gets permissions from the object owner. The bucket owner then delegates those permissions to users in its own account.

- [Example 4: Bucket Owner Granting Cross-account Permission to Objects It Does Not Own \(p. 332\)](#)

After receiving permissions from the object owner, the bucket owner cannot delegate permission to other AWS accounts because cross-account delegation is not supported (see [Permission Delegation \(p. 314\)](#)). Instead, the bucket owner can create an IAM role with permissions to perform specific operations (such as get object) and allow another AWS account to assume that role. Anyone who assumes the role can then access objects. This example shows how a bucket owner can use an IAM role to enable this cross-account delegation.

Before You Try the Example Walkthroughs

These examples use the AWS Management Console to create resources and grant permissions. And to test permissions, the examples use the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code. To test permissions you will need to set up one of these tools. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

In addition, when creating resources these examples don't use root credentials of an AWS account. Instead, you create an administrator user in these accounts to perform these tasks.

About Using an Administrator User to Create Resources and Grant Permissions

AWS Identity and Access Management (IAM) recommends not using the root credentials of your AWS account to make requests. Instead, create an IAM user, grant that user full access, and then use that user's credentials to interact with AWS. We refer to this user as an administrator user. For more information, go to [Root Account Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in the *IAM User Guide*.

All example walkthroughs in this section use the administrator user credentials. If you have not created an administrator user for your AWS account, the topics show you how.

Note that to sign in to the AWS Management Console using the user credentials, you will need to use the IAM User Sign-In URL. The IAM console provides this URL for your AWS account. The topics show you how to get the URL.

Setting Up the Tools for the Example Walkthroughs

The introductory examples (see [Example Walkthroughs: Managing Access to Your Amazon S3 Resources](#) (p. 315)) use the AWS Management Console to create resources and grant permissions. And to test permissions, the examples use the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code. To test permissions, you must set up one of these tools.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*.

[Getting Set Up with the AWS Command Line Interface](#)

[Installing the AWS Command Line Interface](#)

[Configuring the AWS Command Line Interface](#)

2. Set the default profile.

You will store user credentials in the AWS CLI config file. Create a default profile in the config file using your AWS account credentials. See [Configuration and Credential Files](#) for instructions on finding and editing your AWS CLI config file.

```
[default]
aws_access_key_id = access key ID
aws_secret_access_key = secret access key
region = us-west-2
```

3. Verify the setup by entering the following command at the command prompt. Both these commands don't provide credentials explicitly, so the credentials of the default profile are used.

- Try the help command

```
aws help
```

- Use `aws s3 ls` to get a list of buckets on the configured account.

```
aws s3 ls
```

As you go through the example walkthroughs, you will create users, and you will save user credentials in the config files by creating profiles, as the following example shows. Note that these profiles have names (AccountAdmin and AccountBadmin):

```
[profile AccountAadmin]
aws_access_key_id = User AccountAadmin access key ID
aws_secret_access_key = User AccountAadmin secret access key
region = us-west-2

[profile AccountBadmin]
aws_access_key_id = Account B access key ID
aws_secret_access_key = Account B secret access key
region = us-east-1
```

To execute a command using these user credentials, you add the `--profile` parameter specifying the profile name. The following AWS CLI command retrieves a listing of objects in `examplebucket` and specifies the `AccountBadmin` profile.

```
aws s3 ls s3://examplebucket --profile AccountBadmin
```

Alternatively, you can configure one set of user credentials as the default profile by changing the `AWS_DEFAULT_PROFILE` environment variable from the command prompt. Once you've done this, whenever you execute AWS CLI commands without the `--profile` parameter, the AWS CLI will use the profile you set in the environment variable as the default profile.

```
$ export AWS_DEFAULT_PROFILE=AccountAadmin
```

To set up AWS Tools for Windows PowerShell

1. Download and configure the AWS Tools for Windows PowerShell. For instructions, go to [Download and Install the AWS Tools for Windows PowerShell](#) in the *AWS Tools for Windows PowerShell User Guide*.

Note

In order to load the AWS Tools for Windows PowerShell module, you need to enable PowerShell script execution. For more information, go to [Enable Script Execution](#) in the *AWS Tools for Windows PowerShell User Guide*.

2. For these exercises, you will specify AWS credentials per session using the `Set-AWSCredentials` command. The command saves the credentials to a persistent store (`-StoreAs` parameter).

```
Set-AWSCredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas string
```

3. Verify the setup.

- Execute the `Get-Command` to retrieve a list of available commands you can use for Amazon S3 operations.

```
Get-Command -module awspowershell -noun s3* -StoredCredentials string
```

- Execute the `Get-S3Object` command to retrieve a list of objects in a bucket.

```
Get-S3Object -BucketName bucketname -StoredCredentials string
```

For a list of commands, go to [Amazon Simple Storage Service Cmdlets](#).

Now you are ready to try the exercises. Follow the links provided at the beginning of the section.

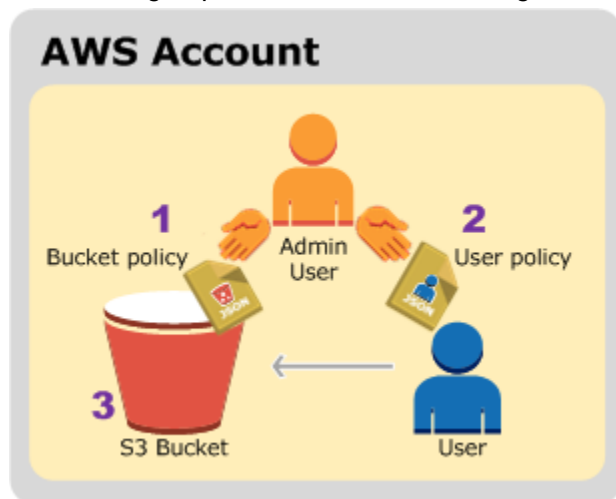
Example 1: Bucket Owner Granting Its Users Bucket Permissions

Topics

- [Step 0: Preparing for the Walkthrough \(p. 318\)](#)
- [Step 1: Create Resources \(a Bucket and an IAM User\) in Account A and Grant Permissions \(p. 319\)](#)
- [Step 2: Test Permissions \(p. 321\)](#)

In this exercise, an AWS account owns a bucket, and it has an IAM user in the account. The user by default has no permissions. The parent account must grant permissions to the user to perform any tasks. Both the bucket owner and the parent account to which the user belongs are the same. Therefore, the AWS account can use a bucket policy, a user policy, or both to grant its user permissions on the bucket. You will grant some permissions using a bucket policy and grant other permissions using a user policy.

The following steps summarize the walkthrough:



1. Account administrator creates a bucket policy granting a set of permissions to the user.
2. Account administrator attaches a user policy to the user granting additional permissions.
3. User then tries permissions granted via both the bucket policy and the user policy.

For this example, you will need an AWS account. Instead of using the root credentials of the account, you will create an administrator user (see [About Using an Administrator User to Create Resources and Grant Permissions \(p. 316\)](#)). We refer to the AWS account and the administrator user as follows:

Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAdmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, to verify the permissions, so you don't need to write any code.

Step 0: Preparing for the Walkthrough

1. Make sure you have an AWS account and that it has a user with administrator privileges.
 - a. Sign up for an account, if needed. We refer to this account as Account A.

- i. Go to <https://aws.amazon.com/s3> and click **Sign Up**.
- ii. Follow the on-screen instructions.

AWS will notify you by email when your account is active and available for you to use.

- b. In Account A, create an administrator user AccountAdmin. Using Account A credentials, sign in to the [IAM console](#) and do the following:

- i. Create user AccountAdmin and note down the user security credentials.

For instructions, see [Creating an IAM User in Your AWS Account](#) in the *IAM User Guide*.

- ii. Grant AccountAdmin administrator privileges by attaching a user policy giving full access.

For instructions, see [Working with Policies](#) in the *IAM User Guide*.

- iii. Note down the **IAM User Sign-In URL** for AccountAdmin. You will need to use this URL when signing in to the AWS Management Console. For more information about where to find it, see [How Users Sign in to Your Account](#) in *IAM User Guide*. Note down the URL for each of the accounts.

2. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:

- If using the AWS CLI, create a profile, AccountAdmin, in the config file.
- If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAdmin.

For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

Step 1: Create Resources (a Bucket and an IAM User) in Account A and Grant Permissions

Using the credentials of user AccountAdmin in Account A, and the special IAM user sign-in URL, sign in to the AWS Management Console and do the following:

1. Create Resources (a bucket and an IAM user)

- a. In the Amazon S3 console create a bucket. Note down the AWS region in which you created it. For instructions, see [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.
- b. In the IAM console, do the following:

- i. Create a user, Dave.

For instructions, see [Creating IAM Users \(AWS Management Console\)](#) in the *IAM User Guide*.

- ii. Note down the UserDave credentials.
- iii. Note down the Amazon Resource Name (ARN) for user Dave. In the IAM console, select the user, and the **Summary** tab provides the user ARN.

2. Grant Permissions.

Because the bucket owner and the parent account to which the user belongs are the same, the AWS account can grant user permissions using a bucket policy, a user policy, or both. In this example, you do both. If the object is also owned by the same account, the bucket owner can grant object permissions in the bucket policy (or an IAM policy).

- a. In the Amazon S3 console, attach the following bucket policy to *examplebucket*.

The policy has two statements.

- The first statement grants Dave the bucket operation permissions `s3:GetBucketLocation` and `s3:ListBucket`.
- The second statement grants the `s3:GetObject` permission. Because Account A also owns the object, the account administrator is able to grant the `s3:GetObject` permission.

In the `Principal` statement, Dave is identified by his user ARN. For more information about policy elements, see [Access Policy Language Overview \(p. 341\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
      },
      "Action": [
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::examplebucket"
      ]
    },
    {
      "Sid": "statement2",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
      },
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3::examplebucket/*"
      ]
    }
  ]
}
```

- b. Create an inline policy for the user Dave by using the following policy. The policy grants Dave the `s3:PutObject` permission. You need to update the policy by providing your bucket name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PermissionForObjectOperations",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3::examplebucket/*"
      ]
    }
  ]
}
```

For instructions, see [Working with Inline Policies](#) in the *IAM User Guide*. Note you need to sign in to the console using Account A credentials.

Step 2: Test Permissions

Using Dave's credentials, verify that the permissions work. You can use either of the following two procedures.

Test Using the AWS CLI

1. Update the AWS CLI config file by adding the following UserDaveAccountA profile. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

```
[profile UserDaveAccountA]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
region = us-east-1
```

2. Verify that Dave can perform the operations as granted in the user policy. Upload a sample object using the following AWS CLI `put-object` command.

The `--body` parameter in the command identifies the source file to upload. For example, if the file is in the root of the C: drive on a Windows machine, you specify `c:\HappyFace.jpg`. The `--key` parameter provides the key name for the object.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body HappyFace.jpg --
profile UserDaveAccountA
```

Execute the following AWS CLI command to get the object.

```
aws s3api get-object --bucket examplebucket --key HappyFace.jpg OutputFile.jpg --
profile UserDaveAccountA
```

Test Using the AWS Tools for Windows PowerShell

1. Store Dave's credentials as AccountADave. You then use these credentials to PUT and GET an object.

```
set-awscredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas
AccountADave
```

2. Upload a sample object using the AWS Tools for Windows PowerShell `write-S3Object` command using user Dave's stored credentials.

```
Write-S3Object -bucketname examplebucket -key HappyFace.jpg -file HappyFace.jpg -
StoredCredentials AccountADave
```

Download the previously uploaded object.

```
Read-S3Object -bucketname examplebucket -key HappyFace.jpg -file Output.jpg -
StoredCredentials AccountADave
```

Example 2: Bucket Owner Granting Cross-Account Bucket Permissions

Topics

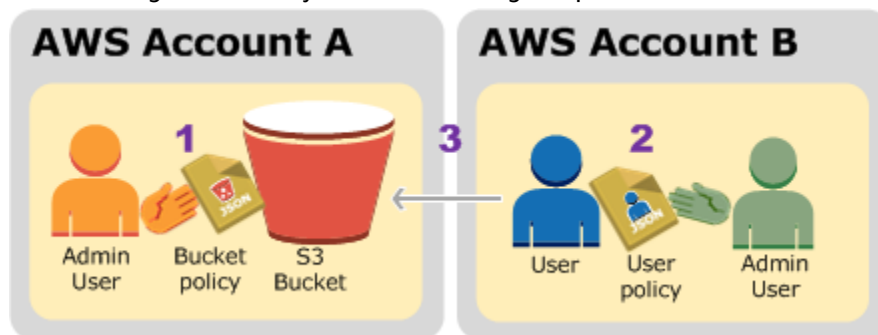
- [Step 0: Preparing for the Walkthrough \(p. 323\)](#)
- [Step 1: Do the Account A Tasks \(p. 324\)](#)
- [Step 2: Do the Account B Tasks \(p. 325\)](#)
- [Step 3: Extra Credit: Try Explicit Deny \(p. 326\)](#)
- [Step 4: Clean Up \(p. 327\)](#)

An AWS account—for example, Account A—can grant another AWS account, Account B, permission to access its resources such as buckets and objects. Account B can then delegate those permissions to users in its account. In this example scenario, a bucket owner grants cross-account permission to another account to perform specific bucket operations.

Note

Account A can also directly grant a user in Account B permissions using a bucket policy. But the user will still need permission from the parent account, Account B, to which the user belongs, even if Account B does not have permissions from Account A. As long as the user has permission from both the resource owner and the parent account, the user will be able to access the resource.

The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy granting cross-account permissions to Account B to perform specific bucket operations.

Note that administrator user in Account B will automatically inherit the permissions.

2. Account B administrator user attaches user policy to the user delegating the permissions it received from Account A.
3. User in Account B then verifies permissions by accessing an object in the bucket owned by Account A.

For this example, you need two accounts. The following table shows how we refer to these accounts and the administrator users in them. Per IAM guidelines (see [About Using an Administrator User to Create Resources and Grant Permissions \(p. 316\)](#)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions.

AWS Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAdmin

AWS Account ID	Account Referred To As	Administrator User in the Account
2222-2222-2222	Account B	AccountBadmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code.

Step 0: Preparing for the Walkthrough

- Make sure you have two AWS accounts and that each account has one administrator user as shown in the table in the preceding section.
 - Sign up for an AWS account, if needed.
 - Go to <https://aws.amazon.com/s3/> and click **Create an AWS Account**.
 - Follow the on-screen instructions.

AWS will notify you by email when your account is active and available for you to use.
 - Using Account A credentials, sign in to the [IAM console](#) to create the administrator user:
 - Create user AccountAadmin and note down the security credentials. For instructions, see [Creating an IAM User in Your AWS Account](#) in the *IAM User Guide*.
 - Grant AccountAadmin administrator privileges by attaching a user policy giving full access. For instructions, see [Working with Policies](#) in the *IAM User Guide*.
 - While you are in the IAM console, note down the **IAM User Sign-In URL** on the **Dashboard**. All users in the account must use this URL when signing in to the AWS Management Console.

For more information, see [How Users Sign in to Your Account](#) in *IAM User Guide*.

 - Repeat the preceding step using Account B credentials and create administrator user AccountBadmin.
- Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create two profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.

For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

- Save the administrator user credentials, also referred to as profiles. You can use the profile name instead of specifying credentials for each command you enter. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).
 - Add profiles in the AWS CLI credentials file for each of the administrator users in the two accounts.

```
[AccountAadmin]
aws_access_key_id = access-key-ID
aws_secret_access_key = secret-access-key
region = us-east-1

[AccountBadmin]
aws_access_key_id = access-key-ID
aws_secret_access_key = secret-access-key
```

```
region = us-east-1
```

- b. If you are using the AWS Tools for Windows PowerShell

```
set-awscredentials -AccessKey AcctA-access-key-ID -SecretKey AcctA-secret-access-key -storeas AccountAdmin  
set-awscredentials -AccessKey AcctB-access-key-ID -SecretKey AcctB-secret-access-key -storeas AccountBadmin
```

Step 1: Do the Account A Tasks

Step 1.1: Sign In to the AWS Management Console

Using the IAM user sign-in URL for Account A first sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a Bucket

1. In the Amazon S3 console, create a bucket. This exercise assumes the bucket is created in the US East (N. Virginia) region and is named `examplebucket`.

For instructions, see [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

2. Upload a sample object to the bucket.

For instructions, go to [Add an Object to a Bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*.

Step 1.3: Attach a Bucket Policy to Grant Cross-Account Permissions to Account B

The bucket policy grants the `s3:GetBucketLocation` and `s3:ListBucket` permissions to Account B. It is assumed you are still signed into the console using AccountAadmin user credentials.

1. Attach the following bucket policy to `examplebucket`. The policy grants Account B permission for the `s3:GetBucketLocation` and `s3:ListBucket` actions.

For instructions, see [How Do I Add an S3 Bucket Policy?](#) in the *Amazon Simple Storage Service Console User Guide*.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "Example permissions",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::AccountB-ID:root"  
      },  
      "Action": [  
        "s3:GetBucketLocation",  
        "s3:ListBucket"  
      ],  
      "Resource": [  
        "arn:aws:s3::examplebucket"  
      ]  
    }  
  ]  
}
```

2. Verify Account B (and thus its administrator user) can perform the operations.

- Using the AWS CLI

```
aws s3 ls s3://examplebucket --profile AccountBadmin
aws s3api get-bucket-location --bucket examplebucket --profile AccountBadmin
```

- Using the AWS Tools for Windows PowerShell

```
get-s3object -BucketName example2bucket -StoredCredentials AccountBadmin
get-s3bucketlocation -BucketName example2bucket -StoredCredentials AccountBadmin
```

Step 2: Do the Account B Tasks

Now the Account B administrator creates a user, Dave, and delegates the permissions received from Account A.

Step 2.1: Sign In to the AWS Management Console

Using the IAM user sign-in URL for Account B, first sign in to the AWS Management Console as AccountBadmin user.

Step 2.2: Create User Dave in Account B

In the IAM console, create a user, Dave.

For instructions, see [Creating IAM Users \(AWS Management Console\)](#) in the *IAM User Guide*.

Step 2.3: Delegate Permissions to User Dave

Create an inline policy for the user Dave by using the following policy. You will need to update the policy by providing your bucket name.

It is assumed you are signed in to the console using AccountBadmin user credentials.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Example",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket"
      ]
    }
  ]
}
```

For instructions, see [Working with Inline Policies](#) in the *IAM User Guide*.

Step 2.4: Test Permissions

Now Dave in Account B can list the contents of examplebucket owned by Account A. You can verify the permissions using either of the following procedures.

Test Using the AWS CLI

1. Add the UserDave profile to the AWS CLI config file. For more information about the config file, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

```
[profile UserDave]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
region = us-east-1
```

2. At the command prompt, enter the following AWS CLI command to verify Dave can now get an object list from the `examplebucket` owned by Account A. Note the command specifies the UserDave profile.

```
aws s3 ls s3://examplebucket --profile UserDave
```

Dave does not have any other permissions. So if he tries any other operation—for example, the following get bucket location—Amazon S3 returns permission denied.

```
aws s3api get-bucket-location --bucket examplebucket --profile UserDave
```

Test Using AWS Tools for Windows PowerShell

1. Store Dave's credentials as AccountBDave.

```
set-awscredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas
AccountBDave
```

2. Try the List Bucket command.

```
get-s3object -BucketName example2bucket -StoredCredentials AccountBDave
```

Dave does not have any other permissions. So if he tries any other operation—for example, the following get bucket location—Amazon S3 returns permission denied.

```
get-s3bucketlocation -BucketName example2bucket -StoredCredentials AccountBDave
```

Step 3: Extra Credit: Try Explicit Deny

You can have permissions granted via an ACL, a bucket policy, and a user policy. But if there is an explicit deny set via either a bucket policy or a user policy, the explicit deny takes precedence over any other permissions. For testing, let's update the bucket policy and explicitly deny Account B the `s3:ListBucket` permission. The policy also grants `s3:ListBucket` permission, but explicit deny takes precedence, and Account B or users in Account B will not be able to list objects in `examplebucket`.

1. Using credentials of user AccountAadmin in Account A, replace the bucket policy by the following.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Example permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountB-ID:root"
      }
    }
  ]
}
```



```
    },
    "Action": [
      "s3:GetBucketLocation",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::examplebucket"
    ]
  },
  {
    "Sid": "Deny permission",
    "Effect": "Deny",
    "Principal": {
      "AWS": "arn:aws:iam::AccountB-ID:root"
    },
    "Action": [
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::examplebucket"
    ]
  }
]
```

2. Now if you try to get a bucket list using AccountBadmin credentials, you will get access denied.

- Using the AWS CLI:

```
aws s3 ls s3://examplebucket --profile AccountBadmin
```

- Using the AWS Tools for Windows PowerShell:

```
get-s3object -BucketName example2bucket -StoredCredentials AccountBDave
```

Step 4: Clean Up

1. After you are done testing, you can do the following to clean up.

- Sign in to the AWS Management Console ([AWS Management Console](#)) using Account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to `examplebucket`. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the AccountAadmin user.

2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.

Example 3: Bucket Owner Granting Its Users Permissions to Objects It Does Not Own

Topics

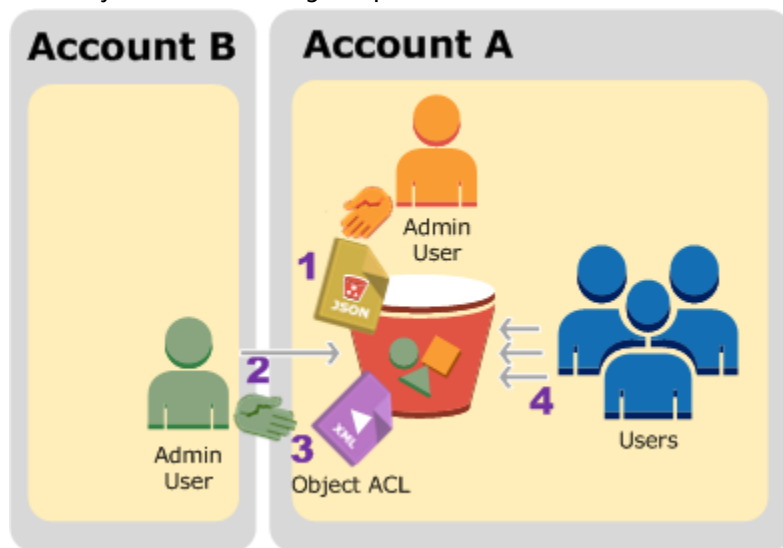
- [Step 0: Preparing for the Walkthrough \(p. 329\)](#)
- [Step 1: Do the Account A Tasks \(p. 329\)](#)
- [Step 2: Do the Account B Tasks \(p. 330\)](#)

- [Step 3: Test Permissions](#) (p. 331)
- [Step 4: Clean Up](#) (p. 332)

The scenario for this example is that a bucket owner wants to grant permission to access objects, but not all objects in the bucket are owned by the bucket owner. How can a bucket owner grant permission on objects it does not own? For this example, the bucket owner is trying to grant permission to users in its own account.

A bucket owner can enable other AWS accounts to upload objects. These objects are owned by the accounts that created them. The bucket owner does not own objects that were not created by the bucket owner. Therefore, for the bucket owner to grant access to these objects, the object owner must first grant permission to the bucket owner using an object ACL. For more information, see [Amazon S3 Bucket and Object Ownership](#) (p. 302).

In this example, the bucket owner delegates permission to users in its own account. The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy with two statements.
 - Allow cross-account permission to Account B to upload objects.
 - Allow a user in its own account to access objects in the bucket.
2. Account B administrator user uploads objects to the bucket owned by Account A.
3. Account B administrator updates the object ACL adding grant that gives the bucket owner full-control permission on the object.
4. User in Account A verifies by accessing objects in the bucket, regardless of who owns them.

For this example, you need two accounts. The following table shows how we refer to these accounts and the administrator users in these accounts. Per IAM guidelines (see [About Using an Administrator User to Create Resources and Grant Permissions](#) (p. 316)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions.

AWS Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAdmin

AWS Account ID	Account Referred To As	Administrator User in the Account
2222-2222-2222	Account B	AccountBadmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code.

Step 0: Preparing for the Walkthrough

- Make sure you have two AWS accounts and each account has one administrator user as shown in the table in the preceding section.
 - Sign up for an AWS account, if needed.
 - Go to <https://aws.amazon.com/s3/> and click **Create an AWS Account**.
 - Follow the on-screen instructions. AWS will notify you by email when your account is active and available for you to use.
 - Using Account A credentials, sign in to the [IAM console](#) and do the following to create an administrator user:
 - Create user AccountAadmin and note down security credentials. For more information about adding users, see [Creating an IAM User in Your AWS Account](#) in the *IAM User Guide*.
 - Grant AccountAadmin administrator privileges by attaching a user policy giving full access. For instructions, see [Working with Policies](#) in the *IAM User Guide*.
 - In the IAM console **Dashboard**, note down the **IAM User Sign-In URL**. Users in this account must use this URL when signing in to the AWS Management Console. For more information, see [How Users Sign in to Your Account](#) in *IAM User Guide*.
 - Repeat the preceding step using Account B credentials and create administrator user AccountBadmin.
- Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create two profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.

For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

Step 1: Do the Account A Tasks

Step 1.1: Sign In to the AWS Management Console

Using the IAM user sign-in URL for Account A first sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a Bucket, a User, and Add a Bucket Policy Granting User Permissions

- In the Amazon S3 console, create a bucket. This exercise assumes the bucket is created in the US East (N. Virginia) region and the name is `examplebucket`.

For instructions, see [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.
- In the IAM console, create a user Dave.

For instructions, see [Creating IAM Users \(AWS Management Console\)](#) in the *IAM User Guide*.

3. Note down the Dave credentials.
4. In the Amazon S3 console, attach the following bucket policy to `examplebucket` bucket. For instructions, see [How Do I Add an S3 Bucket Policy?](#) in the *Amazon Simple Storage Service Console User Guide*. Follow the steps to add a bucket policy. For information about how to find account IDs, see [Finding Your AWS Account ID](#).

The policy grants Account B the `s3:PutObject` and `s3:ListBucket` permissions. The policy also grants user Dave the `s3:GetObject` permission.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountB-ID:root"
      },
      "Action": [
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::examplebucket/*",
        "arn:aws:s3::examplebucket"
      ]
    },
    {
      "Sid": "Statement3",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
      },
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3::examplebucket/*"
      ]
    }
  ]
}
```

Step 2: Do the Account B Tasks

Now that Account B has permissions to perform operations on Account A's bucket, the Account B administrator will do the following;

- Upload an object to Account A's bucket.
- Add a grant in the object ACL to allow Account A, the bucket owner, full control.

Using the AWS CLI

1. Using the `put-object` AWS CLI command, upload an object. The `--body` parameter in the command identifies the source file to upload. For example, if the file is on `C:` drive of a Windows machine, you would specify `c:\HappyFace.jpg`. The `--key` parameter provides the key name for the object.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body HappyFace.jpg --profile AccountBadmin
```

2. Add a grant to the object ACL to allow the bucket owner full control of the object. For information about how to find a canonical user ID, see [Finding Your Account Canonical User ID](#).

```
aws s3api put-object-acl --bucket examplebucket --key HappyFace.jpg --grant-full-control id="AccountA-CanonicalUserID" --profile AccountBadmin
```

Using the AWS Tools for Windows PowerShell

1. Using the Write-S3Object AWS Tools for Windows PowerShell command, upload an object.

```
Write-S3Object -BucketName examplebucket -key HappyFace.jpg -file HappyFace.jpg -StoredCredentials AccountBadmin
```

2. Add a grant to the object ACL to allow the bucket owner full control of the object.

```
Set-S3ACL -BucketName examplebucket -Key HappyFace.jpg -CannedACLName "bucket-owner-full-control" -StoredCreden
```

Step 3: Test Permissions

Now verify user Dave in Account A can access the object owned by Account B.

Using the AWS CLI

1. Add user Dave credentials to the AWS CLI config file and create a new profile, UserDaveAccountA. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

```
[profile UserDaveAccountA]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
region = us-east-1
```

2. Execute the get-object AWS CLI command to download HappyFace.jpg and save it locally. You provide user Dave credentials by adding the --profile parameter.

```
aws s3api get-object --bucket examplebucket --key HappyFace.jpg Outputfile.jpg --profile UserDaveAccountA
```

Using the AWS Tools for Windows PowerShell

1. Store user Dave AWS credentials, as UserDaveAccountA, to persistent store.

```
Set-AWSCredentials -AccessKey UserDave-AccessKey -SecretKey UserDave-SecretAccessKey -storeas UserDaveAccountA
```

2. Execute the Read-S3Object command to download the HappyFace.jpg object and save it locally. You provide user Dave credentials by adding the -StoredCredentials parameter.

```
Read-S3Object -BucketName examplebucket -Key HappyFace.jpg -file HappyFace.jpg -StoredCredentials UserDaveAccountA
```

Step 4: Clean Up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the AWS Management Console ([AWS Management Console](#)) using Account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to `examplebucket`. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the AccountAdmin user.
2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.

Example 4: Bucket Owner Granting Cross-account Permission to Objects It Does Not Own

Topics

- [Background: Cross-Account Permissions and Using IAM Roles \(p. 332\)](#)
- [Step 0: Preparing for the Walkthrough \(p. 334\)](#)
- [Step 1: Do the Account A Tasks \(p. 335\)](#)
- [Step 2: Do the Account B Tasks \(p. 337\)](#)
- [Step 3: Do the Account C Tasks \(p. 338\)](#)
- [Step 4: Clean Up \(p. 339\)](#)
- [Related Resources \(p. 340\)](#)

In this example scenario, you own a bucket and you have enabled other AWS accounts to upload objects. That is, your bucket can have objects that other AWS accounts own.

Now, suppose as a bucket owner, you need to grant cross-account permission on objects, regardless of who the owner is, to a user in another account. For example, that user could be a billing application that needs to access object metadata. There are two core issues:

- The bucket owner has no permissions on those objects created by other AWS accounts. So for the bucket owner to grant permissions on objects it does not own, the object owner, the AWS account that created the objects, must first grant permission to the bucket owner. The bucket owner can then delegate those permissions.
- Bucket owner account can delegate permissions to users in its own account (see [Example 3: Bucket Owner Granting Its Users Permissions to Objects It Does Not Own \(p. 327\)](#)), but it cannot delegate permissions to other AWS accounts, because cross-account delegation is not supported.

In this scenario, the bucket owner can create an AWS Identity and Access Management (IAM) role with permission to access objects, and grant another AWS account permission to assume the role temporarily enabling it to access objects in the bucket.

Background: Cross-Account Permissions and Using IAM Roles

IAM roles enable several scenarios to delegate access to your resources, and cross-account access is one of the key scenarios. In this example, the bucket owner, Account A, uses an IAM role to temporarily delegate object access cross-account to users in another AWS account, Account C. Each IAM role you create has two policies attached to it:

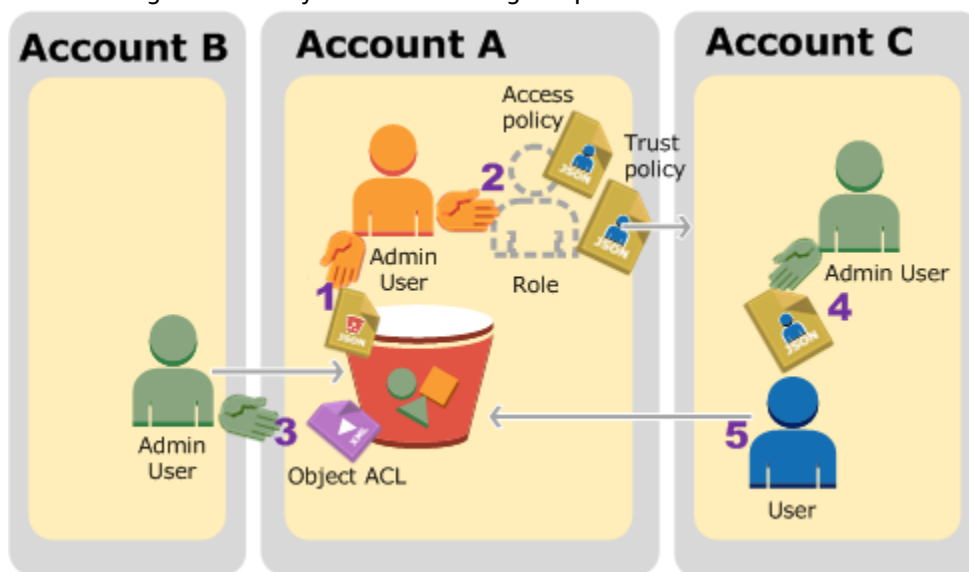
- A trust policy identifying another AWS account that can assume the role.
- An access policy defining what permissions—for example, `s3:GetObject`—are allowed when someone assumes the role. For a list of permissions you can specify in a policy, see [Specifying Permissions in a Policy \(p. 345\)](#).

The AWS account identified in the trust policy then grants its user permission to assume the role. The user can then do the following to access objects:

- Assume the role and, in response, get temporary security credentials.
- Using the temporary security credentials, access the objects in the bucket.

For more information about IAM roles, go to [IAM Roles](#) in *IAM User Guide*.

The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy granting Account B conditional permission to upload objects.
2. Account A administrator creates an IAM role, establishing trust with Account C, so users in that account can access Account A. The access policy attached to the role limits what user in Account C can do when the user accesses Account A.
3. Account B administrator uploads an object to the bucket owned by Account A, granting full-control permission to the bucket owner.
4. Account C administrator creates a user and attaches a user policy that allows the user to assume the role.
5. User in Account C first assumes the role, which returns the user temporary security credentials. Using those temporary credentials, the user then accesses objects in the bucket.

For this example, you need three accounts. The following table shows how we refer to these accounts and the administrator users in these accounts. Per IAM guidelines (see [About Using an Administrator User to Create Resources and Grant Permissions \(p. 316\)](#)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions

AWS Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAdmin
2222-2222-2222	Account B	AccountBAdmin
3333-3333-3333	Account C	AccountCAdmin

Step 0: Preparing for the Walkthrough

Note

You may want to open a text editor and write down some of the information as you walk through the steps. In particular, you will need account IDs, canonical user IDs, IAM User Sign-in URLs for each account to connect to the console, and Amazon Resource Names (ARNs) of the IAM users, and roles.

- Make sure you have three AWS accounts and each account has one administrator user as shown in the table in the preceding section.
 - Sign up for AWS accounts, as needed. We refer to these accounts as Account A, Account B, and Account C.
 - Go to <https://aws.amazon.com/s3/> and click **Create an AWS Account**.
 - Follow the on-screen instructions.

AWS will notify you by email when your account is active and available for you to use.
 - Using Account A credentials, sign in to the [IAM console](#) and do the following to create an administrator user:
 - Create user AccountAdmin and note down security credentials. For more information about adding users, see [Creating an IAM User in Your AWS Account](#) in the *IAM User Guide*.
 - Grant AccountAdmin administrator privileges by attaching a user policy giving full access. For instructions, see [Working with Policies](#) in the *IAM User Guide*.
 - In the IAM Console **Dashboard**, note down the **IAM User Sign-In URL**. Users in this account must use this URL when signing in to the AWS Management Console. For more information, go to [How Users Sign In to Your Account](#) in *IAM User Guide*.
 - Repeat the preceding step to create administrator users in Account B and Account C.
- For Account C, note down the canonical user ID.

When you create an IAM role in Account A, the trust policy grants Account C permission to assume the role by specifying the account ID. You can find account information as follows:

- Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [Amazon S3 Console](#).
 - Choose the name of an Amazon S3 bucket to view the details about that bucket.
 - Choose the **Permissions** tab and then choose **Access Control List**.
 - In the **Access for your AWS account** section, in the **Account** column is a long identifier, such as c1daexampleaaf850ea79cf0430f33d72579fd1611c97f7ded193374c0b163b6. This is your canonical user ID.
- When creating a bucket policy, you will need the following information. Note down these values:
 - Canonical user ID of Account A** – When the Account A administrator grants conditional upload object permission to the Account B administrator, the condition specifies the canonical user ID of the Account A user that must get full-control of the objects.

Note

The canonical user ID is the Amazon S3-only concept. It is a 64-character obfuscated version of the account ID.

- **User ARN for Account B administrator** – You can find the user ARN in the IAM console. You will need to select the user and find the user's ARN in the **Summary** tab.

In the bucket policy, you grant AccountBadmin permission to upload objects and you specify the user using the ARN. Here's an example ARN value:

```
arn:aws:iam::AccountB-ID:user/AccountBadmin
```

4. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.

For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

Step 1: Do the Account A Tasks

In this example, Account A is the bucket owner. So user AccountAadmin in Account A will create a bucket, attach a bucket policy granting the Account B administrator permission to upload objects, create an IAM role granting Account C permission to assume the role so it can access objects in the bucket.

Step 1.1: Sign In to the AWS Management Console

Using the IAM User Sign-in URL for Account A, first sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a Bucket and Attach a Bucket Policy

In the Amazon S3 console, do the following:

1. Create a bucket. This exercise assumes the bucket name is `examplebucket`.

For instructions, see [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

2. Attach the following bucket policy granting conditional permission to the Account B administrator permission to upload objects.

You need to update the policy by providing your own values for `examplebucket`, `AccountB-ID`, and the `CanonicalUserId-of-AWSaccountA-BucketOwner`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "111",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::examplebucket/*"
    }
  ],
}
```

```
{
  "Sid": "112",
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
  },
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3::examplebucket/*",
  "Condition": {
    "StringNotEquals": {
      "s3:x-amz-grant-full-control": "id=CanonicalUserId-of-AWSaccountA-BucketOwner"
    }
  }
}
```

Step 1.3: Create an IAM Role to Allow Account C Cross-Account Access in Account A

In the IAM console, create an IAM role ("examplerole") that grants Account C permission to assume the role. Make sure you are still signed in as the Account A administrator because the role must be created in Account A.

1. Before creating the role, prepare the managed policy that defines the permissions that the role requires. You attach this policy to the role in a later step.
 - a. In the navigation pane on the left, click **Policies** and then click **Create Policy**.
 - b. Next to **Create Your Own Policy**, click **Select**.
 - c. Enter `access-accountA-bucket` in the **Policy Name** field.
 - d. Copy the following access policy and paste it into the **Policy Document** field. The access policy grants the role `s3:GetObject` permission so when Account C user assumes the role, it can only perform the `s3:GetObject` operation.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3::examplebucket/*"
    }
  ]
}
```

- e. Click **Create Policy**.

The new policy appears in the list of managed policies.

2. In the navigation pane on the left, click **Roles** and then click **Create New Role**.
3. Under **Select Role Type**, select **Role for Cross-Account Access**, and then click the **Select** button next to **Provide access between AWS accounts you own**.
4. Enter the Account C account ID.

For this walkthrough you do not need to require users to have multi-factor authentication (MFA) to assume the role, so leave that option unselected.

5. Click **Next Step** to set the permissions that will be associated with the role.
6. Select the box next to the `access-accountA-bucket` policy that you created and then click **Next Step**.

The Review page appears so you can confirm the settings for the role before it's created. One very important item to note on this page is the link that you can send to your users who need to use this role. Users who click the link go straight to the Switch Role page with the Account ID and Role Name fields already filled in. You can also see this link later on the Role Summary page for any cross-account role.

7. Enter `examplerole` for the role name, and then click **Next Step**.
8. After reviewing the role, click **Create Role**.

The `examplerole` role is displayed in the list of roles.

9. Click the role name `examplerole`.
10. Select the **Trust Relationships** tab.
11. Click **Show policy document** and verify the trust policy shown matches the following policy.

The following trust policy establishes trust with Account C, by allowing it the `sts:AssumeRole` action. For more information, go to [AssumeRole](#) in the *AWS Security Token Service API Reference*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountC-ID:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

12. Note down the Amazon Resource Name (ARN) of the `examplerole` role you created.

Later in the following steps, you attach a user policy to allow an IAM user to assume this role, and you identify the role by the ARN value.

Step 2: Do the Account B Tasks

The `examplebucket` owned by Account A needs objects owned by other accounts. In this step, the Account B administrator uploads an object using the command line tools.

- Using the `put-object` AWS CLI command, upload an object to the `examplebucket`.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body HappyFace.jpg --
grant-full-control id="canonicalUserId-ofTheBucketOwner" --profile AccountBadmin
```

Note the following:

- The `--Profile` parameter specifies `AccountBadmin` profile, so the object is owned by Account B.
- The parameter `grant-full-control` grants the bucket owner full-control permission on the object as required by the bucket policy.
- The `--body` parameter identifies the source file to upload. For example, if the file is on the C: drive of a Windows computer, you specify `c:\HappyFace.jpg`.

Step 3: Do the Account C Tasks

In the preceding steps, Account A has already created a role, `examplerole`, establishing trust with Account C. This allows users in Account C to access Account A. In this step, Account C administrator creates a user (Dave) and delegates him the `sts:AssumeRole` permission it received from Account A. This will allow Dave to assume the `examplerole` and temporarily gain access to Account A. The access policy that Account A attached to the role will limit what Dave can do when he accesses Account A—specifically, get objects in `examplebucket`.

Step 3.1: Create a User in Account C and Delegate Permission to Assume `examplerole`

1. Using the IAM user sign-in URL for Account C, first sign in to the AWS Management Console as AccountCAdmin user.
2. In the IAM console, create a user Dave.

For instructions, see [Creating IAM Users \(AWS Management Console\)](#) in the *IAM User Guide*.

3. Note down the Dave credentials. Dave will need these credentials to assume the `examplerole` role.
4. Create an inline policy for the Dave IAM user to delegate the `sts:AssumeRole` permission to Dave on the `examplerole` role in account A.
 - a. In the navigation pane on the left, click **Users**.
 - b. Click the user name Dave.
 - c. On the user details page, select the **Permissions** tab and then expand the **Inline Policies** section.
 - d. Choose **click here (or Create User Policy)**.
 - e. Click **Custom Policy**, and then click **Select**.
 - f. Enter a name for the policy in the **Policy Name** field.
 - g. Copy the following policy into the **Policy Document** field.

You will need to update the policy by providing the Account A ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["sts:AssumeRole"],
      "Resource": "arn:aws:iam::AccountA-ID:role/examplerole"
    }
  ]
}
```

- h. Click **Apply Policy**
5. Save Dave's credentials to the config file of the AWS CLI by adding another profile, AccountCDave.

```
[profile AccountCDave]
aws_access_key_id = UserDaveAccessKeyID
aws_secret_access_key = UserDaveSecretAccessKey
region = us-west-2
```

Step 3.2: Assume Role (`examplerole`) and Access Objects

Now Dave can access objects in the bucket owned by Account A as follows:

- Dave first assumes the `examplerole` using his own credentials. This will return temporary credentials.

- Using the temporary credentials, Dave will then access objects in Account A's bucket.
1. At the command prompt, execute the following AWS CLI `assume-role` command using the AccountCDave profile.

You will need to update the ARN value in the command by providing the Account A ID where `examplerole` is defined.

```
aws sts assume-role --role-arn arn:aws:iam::accountA-ID:role/examplerole --profile AccountCDave --role-session-name test
```

In response, AWS Security Token Service (STS) returns temporary security credentials (access key ID, secret access key, and a session token).

2. Save the temporary security credentials in the AWS CLI config file under the `TempCred` profile.

```
[profile TempCred]
aws_access_key_id = temp-access-key-ID
aws_secret_access_key = temp-secret-access-key
aws_session_token = session-token
region = us-west-2
```

3. At the command prompt, execute the following AWS CLI command to access objects using the temporary credentials. For example, the command specifies the `head-object` API to retrieve object metadata for the `HappyFace.jpg` object.

```
aws s3api get-object --bucket examplebucket --key HappyFace.jpg SaveFileAs.jpg --profile TempCred
```

Because the access policy attached to `examplerole` allows the actions, Amazon S3 processes the request. You can try any other action on any other object in the bucket.

If you try any other action—for example, `get-object-acl`—you will get permission denied because the role is not allowed that action.

```
aws s3api get-object-acl --bucket examplebucket --key HappyFace.jpg --profile TempCred
```

We used user Dave to assume the role and access the object using temporary credentials. It could also be an application in Account C that accesses objects in `examplebucket`. The application can obtain temporary security credentials, and Account C can delegate the application permission to assume `examplerole`.

Step 4: Clean Up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the AWS Management Console ([AWS Management Console](#)) using account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to *examplebucket*. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the `examplerole` you created in Account A.
 - In the IAM console, remove the AccountAadmin user.

2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.
3. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account C credentials. In the IAM console, delete user AccountCadmin and user Dave.

Related Resources

- [Creating a Role to Delegate Permissions to an IAM User](#) in the *IAM User Guide*.
- [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#) in the *IAM User Guide*.
- [Working with Policies](#) in the *IAM User Guide*.

Using Bucket Policies and User Policies

Bucket policy and user policy are two of the access policy options available for you to grant permission to your Amazon S3 resources. Both use JSON-based access policy language. The topics in this section describe the key policy language elements, with emphasis on Amazon S3–specific details, and provide example bucket and user policies.

Important

We recommend that you first review the introductory topics that explain the basic concepts and options available for you to manage access to your Amazon S3 resources. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 301\)](#).

Topics

- [Access Policy Language Overview \(p. 341\)](#)
- [Bucket Policy Examples \(p. 371\)](#)
- [User Policy Examples \(p. 380\)](#)

Access Policy Language Overview

The topics in this section describe the basic elements used in bucket and user policies as used in Amazon S3. For complete policy language information, see the [Overview of IAM Policies](#) and the [AWS IAM Policy Reference](#) topics in the *IAM User Guide*.

Note

Bucket policies are limited to 20 KB in size.

Common Elements in an Access Policy

In its most basic sense, a policy contains the following elements:

- **Resources** – Buckets and objects are the Amazon S3 resources for which you can allow or deny permissions. In a policy, you use the Amazon Resource Name (ARN) to identify the resource.
- **Actions** – For each resource, Amazon S3 supports a set of operations. You identify resource operations that you will allow (or deny) by using action keywords (see [Specifying Permissions in a Policy \(p. 345\)](#)).

For example, the `s3:ListBucket` permission allows the user permission to the Amazon S3 [GET Bucket \(List Objects\)](#) operation.

- **Effect** – What the effect will be when the user requests the specific action—this can be either allow or deny.

If you do not explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do in order to make sure that a user cannot access it, even if a different policy grants access.

- **Principal** – The account or user who is allowed access to the actions and resources in the statement. In a bucket policy, the principal is the user, account, service, or other entity who is the recipient of this permission.

The following example bucket policy shows the preceding common policy elements. The policy allows Dave, a user in account `Account-ID`, `s3:GetObject`, `s3:GetBucketLocation`, and `s3:ListBucket` Amazon S3 permissions on the `examplebucket` bucket.

```
{
  "Version": "2012-10-17",
  "Id": "ExamplePolicy01",
  "Statement": [
```

```
{
  "Sid": "ExampleStatement01",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::Account-ID:user/Dave"
  },
  "Action": [
    "s3:GetObject",
    "s3:GetBucketLocation",
    "s3:ListBucket"
  ],
  "Resource": [
    "arn:aws:s3:::examplebucket/*",
    "arn:aws:s3:::examplebucket"
  ]
}
```

For more information about the access policy elements, see the following topics:

- [Specifying Resources in a Policy \(p. 342\)](#)
- [Specifying a Principal in a Policy \(p. 343\)](#)
- [Specifying Permissions in a Policy \(p. 345\)](#)
- [Specifying Conditions in a Policy \(p. 350\)](#)

The following topics provide additional policy examples:

- [Bucket Policy Examples \(p. 371\)](#)
- [User Policy Examples \(p. 380\)](#)

Specifying Resources in a Policy

The following is the common Amazon Resource Name (ARN) format to identify any resources in AWS.

```
arn:partition:service:region:namespace:relative-id
```

For your Amazon S3 resources:

- `aws` is a common partition name. If your resources are in the China (Beijing) Region, `aws-cn` is the partition name.
- `s3` is the service.
- You don't specify Region and namespace.
- For Amazon S3, it can be a `bucket-name` or a `bucket-name/object-key`. You can use wild card.

Then the ARN format for Amazon S3 resources reduces to the following:

```
arn:aws:s3:::bucket_name
arn:aws:s3:::bucket_name/key_name
```

The following are examples of Amazon S3 resource ARNs.

- This ARN identifies the `/developers/design_info.doc` object in the `examplebucket` bucket.

```
arn:aws:s3:::examplebucket/developers/design_info.doc
```


- You can use wildcards as part of the resource ARN. You can use wildcard characters (* and ?) within any ARN segment (the parts separated by colons). An asterisk (*) represents any combination of zero or more characters, and a question mark (?) represents any single character. You can use multiple * or ? characters in each segment, but a wildcard cannot span segments.
- This ARN uses the wildcard * in the relative-ID part of the ARN to identify all objects in the `examplebucket` bucket.

```
arn:aws:s3:::examplebucket/*
```

This ARN uses * to indicate all Amazon S3 resources (all S3 buckets and objects in your account).

```
arn:aws:s3:::*
```

- This ARN uses both wildcards, * and ?, in the relative-ID part. It identifies all objects in buckets such as `example1bucket`, `example2bucket`, `example3bucket`, and so on.

```
arn:aws:s3:::example?bucket/*
```

- You can use policy variables in Amazon S3 ARNs. At policy evaluation time, these predefined variables are replaced by their corresponding values. Suppose that you organize your bucket as a collection of folders, one folder for each of your users. The folder name is the same as the user name. To grant users permission to their folders, you can specify a policy variable in the resource ARN:

```
arn:aws:s3:::bucket_name/developers/${aws:username}/
```

At runtime, when the policy is evaluated, the variable `${aws:username}` in the resource ARN is substituted with the user name making the request.

To find the ARN for an S3 bucket, you can look at the Amazon S3 console **Bucket Policy** or **CORS configuration** permissions pages. For more information, see [How Do I Add an S3 Bucket Policy?](#) or [How Do I Allow Cross-Domain Resource Sharing with CORS?](#) in the *Amazon Simple Storage Service Console User Guide*.

For more information about ARNs, see the following:

- [Resource](#) in the *IAM User Guide*
- [IAM Policy Variables Overview](#) in the *IAM User Guide*
- [ARNs](#) in the *AWS General Reference*

For more information about other access policy language elements, see [Access Policy Language Overview](#) (p. 341).

Specifying a Principal in a Policy

The `Principal` element specifies the user, account, service, or other entity that is allowed or denied access to a resource. The following are examples of specifying `Principal`. For more information, see [Principal](#) in the *IAM User Guide*.

- To grant permissions to an AWS account, identify the account using the following format.

```
"AWS" : "account-ARN"
```

For example:

```
"Principal":{"AWS":["arn:aws:iam::AccountNumber-WithoutHyphens:root"]}
```

Amazon S3 also supports a canonical user ID, which is an obfuscated form of the AWS account ID. You can specify this ID using the following format.

```
"CanonicalUser":["64-digit-alphanumeric-value"]
```

For example:

```
"Principal":{"CanonicalUser":["64-digit-alphanumeric-value"]}
```

For information about how to find the canonical user ID for your account, see [Finding Your Account Canonical User ID](#).

Important

When you use a canonical user ID in a policy, Amazon S3 might change the canonical ID to the corresponding AWS account ID. This does not impact the policy because both of these IDs identify the same account.

- To grant permission to an IAM user within your account, you must provide an "AWS": "*user-ARN*" name-value pair.

```
"Principal":{"AWS":["arn:aws:iam::account-number-without-hyphens:user/username"]}
```

- To grant permission to everyone, also referred as anonymous access, you set the wildcard, "*", as the `Principal` value. For example, if you configure your bucket as a website, you want all the objects in the bucket to be publicly accessible. The following are equivalent:

```
"Principal":["*"]
```

```
"Principal":{"AWS":["*"]}
```

Warning

Use caution when granting anonymous access to your S3 bucket. When you grant anonymous access, anyone in the world can access your bucket. We highly recommend that you never grant any kind of anonymous write access to your S3 bucket.

- You can require that your users access your Amazon S3 content by using Amazon CloudFront URLs (instead of Amazon S3 URLs). To do this, create a CloudFront origin access identity (OAI), and then change the permissions either on your bucket or on the objects in your bucket. The format for specifying the OAI in a `Principal` statement is as follows:

```
"Principal":{"CanonicalUser":["Amazon S3 Canonical User ID assigned to origin access identity"]}
```

For more information, see [Using an Origin Access Identity to Restrict Access to Your Amazon S3 Content](#) in the *Amazon CloudFront Developer Guide*.

For more information about other access policy language elements, see [Access Policy Language Overview \(p. 341\)](#).

Specifying Permissions in a Policy

Amazon S3 defines a set of permissions that you can specify in a policy. These are keywords, each of which maps to specific Amazon S3 operations (see [Operations on Buckets](#), and [Operations on Objects](#) in the *Amazon Simple Storage Service API Reference*).

Topics

- [Permissions for Object Operations](#) (p. 345)
- [Permissions Related to Bucket Operations](#) (p. 346)
- [Permissions Related to Bucket Subresource Operations](#) (p. 347)
- [Permissions Related to Account Operations](#) (p. 349)

Permissions for Object Operations

This section provides a list of the permissions for object operations that you can specify in a policy.

Amazon S3 Permissions for Object Operations

Permissions	Amazon S3 Operations
s3:AbortMultipartUpload	Abort Multipart Upload
s3:BypassGovernanceRetention	PUT Object Retention , PUT Object , DELETE Object
s3:DeleteObject	DELETE Object
s3:DeleteObjectTagging	DELETE Object tagging
s3:DeleteObjectVersion	DELETE Object (a Specific Version of the Object)
s3:DeleteObjectVersionTagging	DELETE Object tagging (for a Specific Version of the Object)
s3:GetObject	GET Object , HEAD Object , SELECT Object Content When you grant this permission on a version-enabled bucket, you always get the latest version data.
s3:GetObjectAcl	GET Object ACL
s3:GetObjectLegalHold	GET Object Legal Hold , GET Object
s3:GetObjectRetention	Get Object Retention , GET Object
s3:GetObjectTagging	GET Object tagging
s3:GetObjectTorrent	GET Object torrent
s3:GetObjectVersion	GET Object , HEAD Object To grant permission for version-specific object data, you must grant this permission. That is, when you specify version number when making any of these requests, you need this Amazon S3 permission.
s3:GetObjectVersionAcl	GET ACL (for a Specific Version of the Object)
s3:GetObjectVersionTagging	GET Object tagging (for a Specific Version of the Object)
s3:GetObjectVersionTorrent	GET Object Torrent versioning
s3:ListMultipartUploadParts	List Parts

Permissions	Amazon S3 Operations
s3:PutObject	PUT Object , POST Object , Initiate Multipart Upload , Upload Part , Complete Multipart Upload , PUT Object - Copy
s3:PutObjectAcl	PUT Object ACL
s3:PutObjectLegalHold	PUT Object Legal Hold , PUT Object
s3:PutObjectRetention	PUT Object Retention , PUT Object
s3:PutObjectTagging	PUT Object tagging
s3:PutObjectVersionAcl	PUT Object ACL (for a Specific Version of the Object)
s3:PutObjectVersionTagging	PUT Object tagging (for a Specific Version of the Object)
s3:RestoreObject	POST Object restore

The following example bucket policy grants the `s3:PutObject` and the `s3:PutObjectAcl` permissions to a user (Dave). If you remove the `Principal` element, you can attach the policy to a user. These are object operations, and accordingly the relative-id portion of the `Resource` ARN identifies objects (examplebucket/*). For more information, see [Specifying Resources in a Policy](#) (p. 342).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountB-ID:user/Dave"
      },
      "Action": [ "s3:PutObject", "s3:PutObjectAcl" ],
      "Resource": "arn:aws:s3:::examplebucket/*"
    }
  ]
}
```

You can use a wildcard to grant permission for all Amazon S3 actions.

```
"Action":    ""
```

Permissions Related to Bucket Operations

This section provides a list of the permissions related to bucket operations that you can specify in a policy.

Amazon S3 Permissions Related to Bucket Operations

Permission Keywords	Amazon S3 Operation(s) Covered
s3:CreateBucket	PUT Bucket
s3>DeleteBucket	DELETE Bucket
s3:ListBucket	GET Bucket (List Objects) , HEAD Bucket
s3:ListBucketVersions	GET Bucket Object versions

Permission Keywords	Amazon S3 Operation(s) Covered
s3:ListAllMyBuckets	GET Service
s3:ListBucketMultipartUploads	GET Multipart Uploads

The following example user policy grants the `s3:CreateBucket`, `s3:ListAllMyBuckets`, and the `s3:GetBucketLocation` permissions to a user. Note that for all these permissions, you set the relative-id part of the Resource ARN to `"*"`. For all other bucket actions, you must specify a bucket name. For more information, see [Specifying Resources in a Policy \(p. 342\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3::*"
      ]
    }
  ]
}
```

If your user is going to use the console to view buckets and see the contents of any of these buckets, the user must have the `s3:ListAllMyBuckets` and `s3:GetBucketLocation` permissions. For an example, see "Policy for Console Access" at [Writing IAM Policies: How to Grant Access to an S3 Bucket](#).

Permissions Related to Bucket Subresource Operations

This section provides a list of the permissions related to bucket subresource operations that you can specify in a policy.

Amazon S3 Permissions Related to Bucket Subresource Operations

Permissions	Amazon S3 Operation(s) Covered
s3:DeleteBucketPolicy	DELETE Bucket policy
s3:DeleteBucketWebsite	DELETE Bucket website
s3:GetAccelerateConfiguration	GET Bucket accelerate
s3:GetAnalyticsConfiguration	GET Bucket analytics, List Bucket Analytics Configurations
s3:GetBucketAcl	GET Bucket acl
s3:GetBucketCORS	GET Bucket cors
s3:GetBucketLocation	GET Bucket location
s3:GetBucketLogging	GET Bucket logging
s3:GetBucketNotification	GET Bucket notification

Permissions	Amazon S3 Operation(s) Covered
s3:GetBucketObjectLockConfiguration	GET Bucket Object Lock configuration
s3:GetBucketPolicy	GET Bucket policy
s3:GetBucketPolicyStatus	GET BucketPolicyStatus
s3:GetBucketPublicAccessBlock	GET PublicAccessBlock
s3:GetBucketRequestPayment	GET Bucket requestPayment
s3:GetBucketTagging	GET Bucket tagging
s3:GetBucketVersioning	GET Bucket versioning
s3:GetBucketWebsite	GET Bucket website
s3:GetEncryptionConfiguration	GET Bucket encryption
s3:GetInventoryConfiguration	GET Bucket inventory, List Bucket Inventory Configurations
s3:GetLifecycleConfiguration	GET Bucket lifecycle
s3:GetMetricsConfiguration	GET Bucket metrics, List Bucket Metrics Configurations
s3:GetReplicationConfiguration	GET Bucket replication
s3:PutAccelerateConfiguration	PUT Bucket accelerate
s3:PutAnalyticsConfiguration	PUT Bucket analytics, DELETE Bucket analytics
s3:PutBucketAcl	PUT Bucket acl
s3:PutBucketCORS	PUT Bucket cors, DELETE Bucket cors
s3:PutBucketLogging	PUT Bucket logging
s3:PutBucketNotification	PUT Bucket notification
s3:PutBucketObjectLockConfiguration	PUT Bucket Object Lock configuration
s3:PutBucketPolicy	PUT Bucket policy
s3:PutBucketPublicAccessBlock	PUT PublicAccessBlock, DELETE PublicAccessBlock
s3:PutBucketRequestPayment	PUT Bucket requestPayment
s3:PutBucketTagging	DELETE Bucket tagging, PUT Bucket tagging
s3:PutBucketVersioning	PUT Bucket versioning
s3:PutBucketWebsite	PUT Bucket website
s3:PutEncryptionConfiguration	PUT Bucket encryption, DELETE Bucket encryption
s3:PutInventoryConfiguration	PUT Bucket inventory, DELETE Bucket inventory
s3:PutLifecycleConfiguration	PUT Bucket lifecycle, DELETE Bucket lifecycle
s3:PutMetricsConfiguration	PUT Bucket metrics, DELETE Bucket metrics
s3:PutReplicationConfiguration	PUT Bucket replication, DELETE Bucket replication

The following user policy grants the `s3:GetBucketAcl` permission on the `examplebucket` bucket to user Dave.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::Account-ID:user/Dave"
      },
      "Action": [
        "s3:GetObjectVersion",
        "s3:GetBucketAcl"
      ],
      "Resource": "arn:aws:s3:::examplebucket"
    }
  ]
}
```

You can delete objects either by explicitly calling the DELETE Object API or by configuring its lifecycle (see [Object Lifecycle Management \(p. 119\)](#)) so that Amazon S3 can remove the objects when their lifetime expires. To explicitly block users or accounts from deleting objects, you must explicitly deny them `s3:DeleteObject`, `s3:DeleteObjectVersion`, and `s3:PutLifecycleConfiguration` permissions. By default, users have no permissions. But as you create users, add users to groups, and grant them permissions, it is possible for users to get certain permissions that you did not intend to give. That is where you can use explicit deny, which supersedes all other permissions a user might have and denies the user permissions for specific actions.

Permissions Related to Account Operations

This section provides a list of the permissions related to account operations that you can specify in a policy.

Amazon S3 Permissions Related to Account Operations

Permission Keywords	Amazon S3 Operation(s) Covered
<code>s3:CreateJob</code>	CreateJob
<code>s3:DescribeJob</code>	DescribeJob
<code>s3:GetAccountPublicAccessBlock</code>	GET PublicAccessBlock
<code>s3:ListJobs</code>	ListJobs
<code>s3:PutAccountPublicAccessBlock</code>	PUT PublicAccessBlock , DELETE PublicAccessBlock
<code>s3:UpdateJobPriority</code>	UpdateJobPriority
<code>s3:UpdateJobStatus</code>	UpdateJobStatus

The following example user policy grants the `s3:GetAccountPublicAccessBlock` permission to a user. Note that for these permissions, you set the Resource value to `"*"`. For more information, see [Specifying Resources in a Policy \(p. 342\)](#).

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "statement1",
    "Effect": "Allow",
    "Action": [
      "s3:GetAccountPublicAccessBlock"
    ],
    "Resource": [
      "*"
    ]
  }
]
```

Specifying Conditions in a Policy

The access policy language allows you to specify conditions when granting permissions. The `Condition` element (or `Condition` block) lets you specify conditions for when a policy is in effect. In the `Condition` element, which is optional, you build expressions in which you use Boolean operators (equal, less than, etc.) to match your condition against values in the request. For example, when granting a user permission to upload an object, the bucket owner can require the object be publicly readable by adding the `StringEquals` condition as shown here:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": [
            "public-read"
          ]
        }
      }
    }
  ]
}
```

The `Condition` block specifies the `StringEquals` condition that is applied to the specified key-value pair, `"s3:x-amz-acl": ["public-read"]`. There is a set of predefined keys you can use in expressing a condition. The example uses the `s3:x-amz-acl` condition key. This condition requires user to include the `x-amz-acl` header with value `public-read` in every `PUT` object request.

For more information about specifying conditions in an access policy language, see [Condition](#) in the *IAM User Guide*.

The following topics describe AWS-wide and Amazon S3-specific condition keys and provide example policies.

Topics

- [Available Condition Keys \(p. 351\)](#)

- [Amazon S3 Condition Keys for Object Operations \(p. 353\)](#)
- [Amazon S3 Condition Keys for Bucket Operations \(p. 365\)](#)

Available Condition Keys

The predefined keys available for specifying conditions in an Amazon S3 access policy can be classified as follows:

- **AWS-wide keys** – AWS provides a set of common keys that are supported by all AWS services that support policies. These keys that are common to all services are called AWS-wide keys and use the prefix `aws:`. For a list of AWS-wide keys, see [Available Keys for Conditions](#) in the *IAM User Guide*. There are also keys that are specific to Amazon S3, which use the prefix `s3:`. Amazon S3-specific keys are discussed in the next bulleted item.

The new condition keys `aws:sourceVpce` and `aws:sourceVpc` are used in bucket policies for VPC endpoints. For examples of using these condition keys, see [Example Bucket Policies for VPC Endpoints for Amazon S3 \(p. 378\)](#).

The following example bucket policy allows authenticated users permission to use the `s3:GetObject` action if the request originates from a specific range of IP addresses (192.168.143.*), unless the IP address is 192.168.143.188. In the condition block, the `IpAddress` and the `NotIpAddress` are conditions, and each condition is provided a key-value pair for evaluation. Both the key-value pairs in this example use the `aws:SourceIp` AWS-wide key.

Note

The `IpAddress` and `NotIpAddress` key values specified in the condition uses CIDR notation as described in RFC 4632. For more information, go to <http://www.rfc-editor.org/rfc/rfc4632.txt>.

```
{
  "Version": "2012-10-17",
  "Id": "S3PolicyId1",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": "192.168.143.0/24"
        },
        "NotIpAddress": {
          "aws:SourceIp": "192.168.143.188/32"
        }
      }
    }
  ]
}
```

- **Amazon S3-specific keys** – In addition to the AWS-wide keys, the following are the condition keys that are applicable only in the context of granting Amazon S3 specific permissions. These Amazon S3-specific keys use the prefix `s3:`.
 - `s3:x-amz-acl`
 - `s3:x-amz-copy-source`
 - `s3:x-amz-metadata-directive`

- `s3:x-amz-server-side-encryption`
- `s3:VersionId`
- `s3:LocationConstraint`
- `s3:delimiter`
- `s3:max-keys`
- `s3:prefix`
- `s3:x-amz-server-side-encryption-aws-kms-key-id`
- `s3:ExistingObjectTag/<tag-key>`

For example policies using object tags related condition keys, see [Object Tagging and Access Control Policies](#) (p. 113).

- `s3:RequestObjectTagKeys`
- `s3:RequestObjectTag/<tag-key>`
- `s3:object-lock-remaining-retention-days`
- `s3:object-lock-mode`
- `s3:object-lock-retain-until-date`
- `s3:object-lock-legal-hold`

For example, the following bucket policy allows the `s3:PutObject` permission for two AWS accounts if the request includes the `x-amz-acl` header making the object publicly readable.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AddCannedAcl",
      "Effect": "Allow",
      "Principal": {
        "AWS": [ "arn:aws:iam::account1-ID:root", "arn:aws:iam::account2-ID:root" ]
      },
      "Action": [ "s3:PutObject" ],
      "Resource": [ "arn:aws:s3:::examplebucket/*" ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": [ "public-read" ]
        }
      }
    }
  ]
}
```

The `Condition` block uses the `StringEquals` condition, and it is provided a key-value pair, `"s3:x-amz-acl": ["public-read"]`, for evaluation. In the key-value pair, the `s3:x-amz-acl` is an Amazon S3-specific key, as indicated by the prefix `s3:`.

Important

Not all conditions make sense for all actions. For example, it makes sense to include an `s3:LocationConstraint` condition on a policy that grants the `s3:CreateBucket` Amazon S3 permission, but not for the `s3:GetObject` permission. Amazon S3 can test for semantic errors of this type that involve Amazon S3-specific conditions. However, if you are creating a policy for an IAM user and you include a semantically invalid Amazon S3 condition, no error is reported, because IAM cannot validate Amazon S3 conditions.

The following section describes the condition keys that can be used to grant conditional permission for bucket and object operations. In addition, there are condition keys related to Amazon S3 Signature Version 4 authentication. For more information, go to [Amazon S3 Signature Version 4 Authentication Specific Policy Keys](#) in the *Amazon Simple Storage Service API Reference*.

Amazon S3 Condition Keys for Object Operations

The following table shows which Amazon S3 conditions you can use with which Amazon S3 actions. Example policies are provided following the table. Note the following about the Amazon S3–specific condition keys described in the following table:

- The condition key names are preceded by the prefix `s3:`. For example, `s3:x-amz-acl`.
- Each condition key maps to the same name request header allowed by the API on which the condition can be set. That is, these condition keys dictate behavior of the same name request headers. For example:
 - The condition key `s3:x-amz-acl` that you can use to grant condition permission for the `s3:PutObject` permission defines behavior of the `x-amz-acl` request header that the PUT Object API supports.
 - The condition key `s3:VersionId` that you can use to grant conditional permission for the `s3:GetObjectVersion` permission defines behavior of the `versionId` query parameter that you set in a GET Object request.

Permission	Applicable Condition Keys (or keywords)	Description
<code>s3:PutObject</code>	<ul style="list-style-type: none"> • <code>s3:x-amz-acl</code> (for canned ACL permissions) • <code>s3:x-amz-grant-permission</code> (for explicit permissions), where <i>permission</i> can be: <code>read, write, read-acp, write-acp, full-control</code> 	<p>The PUT Object operation allows access control list (ACL)–specific headers that you can use to grant ACL-based permissions. Using these keys, the bucket owner can set a condition to require specific access permissions when the user uploads an object.</p> <p>For an example policy, see Example 1: Granting s3:PutObject Permission with a Condition Requiring the Bucket Owner to Get Full Control (p. 360).</p> <p>For more information about ACLs, see Access Control List (ACL) Overview (p. 403).</p>
	<code>s3:x-amz-copy-source</code>	<p>To copy an object, you use the PUT Object API (see PUT Object) and specify the source using the <code>x-amz-copy-source</code> header. Using this key, the bucket owner can restrict the copy source to a specific bucket, a specific folder in the bucket, or a specific object in a bucket.</p> <p>For a policy example, see Example 3: Granting s3:PutObject Permission to</p>

Permission	Applicable Condition Keys (or keywords)	Description
		Copy Objects with a Restriction on the Copy Source (p. 363) .
	<code>s3:x-amz-server-side-encryption</code>	<p>When you upload an object, you can use the <code>x-amz-server-side-encryption</code> header to request Amazon S3 to encrypt the object when it is saved, using an envelope encryption key managed either by AWS Key Management Service (AWS KMS) or by Amazon S3 (see Protecting Data Using Server-Side Encryption (p. 265)).</p> <p>When granting the <code>s3:PutObject</code> permission, the bucket owner can add a condition using this key to require the user to specify this header in the request. A bucket owner can grant such conditional permission to ensure that objects the user uploads are encrypted when they are saved.</p> <p>For a policy example, see Example 1: Granting s3:PutObject Permission with a Condition Requiring the Bucket Owner to Get Full Control (p. 360).</p>
	<code>s3:x-amz-server-side-encryption-aws-kms-key-id</code>	<p>When you upload an object, you can use the <code>x-amz-server-side-encryption-aws-kms-key-id</code> header to request Amazon S3 to encrypt the object using the specified AWS KMS key when it is saved (see Protecting Data Using Server-Side Encryption with keys stored in AWS KMS(SSE-KMS) (p. 265)).</p> <p>When granting the <code>s3:PutObject</code> permission, the bucket owner can add a condition using this key to restrict the AWS KMS key ID used for object encryption to a specific value.</p> <p>A bucket owner can grant such conditional permission to ensure that objects the user uploads are encrypted with a specific key when they are saved.</p> <p>The AWS KMS key you specify in the policy must use the following format:</p> <p><code>arn:aws:kms:region:acct-id:key/key-id</code></p>

Permission	Applicable Condition Keys (or keywords)	Description
	<code>s3:x-amz-metadata-directive</code>	<p>When you copy an object using the PUT Object API (see PUT Object), you can optionally add the <code>x-amz-metadata-directive</code> header to specify whether you want the object metadata copied from the source object or replaced with metadata provided in the request.</p> <p>Using this key bucket, an owner can add a condition to enforce certain behavior when objects are uploaded.</p> <p>Valid values: <code>COPY</code> <code>REPLACE</code>. The default is <code>COPY</code>.</p>
	<code>s3:x-amz-storage-class</code>	<p>By default <code>s3:PutObject</code> stores objects using the <code>STANDARD</code> storage class, but you can use the <code>x-amz-storage-class</code> request header to specify a different storage class.</p> <p>When granting the <code>s3:PutObject</code> permission, you can use the <code>s3:x-amz-storage-class</code> condition key to restrict which storage class to use when storing uploaded objects. For more information about storage classes, see Storage Classes.</p> <p>For an example policy, see Example 5: Restricting Object Uploads to Objects with a Specific Storage Class (p. 365).</p> <p>For valid values, see Amazon S3 PUT Object Requests.</p>
	<ul style="list-style-type: none"> <code>s3:RequestObjectTagKeys</code> <code>s3:RequestObjectTag/<tag-key></code> 	<p>Using this condition key, you can limit permission for the <code>s3:PutObject</code> action by restricting the object tags allowed in the request. For examples of using these condition keys, see Object Tagging and Access Control Policies (p. 113).</p>
	<code>s3:object-lock-mode</code>	<p>When you upload an object, you can use the <code>s3:object-lock-mode</code> condition to restrict the user to only set a <code>COMPLIANCE</code> or <code>GOVERNANCE</code> mode on an object.</p>

Permission	Applicable Condition Keys (or keywords)	Description
	s3:object-lock-retain-until-date	When you upload an object, you can use the s3:object-lock-retain-until-date condition to limit the retention dates allowed on an object.
	s3:object-lock-legal-hold	When you upload an object, you can use the s3:object-lock-legal-hold condition to restrict the user from setting legal hold on an object.
s3:PutObjectAcl	<ul style="list-style-type: none"> s3:x-amz-acl (for canned ACL permissions) s3:x-amz-grant-permission (for explicit permissions), where <i>permission</i> can be: read, write, read-acp, write-acp, grant-full-control 	<p>The PUT Object ACL API sets the access control list (ACL) on the specified object. The operation supports ACL-related headers. When granting this permission, the bucket owner can add conditions using these keys to require certain permissions. For more information about ACLs, see Access Control List (ACL) Overview (p. 403).</p> <p>For example, the bucket owner may want to retain control of the object regardless of who owns the object. To accomplish this, the bucket owner can add a condition using one of these keys to require the user to include specific permissions to the bucket owner.</p>
	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission for the s3:PutObjectAcl action to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113) .
s3:PutObjectTagging	<ul style="list-style-type: none"> s3:RequestObjectTagKeys s3:RequestObjectTag/<tag-key> 	Using this condition key, you can limit permission for the s3:PutObjectTagging action by restricting the object tags allowed in the request. For examples of using these condition keys, see Object Tagging and Access Control Policies (p. 113) .
	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113) .

Permission	Applicable Condition Keys (or keywords)	Description
s3:PutObjectVersionTagging	<ul style="list-style-type: none"> s3:RequestObjectTagKeys s3:RequestObjectTag/<tag-key> 	Using this condition key, you can limit permission for the s3:PutObjectVersionTagging action by restricting the object tags allowed in the request. For examples of using these condition keys, see Object Tagging and Access Control Policies (p. 113).
	s3:VersionId	Using this condition key, you can limit the permission for the s3:PutObjectVersionTagging action to a specific object version. For an example policy, see Example 4: Granting Access to a Specific Version of an Object (p. 364).
	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113).
s3:GetObjectVersion	s3:VersionId	<p>This Amazon S3 permission enables the user to perform a set of Amazon S3 API operations (see Amazon S3 Permissions for Object Operations (p. 345)). For a version-enabled bucket, you can specify the object version to retrieve data for.</p> <p>By adding a condition using this key, the bucket owner can restrict the user to accessing data only for a specific version of the object. For an example policy, see Example 4: Granting Access to a Specific Version of an Object (p. 364).</p>
	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113).
s3:GetObject	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113).

Permission	Applicable Condition Keys (or keywords)	Description
s3:GetObjectAcl	s3:ExistingObjectTag/ <tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113).
s3:GetObjectVersionAcl	s3:VersionId	You can retrieve the access control list (ACL) of a specific object version using the GET Object acl API. The user must have permission for the s3:GetObjectVersionAcl action. For a version-enabled bucket, this Amazon S3 permission allows a user to get the ACL for a specific version of the object. The bucket owner can add a condition using the key to restrict the user to a specific version of the object.
	s3:ExistingObjectTag/ <tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113).
s3:PutObjectVersionAcl	s3:VersionId	For a version-enabled bucket, you can specify the object version in the PUT Object acl request to set ACL on a specific object version. Using this condition, the bucket owner can restrict the user to setting an ACL only on a specific version of an object.
	<ul style="list-style-type: none"> s3:x-amz-acl (for canned ACL permissions) s3:x-amz-grant-permission (for explicit permissions), where <i>permission</i> can be: read, write, read-acp, write-acp, grant-full-control 	<p>For a version-enabled bucket, this Amazon S3 permission allows you to set an ACL on a specific version of the object.</p> <p>For a description of these condition keys, see the s3:PutObjectACL permission in this table.</p>
	s3:ExistingObjectTag/ <tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113).

Permission	Applicable Condition Keys (or keywords)	Description
s3:DeleteObjectVersion	s3:VersionId	<p>For a version-enabled bucket, this Amazon S3 permission allows the user to delete a specific version of the object.</p> <p>The bucket owner can add a condition using this key to limit the user's ability to delete only a specific version of the object.</p> <p>For an example of using this condition key, see Example 4: Granting Access to a Specific Version of an Object (p. 364). The example is about granting the s3:GetObjectVersion action, but the policy shows the use of this condition key.</p>
s3:DeleteObjectTagging	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113) .
s3:DeleteObjectVersionTagging	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113) .
	s3:VersionId	Using this condition key, you can limit the permission for the s3:DeleteObjectVersionTagging action to a specific object version. For an example policy, see Example 4: Granting Access to a Specific Version of an Object (p. 364) .
s3:GetObjectTagging	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113) .
s3:GetObjectVersionTagging	s3:ExistingObjectTag/<tag-key>	Using this condition key, you can limit the permission to only on objects that have a specific tag key and value. For examples, see Object Tagging and Access Control Policies (p. 113) .

Permission	Applicable Condition Keys (or keywords)	Description
	s3:VersionId	Using this condition key, you can limit the permission for the s3:GetObjectVersionTagging action to a specific object version. For an example policy, see Example 4: Granting Access to a Specific Version of an Object (p. 364).
s3:PutObjectRetention	s3:object-lock-remaining-retention-days	When granting this permission, the bucket owner can set minimum and maximum allowable retention periods for objects within the bucket using this condition key.
	s3:object-lock-mode	You can use the s3:object-lock-mode condition to restrict the user to only set a COMPLIANCE or GOVERNANCE mode on an object.
	s3:object-lock-retain-until-date	You can use the s3:object-lock-retain-until-date condition to limit the retention dates allowed on an object.
	s3:object-lock-legal-hold	You can use the s3:object-lock-legal-hold condition to restrict the user from setting legal hold on an object.
s3:CreateMultipartUpload	s3:object-lock-mode	When you upload an object, you can use the s3:object-lock-mode condition to restrict the user to only set a COMPLIANCE or GOVERNANCE mode on an object.
	s3:object-lock-retain-until-date	When you upload an object, you can use the s3:object-lock-retain-until-date condition to limit the retention dates allowed on an object.
	s3:object-lock-legal-hold	When you upload an object, you can use the s3:object-lock-legal-hold condition to restrict the user from setting legal hold on an object.

Example 1: Granting s3:PutObject Permission with a Condition Requiring the Bucket Owner to Get Full Control

Suppose that Account A owns a bucket and the account administrator wants to grant Dave, a user in Account B, permissions to upload objects. By default, objects that Dave uploads are owned by Account B, and Account A has no permissions on these objects. Because the bucket owner is paying the bills, it wants full permissions on the objects that Dave uploads. The Account A administrator can do this by granting the s3:PutObject permission to Dave, with a condition that the request include ACL-specific headers, that either grants full permission explicitly or uses a canned ACL (see [PUT Object](#)).

- Require the `x-amz-full-control` header in the request with full control permission to the bucket owner.

The following bucket policy grants the `s3:PutObject` permission to user Dave with a condition using the `s3:x-amz-grant-full-control` condition key, which requires the request to include the `x-amz-full-control` header.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountB-ID:user/Dave"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
        }
      }
    }
  ]
}
```

Note

This example is about cross-account permission. However, if Dave (who is getting the permission) belongs to the AWS account that owns the bucket, this conditional permission is not necessary. This is because the parent account to which Dave belongs owns objects that the user uploads.

The preceding bucket policy grants conditional permission to user Dave in Account B. While this policy is in effect, it is possible for Dave to get the same permission without any condition via some other policy. For example, Dave can belong to a group, and you grant the group `s3:PutObject` permission without any condition. To avoid such permission loopholes, you can write a stricter access policy by adding explicit deny. In this example, you explicitly deny the user Dave upload permission if he does not include the necessary headers in the request granting full permissions to the bucket owner. Explicit deny always supersedes any other permission granted. The following is the revised access policy example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
        }
      }
    },
    {
      "Sid": "statement2",
```

```
    "Effect": "Deny",
    "Principal": {
      "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::examplebucket/*",
    "Condition": {
      "StringNotEquals": {
        "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
      }
    }
  }
]
```

If you have two AWS accounts, you can test the policy using the AWS Command Line Interface (AWS CLI). You attach the policy and, using Dave's credentials, test the permission using the following AWS CLI `put-object` command. You provide Dave's credentials by adding the `--profile` parameter. You grant full control permission to the bucket owner by adding the `--grant-full-control` parameter. For more information about setting up and using the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body c:\HappyFace.jpg
--grant-full-control id="AccountA-CanonicalUserID" --profile AccountBUserProfile
```

- Require the `x-amz-acl` header with a canned ACL granting full control permission to the bucket owner.

To require the `x-amz-acl` header in the request, you can replace the key-value pair in the `Condition` block and specify the `s3:x-amz-acl` condition key, as shown in the following example.

```
"Condition": {
  "StringNotEquals": {
    "s3:x-amz-acl": "bucket-owner-full-control"
  }
}
```

To test the permission using the AWS CLI, you specify the `--acl` parameter. The AWS CLI then adds the `x-amz-acl` header when it sends the request.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body c:\HappyFace.jpg
--acl "bucket-owner-full-control" --profile AccountBadmin
```

Example 2: Granting `s3:PutObject` Permission Requiring Objects Stored Using Server-Side Encryption

Suppose that Account A owns a bucket. The account administrator wants to grant Jane, a user in Account A, permission to upload objects with a condition that Jane always request server-side encryption so that Amazon S3 saves objects encrypted. The Account A administrator can accomplish using the `s3:x-amz-server-side-encryption` condition key as shown. The key-value pair in the `Condition` block specifies the `s3:x-amz-server-side-encryption` key.

```
"Condition": {
  "StringNotEquals": {
    "s3:x-amz-server-side-encryption": "AES256"
  }
}
```

When testing the permission using the AWS CLI, you must add the required parameter using the `--server-side-encryption` parameter.

```
aws s3api put-object --bucket example1bucket --key HappyFace.jpg --body c:\HappyFace.jpg --server-side-encryption "AES256" --profile AccountBadmin
```

Example 3: Granting s3:PutObject Permission to Copy Objects with a Restriction on the Copy Source

In the PUT Object request, when you specify a source object, it is a copy operation (see [PUT Object - Copy](#)). Accordingly, the bucket owner can grant a user permission to copy objects with restrictions on the source—for example:

- Allow copying objects only from the `sourcebucket` bucket.
- Allow copying objects from the `sourcebucket` bucket, and only the objects whose key name prefix starts with `public/`. For example, `sourcebucket/public/*`
- Allow copying only a specific object from the `sourcebucket`; for example, `sourcebucket/example.jpg`.

The following bucket policy grants user Dave `s3:PutObject` permission. It allows him to copy objects only with a condition that the request include the `s3:x-amz-copy-source` header and the header value specify the `/examplebucket/public/*` key name prefix.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "cross-account permission to user in your own account",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
      },
      "Action": ["s3:PutObject"],
      "Resource": "arn:aws:s3:::examplebucket/*"
    },
    {
      "Sid": "Deny your user permission to upload object if copy source is not / bucket/folder",
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "StringNotLike": {
          "s3:x-amz-copy-source": "examplebucket/public/*"
        }
      }
    }
  ]
}
```

You can test the permission using the AWS CLI `copy-object` command. You specify the source by adding the `--copy-source` parameter, and the key name prefix must match the prefix allowed in the policy. You need to provide the user Dave credentials using the `--profile` parameter. For more information about setting up the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs](#) (p. 316).

```
aws s3api copy-object --bucket examplebucket --key HappyFace.jpg  
--copy-source examplebucket/public/PublicHappyFace1.jpg --profile AccountADave
```

The preceding policy uses the `StringNotLike` condition. To grant permission to copy only a specific object, you must change the condition from `StringNotLike` to `StringNotEquals` and then specify the exact object key as shown.

```
"Condition": {  
  "StringNotEquals": {  
    "s3:x-amz-copy-source": "examplebucket/public/PublicHappyFace1.jpg"  
  }  
}
```

Example 4: Granting Access to a Specific Version of an Object

Suppose that Account A owns a version-enabled bucket. The bucket has several versions of the `HappyFace.jpg` object. The account administrator now wants to grant its user (Dave) permission to get only a specific version of the object. The account administrator can accomplish this by granting Dave `s3:GetObjectVersion` permission conditionally as shown. The key-value pair in the `Condition` block specifies the `s3:VersionId` condition key.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "statement1",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"  
      },  
      "Action": ["s3:GetObjectVersion"],  
      "Resource": "arn:aws:s3::examplebucketversionenabled/HappyFace.jpg"  
    },  
    {  
      "Sid": "statement2",  
      "Effect": "Deny",  
      "Principal": {  
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"  
      },  
      "Action": ["s3:GetObjectVersion"],  
      "Resource": "arn:aws:s3::examplebucketversionenabled/HappyFace.jpg",  
      "Condition": {  
        "StringNotEquals": {  
          "s3:VersionId": "AaaHbAQitwiL_h47_44lR02DDfLLB05e"  
        }  
      }  
    }  
  ]  
}
```

In this case, Dave needs to know the exact object version ID to retrieve the object.

You can test the permissions using the AWS CLI `get-object` command with the `--version-id` parameter identifying the specific object version. The command retrieves the object and saves it to the `OutputFile.jpg` file.

```
aws s3api get-object --bucket examplebucketversionenabled --key HappyFace.jpg  
OutputFile.jpg --version-id AaaHbAQitwiL_h47_44lR02DDfLLB05e --profile AccountADave
```

Example 5: Restricting Object Uploads to Objects with a Specific Storage Class

Suppose that Account A owns a bucket. The account administrator wants to restrict Dave, a user in Account A, to be able to only upload objects to the bucket that are stored with the `STANDARD_IA` storage class. The Account A administrator can do this by using the `s3:x-amz-storage-class` condition key as shown in the following example bucket policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
      },
      "Action": "s3:PutObject",
      "Resource": [
        "arn:aws:s3:::examplebucket/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-storage-class": [
            "STANDARD_IA"
          ]
        }
      }
    }
  ]
}
```

Amazon S3 Condition Keys for Bucket Operations

The following table shows list of bucket operation-specific permissions that you can grant in policies. For each permission, it shows the available keys that you can use in specifying a condition.

Permission	Applicable Condition Keys	Description
<code>s3:CreateBucket</code>	<ul style="list-style-type: none"> <code>s3:x-amz-acl</code> (for canned ACL permissions) <code>s3:x-amz-grant-permission</code> (for explicit permissions), where <i>permission</i> can be: read, write, read-acp, write-acp, full-control 	The Create Bucket API (see PUT Bucket) supports ACL-specific headers. Using these condition keys, you can require a user to set these headers in the request granting specific permissions.
	<code>s3:LocationConstraint</code>	Using this condition key, you can restrict a user to create a bucket in a specific AWS Region. For a policy example, see Example 1: Allow a User to Create a Bucket but Only in a Specific Region (p. 368).
<code>s3:ListBucket</code>	<code>s3:prefix</code>	Using this condition key, you can limit the response of the Get Bucket (List Objects) API (see GET Bucket (List Objects) API (see GET Bucket (List

Permission	Applicable Condition Keys	Description
		<p>Objects)) to key names with a specific prefix.</p> <p>The Get Bucket (List Objects) API returns a list of object keys in the specified bucket. This API supports the <code>prefix</code> header to retrieve only the object keys with a specific prefix. This condition key relates to the <code>prefix</code> header.</p> <p>For example, the Amazon S3 console supports the folder concept using key name prefixes. So if you have two objects with key names <code>public/object1.jpg</code> and <code>public/object2.jpg</code>, the console shows the objects under the <code>public</code> folder. If you organize your object keys using such prefixes, you can grant <code>s3:ListBucket</code> permission with the condition that will allow the user to get a list of key names with a specific prefix.</p> <p>For a policy example, see Example 2: Allow a User to Get a List of Objects in a Bucket According to a Specific Prefix (p. 369).</p>
	<code>s3:delimiter</code>	<p>If you organize your object key names using prefixes and delimiters, you can use this condition key to require the user to specify the <code>delimiter</code> parameter in the Get Bucket (List Objects) request. In this case, the response Amazon S3 returns is a list of object keys with common prefixes grouped together. For an example of using prefixes and delimiters, go to Get Bucket (List Objects).</p>
	<code>s3:max-keys</code>	<p>Using this condition, you can limit the number of keys Amazon S3 returns in response to the Get Bucket (List Objects) request by requiring the user to specify the <code>max-keys</code> parameter. By default the API returns up to 1,000 key names.</p> <p>For a list of numeric conditions you can use, see Numeric Condition Operators in the <i>IAM User Guide</i>.</p>

Permission	Applicable Condition Keys	Description
s3:ListBucketVersions	s3:prefix	<p>If your bucket is version-enabled, you can use the GET Bucket Object versions API (see GET Bucket Object versions) to retrieve metadata of all of the versions of objects. For this API, the bucket owner must grant the s3:ListBucketVersions permission in the policy.</p> <p>Using this condition key, you can limit the response of the API to key names with a specific prefix by requiring the user to specify the prefix parameter in the request with a specific value.</p> <p>For example, the Amazon S3 console supports the folder concept of using key name prefixes. If you have two objects with key names public/object1.jpg and public/object2.jpg, the console shows the objects under the public folder. If you organize your object keys using such prefixes, you can grant s3:ListBucket permission with the condition that will allow a use to get a list of key names with a specific prefix.</p> <p>For a policy example, see Example 2: Allow a User to Get a List of Objects in a Bucket According to a Specific Prefix (p. 369).</p>
	s3:delimiter	<p>If you organize your object key names using prefixes and delimiters, you can use this condition key to require the user to specify the delimiter parameter in the GET Bucket Object versions request. In this case, the response Amazon S3 returns is a list of object keys with common prefixes grouped together.</p>
	s3:max-keys	<p>Using this condition, you can limit the number of keys Amazon S3 returns in response to the GET Bucket Object versions request by requiring the user to specify the max-keys parameter. By default, the API returns up to 1,000 key names. For a list of numeric conditions you can use, see Numeric Condition Operators in the <i>IAM User Guide</i>.</p>

Permission	Applicable Condition Keys	Description
s3:PutBucketAcl	<ul style="list-style-type: none"> s3:x-amz-acl (for canned ACL permissions) s3:x-amz-grant-permission (for explicit permissions), where <i>permission</i> can be: read, write, read-acp, write-acp, full-control 	The PUT Bucket acl API (see PUT Bucket) supports ACL-specific headers. You can use these condition keys to require a user to set these headers in the request.

Example 1: Allow a User to Create a Bucket but Only in a Specific Region

Suppose that an AWS account administrator wants to grant its user (Dave) permission to create a bucket in the South America (São Paulo) Region only. The account administrator can attach the following user policy granting the s3:CreateBucket permission with a condition as shown. The key-value pair in the Condition block specifies the s3:LocationConstraint key and the sa-east-1 Region as its value.

Note

In this example, the bucket owner is granting permission to one of its users, so either a bucket policy or a user policy can be used. This example shows a user policy.

For a list of Amazon S3 Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket"
      ],
      "Resource": [
        "arn:aws:s3::*"
      ],
      "Condition": {
        "StringLike": {
          "s3:LocationConstraint": "sa-east-1"
        }
      }
    }
  ]
}
```

This policy restricts the user from creating a bucket in any other Region except sa-east-1. However, it is possible some other policy will grant this user permission to create buckets in another Region. For example, if the user belongs to a group, the group might have a policy attached to it allowing all users in the group permission to create buckets in another Region. To ensure that the user does not get permission to create buckets in any other Region, you can add an explicit deny statement in this policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
```

```
        "Action": [
            "s3:CreateBucket"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ],
        "Condition": {
            "StringLike": {
                "s3:LocationConstraint": "sa-east-1"
            }
        }
    },
    {
        "Sid": "statement2",
        "Effect": "Deny",
        "Action": [
            "s3:CreateBucket"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ],
        "Condition": {
            "StringNotLike": {
                "s3:LocationConstraint": "sa-east-1"
            }
        }
    }
  ]
}
```

The Deny statement uses the `StringNotLike` condition. That is, a create bucket request is denied if the location constraint is not "sa-east-1". The explicit deny does not allow the user to create a bucket in any other Region, no matter what other permission the user gets.

You can test the policy using the following `create-bucket` AWS CLI command. This example uses the `bucketconfig.txt` file to specify the location constraint. Note the Windows file path. You need to update the bucket name and path as appropriate. You must provide user credentials using the `--profile` parameter. For more information about setting up and using the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

```
aws s3api create-bucket --bucket examplebucket --profile AccountADave --create-bucket-configuration file:///c:/Users/someUser/bucketconfig.txt
```

The `bucketconfig.txt` file specifies the configuration as follows.

```
{"LocationConstraint": "sa-east-1"}
```

Example 2: Allow a User to Get a List of Objects in a Bucket According to a Specific Prefix

A bucket owner can restrict a user to list the contents of a specific folder in the bucket. This is useful if objects in the bucket are organized by key name prefixes. The Amazon S3 console then uses the prefixes to show a folder hierarchy (only the console supports the concept of folders; the Amazon S3 API supports only buckets and objects).

In this example, the bucket owner and the parent account to which the user belongs are the same. So the bucket owner can use either a bucket policy or a user policy. First, we show a user policy.

The following user policy grants the `s3:ListBucket` permission (see [GET Bucket \(List Objects\)](#)) with a condition that requires the user to specify the prefix in the request with the value `projects`.

```
{
```

```

"Version":"2012-10-17",
"Statement":[
  {
    "Sid":"statement1",
    "Effect":"Allow",
    "Action":[
      "s3:ListBucket"
    ],
    "Resource":[
      "arn:aws:s3:::examplebucket"
    ],
    "Condition" : {
      "StringEquals" : {
        "s3:prefix": "projects"
      }
    }
  },
  {
    "Sid":"statement2",
    "Effect":"Deny",
    "Action":[
      "s3:ListBucket"
    ],
    "Resource":[
      "arn:aws:s3:::examplebucket"
    ],
    "Condition" : {
      "StringNotEquals" : {
        "s3:prefix": "projects"
      }
    }
  }
]
}

```

The condition restricts the user to listing object keys with the `projects` prefix. The added explicit deny denies the user request for listing keys with any other prefix no matter what other permissions the user might have. For example, it is possible that the user gets permission to list object keys without any restriction; for example, either by updates to the preceding user policy or via a bucket policy. But because explicit deny always supersedes, the user request to list keys other than the `project` prefix is denied.

The preceding policy is a user policy. If you add the `Principal` element to the policy, identifying the user, you now have a bucket policy as shown.

```

{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"statement1",
      "Effect":"Allow",
      "Principal": {
        "AWS": "arn:aws:iam::BucketOwner-accountID:user/user-name"
      },
      "Action":[
        "s3:ListBucket"
      ],
      "Resource":[
        "arn:aws:s3:::examplebucket"
      ],
      "Condition" : {
        "StringEquals" : {
          "s3:prefix": "examplefolder"
        }
      }
    }
  ]
}

```

```
    },
    {
      "Sid": "statement2",
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::BucketOwner-AccountID:user/user-name"
      },
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket"
      ],
      "Condition": {
        "StringNotEquals": {
          "s3:prefix": "examplefolder"
        }
      }
    }
  ]
}
```

You can test the policy using the following `list-object` AWS CLI command. In the command, you provide user credentials using the `--profile` parameter. For more information about setting up and using the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 316\)](#).

```
aws s3api list-objects --bucket examplebucket --prefix examplefolder --profile AccountADave
```

Now if the bucket is version-enabled, to list the objects in the bucket, instead of `s3:ListBucket` permission, you must grant the `s3:ListBucketVersions` permission in the preceding policy. This permission also supports the `s3:prefix` condition key.

Bucket Policy Examples

This section presents a few examples of typical use cases for bucket policies. The policies use *bucket* and *examplebucket* strings in the resource value. To test these policies, you need to replace these strings with your bucket name. For information about access policy language, see [Access Policy Language Overview \(p. 341\)](#).

Note

Bucket policies are limited to 20 KB in size.

You can use the [AWS Policy Generator](#) to create a bucket policy for your Amazon S3 bucket. You can then use the generated document to set your bucket policy by using the [Amazon S3 console](#), by a number of third-party tools, or via your application.

Important

When testing permissions using the Amazon S3 console, you will need to grant additional permissions that the console requires—`s3:ListAllMyBuckets`, `s3:GetBucketLocation`, and `s3:ListBucket` permissions. For an example walkthrough that grants permissions to users and tests them using the console, see [Walkthrough: Controlling Access to a Bucket with User Policies \(p. 385\)](#).

Topics

- [Granting Permissions to Multiple Accounts with Added Conditions \(p. 372\)](#)
- [Granting Read-Only Permission to an Anonymous User \(p. 372\)](#)
- [Restricting Access to Specific IP Addresses \(p. 372\)](#)
- [Restricting Access to a Specific HTTP Referrer \(p. 374\)](#)
- [Granting Permission to an Amazon CloudFront Origin Identity \(p. 375\)](#)

- [Adding a Bucket Policy to Require MFA \(p. 375\)](#)
- [Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control \(p. 377\)](#)
- [Granting Permissions for Amazon S3 Inventory and Amazon S3 Analytics \(p. 377\)](#)
- [Example Bucket Policies for VPC Endpoints for Amazon S3 \(p. 378\)](#)

Granting Permissions to Multiple Accounts with Added Conditions

The following example policy grants the `s3:PutObject` and `s3:PutObjectAcl` permissions to multiple AWS accounts and requires that any request for these operations include the `public-read` canned ACL. For more information, see [Specifying Permissions in a Policy \(p. 345\)](#) and [Specifying Conditions in a Policy \(p. 350\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AddCannedAcl",
      "Effect": "Allow",
      "Principal": { "AWS":
        [ "arn:aws:iam::11112223333:root", "arn:aws:iam::444455556666:root" ] },
      "Action": [ "s3:PutObject", "s3:PutObjectAcl" ],
      "Resource": [ "arn:aws:s3:::examplebucket/*" ],
      "Condition": { "StringEquals": { "s3:x-amz-acl": [ "public-read" ] } }
    }
  ]
}
```

Granting Read-Only Permission to an Anonymous User

The following example policy grants the `s3:GetObject` permission to any public anonymous users. (For a list of permissions and the operations that they allow, see [Specifying Permissions in a Policy \(p. 345\)](#).) This permission allows anyone to read the object data, which is useful for when you configure your bucket as a website and want everyone to be able to read objects in the bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AddPerm",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [ "s3:GetObject" ],
      "Resource": [ "arn:aws:s3:::examplebucket/*" ]
    }
  ]
}
```

Warning

Use caution when granting anonymous access to your S3 bucket. When you grant anonymous access, anyone in the world can access your bucket. We highly recommend that you never grant any kind of anonymous write access to your S3 bucket.

Restricting Access to Specific IP Addresses

The following example grants permissions to any user to perform any Amazon S3 operations on objects in the specified bucket. However, the request must originate from the range of IP addresses specified in the condition.

The condition in this statement identifies the 54.240.143.* range of allowed Internet Protocol version 4 (IPv4) IP addresses, with one exception: 54.240.143.188.

The Condition block uses the `IpAddress` and `NotIpAddress` conditions and the `aws:SourceIp` condition key, which is an AWS-wide condition key. For more information about these condition keys, see [Specifying Conditions in a Policy \(p. 350\)](#). The `aws:SourceIp` IPv4 values use the standard CIDR notation. For more information, see [IP Address Condition Operators](#) in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Id": "S3PolicyId1",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "IpAddress": {"aws:SourceIp": "54.240.143.0/24"},
        "NotIpAddress": {"aws:SourceIp": "54.240.143.188/32"}
      }
    }
  ]
}
```

Allowing IPv4 and IPv6 Addresses

When you start using IPv6 addresses, we recommend that you update all of your organization's policies with your IPv6 address ranges in addition to your existing IPv4 ranges to ensure that the policies continue to work as you make the transition to IPv6.

The following example bucket policy shows how to mix IPv4 and IPv6 address ranges to cover all of your organization's valid IP addresses. The example policy would allow access to the example IP addresses 54.240.143.1 and 2001:DB8:1234:5678::1 and would deny access to the addresses 54.240.143.129 and 2001:DB8:1234:5678:ABCD::1.

The IPv6 values for `aws:SourceIp` must be in standard CIDR format. For IPv6 we support using `::` to represent a range of 0s, for example, 2032001:DB8:1234:5678::IP Address Condition Operators in the *IAM User Guide*.

```
{
  "Id": "PolicyId2",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowIPmix",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::
```

```
    ]
  }
}
]
```

Restricting Access to a Specific HTTP Referrer

Suppose you have a website with domain name (`www.example.com` or `example.com`) with links to photos and videos stored in your S3 bucket, `examplebucket`. By default, all the S3 resources are private, so only the AWS account that created the resources can access them. To allow read access to these objects from your website, you can add a bucket policy that allows `s3:GetObject` permission with a condition, using the `aws:Referer` key, that the get request must originate from specific webpages. The following policy specifies the `StringLike` condition with the `aws:Referer` condition key.

```
{
  "Version": "2012-10-17",
  "Id": "http referer policy example",
  "Statement": [
    {
      "Sid": "Allow get requests originating from www.example.com and example.com.",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "StringLike": {"aws:Referer": ["http://www.example.com/*", "http://example.com/*"]}
      }
    }
  ]
}
```

Make sure the browsers you use include the `http referer` header in the request.

You can further secure access to objects in the `examplebucket` bucket by adding explicit deny to the bucket policy as shown in the following example. Explicit deny supersedes any permission you might grant to objects in the `examplebucket` bucket using other means such as ACLs or user policies.

Important

Be aware that this example will prevent all users (including the root user) from performing all Amazon S3 actions, including managing bucket policies. Consider adding a third `Sid` that grants the root user `s3:*` actions.

```
{
  "Version": "2012-10-17",
  "Id": "http referer policy example",
  "Statement": [
    {
      "Sid": "Allow get requests referred by www.example.com and example.com.",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "StringLike": {"aws:Referer": ["http://www.example.com/*", "http://example.com/*"]}
      }
    },
    {
      "Sid": "Explicit deny to ensure requests are allowed only from specific referer.",
```



```
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::examplebucket/*",
    "Condition": {
      "StringNotLike": {"aws:Referer": ["http://www.example.com/*", "http://example.com/*"]}
    }
  }
]
```

Granting Permission to an Amazon CloudFront Origin Identity

The following example bucket policy grants a CloudFront Origin Identity permission to get (list) all objects in your Amazon S3 bucket. The CloudFront Origin Identity is used to enable the CloudFront private content feature. The policy uses the CanonicalUser prefix, instead of AWS, to specify a Canonical User ID. To learn more about CloudFront support for serving private content, go to the [Serving Private Content](#) topic in the *Amazon CloudFront Developer Guide*. You must specify the canonical user ID for your CloudFront distribution's origin access identity. For instructions about finding the canonical user ID, see [Specifying a Principal in a Policy](#) (p. 343).

```
{
  "Version": "2012-10-17",
  "Id": "PolicyForCloudFrontPrivateContent",
  "Statement": [
    {
      "Sid": "Grant a CloudFront Origin Identity access to support private content",
      "Effect": "Allow",
      "Principal": {"CanonicalUser": "CloudFront Origin Identity Canonical User ID"},
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::examplebucket/*"
    }
  ]
}
```

Adding a Bucket Policy to Require MFA

Amazon S3 supports MFA-protected API access, a feature that can enforce multi-factor authentication (MFA) for access to your Amazon S3 resources. Multi-factor authentication provides an extra level of security you can apply to your AWS environment. It is a security feature that requires users to prove physical possession of an MFA device by providing a valid MFA code. For more information, go to [AWS Multi-Factor Authentication](#). You can require MFA authentication for any requests to access your Amazon S3 resources.

You can enforce the MFA authentication requirement using the `aws:MultiFactorAuthAge` key in a bucket policy. IAM users can access Amazon S3 resources by using temporary credentials issued by the AWS Security Token Service (STS). You provide the MFA code at the time of the STS request.

When Amazon S3 receives a request with MFA authentication, the `aws:MultiFactorAuthAge` key provides a numeric value indicating how long ago (in seconds) the temporary credential was created. If the temporary credential provided in the request was not created using an MFA device, this key value is null (absent). In a bucket policy, you can add a condition to check this value, as shown in the following example bucket policy. The policy denies any Amazon S3 operation on the `/taxdocuments` folder in the `examplebucket` bucket if the request is not MFA authenticated. To learn more about MFA authentication, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Id": "123",
```

```
"Statement": [
  {
    "Sid": "",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
    "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
  }
]
```

The `Null` condition in the `Condition` block evaluates to `true` if the `aws:MultiFactorAuthAge` key value is null, indicating that the temporary security credentials in the request were created without the MFA key.

The following bucket policy is an extension of the preceding bucket policy. It includes two policy statements. One statement allows the `s3:GetObject` permission on a bucket (`examplebucket`) to everyone and another statement further restricts access to the `examplebucket/taxdocuments` folder in the bucket by requiring MFA authentication.

```
{
  "Version": "2012-10-17",
  "Id": "123",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
      "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
    },
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::examplebucket/*"
    }
  ]
}
```

You can optionally use a numeric condition to limit the duration for which the `aws:MultiFactorAuthAge` key is valid, independent of the lifetime of the temporary security credential used in authenticating the request. For example, the following bucket policy, in addition to requiring MFA authentication, also checks how long ago the temporary session was created. The policy denies any operation if the `aws:MultiFactorAuthAge` key value indicates that the temporary session was created more than an hour ago (3,600 seconds).

```
{
  "Version": "2012-10-17",
  "Id": "123",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
      "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
    },
  ],
}
```

```
{
  "Sid": "",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:*",
  "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
  "Condition": { "NumericGreaterThan": { "aws:MultiFactorAuthAge": 3600 } }
},
{
  "Sid": "",
  "Effect": "Allow",
  "Principal": "*",
  "Action": ["s3:GetObject"],
  "Resource": "arn:aws:s3:::examplebucket/*"
}
]
```

Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control

You can allow another AWS account to upload objects to your bucket. However, you may decide that as a bucket owner you must have full control of the objects uploaded to your bucket. The following policy enforces that a specific AWS account (111111111111) be denied the ability to upload objects unless that account grants full-control access to the bucket owner identified by the email address (xyz@amazon.com). The `StringNotEquals` condition in the policy specifies the `s3:x-amz-grant-full-control` condition key to express the requirement (see [Specifying Conditions in a Policy](#) (p. 350)).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "111",
      "Effect": "Allow",
      "Principal": { "AWS": "111111111111" },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::examplebucket/*"
    },
    {
      "Sid": "112",
      "Effect": "Deny",
      "Principal": { "AWS": "111111111111" },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::examplebucket/*",
      "Condition": {
        "StringNotEquals": { "s3:x-amz-grant-full-control": [ "emailAddress=xyz@amazon.com" ] }
      }
    }
  ]
}
```

Granting Permissions for Amazon S3 Inventory and Amazon S3 Analytics

Amazon S3 inventory creates lists of the objects in an S3 bucket and Amazon S3 analytics export creates output files of the data used in the analysis. The bucket that the inventory lists the objects for is called the *source bucket*. The bucket where the inventory file is written and the bucket where the analytics export file is written is called a *destination bucket*. You must create a bucket policy for the destination bucket when setting up inventory for an S3 bucket and when setting up the analytics export. For more information, see [Amazon S3 Inventory](#) (p. 422) and [Amazon S3 Analytics – Storage Class Analysis](#) (p. 257).

The following example bucket policy grants Amazon S3 permission to write objects (PUTs) from the account for the source bucket to the destination bucket. You use a bucket policy like this on the destination bucket when setting up Amazon S3 inventory and Amazon S3 analytics export.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "InventoryAndAnalyticsExamplePolicy",
      "Effect": "Allow",
      "Principal": { "Service": "s3.amazonaws.com" },
      "Action": [ "s3:PutObject" ],
      "Resource": [ "arn:aws:s3:::destination-bucket/*" ],
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:s3:::source-bucket"
        },
        "StringEquals": {
          "aws:SourceAccount": "1234567890",
          "s3:x-amz-acl": "bucket-owner-full-control"
        }
      }
    }
  ]
}
```

Example Bucket Policies for VPC Endpoints for Amazon S3

You can use Amazon S3 bucket policies to control access to buckets from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints, or specific VPCs. This section contains example bucket policies that can be used to control S3 bucket access from VPC endpoints. To learn how to set up VPC endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*.

Amazon VPC enables you to launch Amazon Web Services (AWS) resources into a virtual network that you define. A VPC endpoint enables you to create a private connection between your VPC and another AWS service without requiring access over the Internet, through a VPN connection, through a NAT instance, or through AWS Direct Connect.

A VPC endpoint for Amazon S3 is a logical entity within a VPC that allows connectivity only to Amazon S3. The VPC endpoint routes requests to Amazon S3 and routes responses back to the VPC. VPC endpoints change only how requests are routed. Amazon S3 public endpoints and DNS names will continue to work with VPC endpoints. For important information about using Amazon VPC endpoints with Amazon S3, see [Gateway VPC Endpoints](#) and [Endpoints for Amazon S3](#) in the *Amazon VPC User Guide*.

VPC endpoints for Amazon S3 provides two ways to control access to your Amazon S3 data:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint. For information on this type of access control, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.
- You can control which VPCs or VPC endpoints have access to your S3 buckets by using S3 bucket policies. For examples of this type of bucket policy access control, see the following topics on restricting access.

Topics

- [Restricting Access to a Specific VPC Endpoint \(p. 379\)](#)
- [Restricting Access to a Specific VPC \(p. 379\)](#)
- [Related Resources \(p. 380\)](#)

Important

When applying the S3 bucket policies for VPC endpoints described in this section, you might block your access to the bucket without intending to do so. Bucket permissions intended to specifically limit bucket access to connections originating from your VPC endpoint can block all connections to the bucket. For information about how to fix this issue, see [How do I regain access to an Amazon S3 bucket after applying a policy to the bucket that restricts access to my VPC endpoint?](#) in the *AWS Support Knowledge Center*.

Restricting Access to a Specific VPC Endpoint

The following is an example of an S3 bucket policy that restricts access to a specific bucket, `examplebucket`, only from the VPC endpoint with the ID `vpce-1a2b3c4d`. The policy denies all access to the bucket if the specified endpoint is not being used. The `aws:sourceVpce` condition is used to specify the endpoint. The `aws:sourceVpce` condition does not require an ARN for the VPC endpoint resource, only the VPC endpoint ID. For more information about using conditions in a policy, see [Specifying Conditions in a Policy](#) (p. 350).

```
{
  "Version": "2012-10-17",
  "Id": "Policy1415115909152",
  "Statement": [
    {
      "Sid": "Access-to-specific-VPCE-only",
      "Principal": "*",
      "Action": "s3:*",
      "Effect": "Deny",
      "Resource": [ "arn:aws:s3:::examplebucket",
                    "arn:aws:s3:::examplebucket/*" ],
      "Condition": {
        "StringNotEquals": {
          "aws:sourceVpce": "vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

Restricting Access to a Specific VPC

You can create a bucket policy that restricts access to a specific VPC by using the `aws:sourceVpc` condition. This is useful if you have multiple VPC endpoints configured in the same VPC, and you want to manage access to your S3 buckets for all of your endpoints. The following is an example of a policy that allows VPC `vpc-111bbb22` to access `examplebucket` and its objects. The policy denies all access to the bucket if the specified VPC is not being used. The `vpc-111bbb22` condition key does not require an ARN for the VPC resource, only the VPC ID.

```
{
  "Version": "2012-10-17",
  "Id": "Policy1415115909153",
  "Statement": [
    {
      "Sid": "Access-to-specific-VPC-only",
      "Principal": "*",
      "Action": "s3:*",
      "Effect": "Deny",
      "Resource": [ "arn:aws:s3:::examplebucket",
                    "arn:aws:s3:::examplebucket/*" ],
      "Condition": {
        "StringNotEquals": {
          "aws:sourceVpc": "vpc-111bbb22"
        }
      }
    }
  ]
}
```

```
}  
  }  
]  
}
```

Related Resources

- [Bucket Policy Examples \(p. 371\)](#)
- [VPC Endpoints](#) in the *Amazon VPC User Guide*

User Policy Examples

This section shows several IAM user policies for controlling user access to Amazon S3. For information about access policy language, see [Access Policy Language Overview \(p. 341\)](#).

The following example policies will work if you test them programmatically. However, to use them with the Amazon S3 console, you must grant additional permissions that are required by the console. For information about using policies such as these with the Amazon S3 console, see [Walkthrough: Controlling Access to a Bucket with User Policies \(p. 385\)](#).

Topics

- [Allowing an IAM User Access to One of Your Buckets \(p. 380\)](#)
- [Allowing Each IAM User Access to a Folder in a Bucket \(p. 381\)](#)
- [Allowing a Group to Have a Shared Folder in Amazon S3 \(p. 383\)](#)
- [Allowing All Your Users to Read Objects in a Portion of the Corporate Bucket \(p. 384\)](#)
- [Allowing a Partner to Drop Files into a Specific Portion of the Corporate Bucket \(p. 384\)](#)
- [Walkthrough: Controlling Access to a Bucket with User Policies \(p. 385\)](#)

Allowing an IAM User Access to One of Your Buckets

In this example, you want to grant an IAM user in your AWS account access to one of your buckets, `examplebucket`, and allow the user to add, update, and delete objects.

In addition to granting the `s3:PutObject`, `s3:GetObject`, and `s3:DeleteObject` permissions to the user, the policy also grants the `s3:ListAllMyBuckets`, `s3:GetBucketLocation`, and `s3:ListBucket` permissions. These are the additional permissions required by the console. Also, the `s3:PutObjectAcl` and the `s3:GetObjectAcl` actions are required to be able to copy, cut, and paste objects in the console. For an example walkthrough that grants permissions to users and tests them using the console, see [Walkthrough: Controlling Access to a Bucket with User Policies \(p. 385\)](#).

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:ListAllMyBuckets"  
      ],  
      "Resource": "arn:aws:s3:::*"  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:ListBucket",  
        "s3:GetBucketLocation"  
      ],  
      "Resource": "arn:aws:s3:::*"  
    }  
  ]  
}
```

```
        "Resource": "arn:aws:s3:::examplebucket"
      },
      {
        "Effect": "Allow",
        "Action": [
          "s3:PutObject",
          "s3:PutObjectAcl",
          "s3:GetObject",
          "s3:GetObjectAcl",
          "s3:DeleteObject"
        ],
        "Resource": "arn:aws:s3:::examplebucket/*"
      }
    ]
  }
}
```

Allowing Each IAM User Access to a Folder in a Bucket

In this example, you want two IAM users, Alice and Bob, to have access to your bucket, `examplebucket`, so that they can add, update, and delete objects. However, you want to restrict each user's access to a single folder in the bucket. You might create folders with names that match the user names.

```
examplebucket
  Alice/
  Bob/
```

To grant each user access only to his or her folder, you can write a policy for each user and attach it individually. For example, you can attach the following policy to user Alice to allow her specific Amazon S3 permissions on the `examplebucket/Alice` folder.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
      ],
      "Resource": "arn:aws:s3:::examplebucket/Alice/*"
    }
  ]
}
```

You then attach a similar policy to user Bob, identifying folder Bob in the Resource value.

Instead of attaching policies to individual users, you can write a single policy that uses a policy variable and attach the policy to a group. First you must create a group and add both Alice and Bob to the group. The following example policy allows a set of Amazon S3 permissions in the `examplebucket/${aws:username}` folder. When the policy is evaluated, the policy variable `${aws:username}` is replaced by the requester's user name. For example, if Alice sends a request to put an object, the operation is allowed only if Alice is uploading the object to the `examplebucket/Alice` folder.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```

        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:DeleteObject",
            "s3:DeleteObjectVersion"
        ],
        "Resource": "arn:aws:s3:::examplebucket/${aws:username}/*"
    }
}

```

Note

When using policy variables, you must explicitly specify version 2012-10-17 in the policy. The default version of the access policy language, 2008-10-17, does not support policy variables.

If you want to test the preceding policy on the Amazon S3 console, the console requires permission for additional Amazon S3 permissions, as shown in the following policy. For information about how the console uses these permissions, see [Walkthrough: Controlling Access to a Bucket with User Policies \(p. 385\)](#).

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGroupToSeeBucketListInTheConsole",
      "Action": [ "s3:ListAllMyBuckets", "s3:GetBucketLocation" ],
      "Effect": "Allow",
      "Resource": [ "arn:aws:s3:::*" ]
    },
    {
      "Sid": "AllowRootLevelListingOfTheBucket",
      "Action": [ "s3:ListBucket" ],
      "Effect": "Allow",
      "Resource": [ "arn:aws:s3:::examplebucket" ],
      "Condition": {
        "StringEquals": {
          "s3:prefix": [ "" ],
          "s3:delimiter": [ "/" ]
        }
      }
    },
    {
      "Sid": "AllowListBucketOfASpecificUserPrefix",
      "Action": [ "s3:ListBucket" ],
      "Effect": "Allow",
      "Resource": [ "arn:aws:s3:::examplebucket" ],
      "Condition": {
        "StringLike": { "s3:prefix": [ "${aws:username}/*" ] }
      }
    },
    {
      "Sid": "AllowUserSpecificActionsOnlyInTheSpecificUserPrefix",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
      ],
      "Resource": "arn:aws:s3:::examplebucket/${aws:username}/*"
    }
  ]
}

```


Note

In the 2012-10-17 version of the policy, policy variables start with \$. This change in syntax can potentially create a conflict if your object key includes a \$. For example, to include an object key `my$file` in a policy, you specify the \$ character with `${$}`, `my${$}file`.

Although IAM user names are friendly, human-readable identifiers, they are not required to be globally unique. For example, if user Bob leaves the organization and another Bob joins, then new Bob could access old Bob's information. Instead of using user names, you could create folders based on user IDs. Each user ID is unique. In this case, you must modify the preceding policy to use the `${aws:user-id}` policy variable. For more information about user identifiers, see [IAM Identifiers](#) in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
      ],
      "Resource": "arn:aws:s3:::my_corporate_bucket/home/${aws:user-id}/*"
    }
  ]
}
```

Allowing Non-IAM Users (Mobile App Users) Access to Folders in a Bucket

Suppose that you want to develop a mobile app, a game that stores users' data in an S3 bucket. For each app user, you want to create a folder in your bucket. You also want to limit each user's access to his or her own folder. But you cannot create folders before someone downloads your app and starts playing the game, because you don't have a user ID.

In this case, you can require users to sign in to your app by using public identity providers such as Login with Amazon, Facebook, or Google. After users have signed in to your app through one of these providers, they have a user ID that you can use to create user-specific folders at runtime.

You can then use web identity federation in AWS Security Token Service to integrate information from the identity provider with your app and to get temporary security credentials for each user. You can then create IAM policies that allow the app to access your bucket and perform such operations as creating user-specific folders and uploading data. For more information about web identity federation, see [About Web Identity Federation](#) in the *IAM User Guide*.

Allowing a Group to Have a Shared Folder in Amazon S3

Attaching the following policy to the group grants everybody in the group access to the following folder in Amazon S3: `my_corporate_bucket/share/marketing`. Group members are allowed to access only the specific Amazon S3 permissions shown in the policy and only for objects in the specified folder.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetObjectVersion",

```

```
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
    ],
    "Resource": "arn:aws:s3:::my_corporate_bucket/share/marketing/*"
}
]
```

Allowing All Your Users to Read Objects in a Portion of the Corporate Bucket

In this example, you create a group named `AllUsers`, which contains all the IAM users that are owned by the AWS account. You then attach a policy that gives the group access to `GetObject` and `GetObjectVersion`, but only for objects in the `my_corporate_bucket/readonly` folder.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::my_corporate_bucket/readonly/*"
    }
  ]
}
```

Allowing a Partner to Drop Files into a Specific Portion of the Corporate Bucket

In this example, you create a group called `WidgetCo` that represents a partner company. You create an IAM user for the specific person or application at the partner company that needs access, and then you put the user in the group.

You then attach a policy that gives the group `PutObject` access to the following folder in the corporate bucket: `my_corporate_bucket/uploads/widgetco`.

You want to prevent the `WidgetCo` group from doing anything else with the bucket, so you add a statement that explicitly denies permission to any Amazon S3 permissions except `PutObject` on any Amazon S3 resource in the AWS account. This step is necessary only if there's a broad policy in use elsewhere in your AWS account that gives users wide access to Amazon S3 resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"
    },
    {
      "Effect": "Deny",
      "NotAction": "s3:PutObject",
      "Resource": "arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"
    },
    {
      "Effect": "Deny",
      "Action": "s3:*",
      "NotResource": "arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"
    }
  ]
}
```

```
}
```

Walkthrough: Controlling Access to a Bucket with User Policies

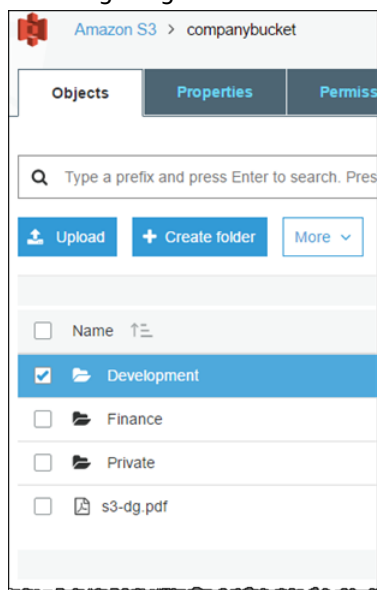
This walkthrough explains how user permissions work with Amazon S3. In this example, you create a bucket with folders. You then create AWS Identity and Access Management (IAM) users in your AWS account and grant those users incremental permissions on your Amazon S3 bucket and the folders in it.

Topics

- [The Basics of Buckets and Folders \(p. 385\)](#)
- [Walkthrough Summary \(p. 387\)](#)
- [Preparing for the Walkthrough \(p. 387\)](#)
- [Step 1: Create a Bucket \(p. 388\)](#)
- [Step 2: Create IAM Users and a Group \(p. 388\)](#)
- [Step 3: Verify That IAM Users Have No Permissions \(p. 389\)](#)
- [Step 4: Grant Group-Level Permissions \(p. 389\)](#)
- [Step 5: Grant IAM User Alice Specific Permissions \(p. 396\)](#)
- [Step 6: Grant IAM User Bob Specific Permissions \(p. 400\)](#)
- [Step 7: Secure the Private Folder \(p. 400\)](#)
- [Step 8: Clean Up \(p. 402\)](#)
- [Related Resources \(p. 402\)](#)

The Basics of Buckets and Folders

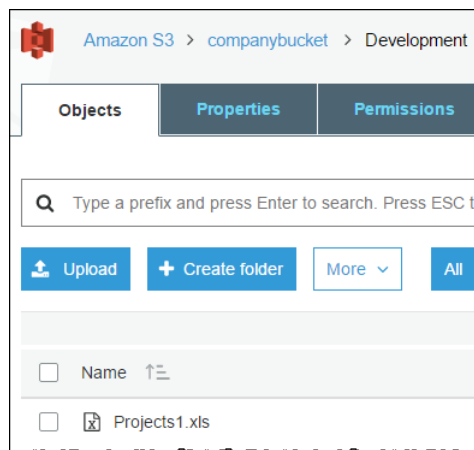
The Amazon S3 data model is a flat structure: You create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders, but you can emulate a folder hierarchy. Tools like the Amazon S3 console can present a view of these logical folders and subfolders in your bucket, as shown in the following image.



The console shows that a bucket named `companybucket` has three folders, `Private`, `Development`, and `Finance`, and an object, `s3-dg.pdf`. The console uses the object names (keys) to create a logical hierarchy with folders and subfolders. Consider the following examples:

- When you create the `Development` folder, the console creates an object with the key `Development/`. Note the trailing slash (/) delimiter.
- When you upload an object named `Projects1.xls` in the `Development` folder, the console uploads the object and gives it the key `Development/Projects1.xls`.

In the key, `Development` is the prefix and `/` is the delimiter. The Amazon S3 API supports prefixes and delimiters in its operations. For example, you can get a list of all objects from a bucket with a specific prefix and delimiter. On the console, when you open the `Development` folder, the console lists the objects in that folder. In the following example, the `Development` folder contains one object.



When the console lists the `Development` folder in the `companybucket` bucket, it sends a request to Amazon S3 in which it specifies a prefix of `Development` and a delimiter of `/` in the request. The console's response looks just like a folder list in your computer's file system. The preceding example shows that the bucket `companybucket` has an object with the key `Development/Projects1.xls`.

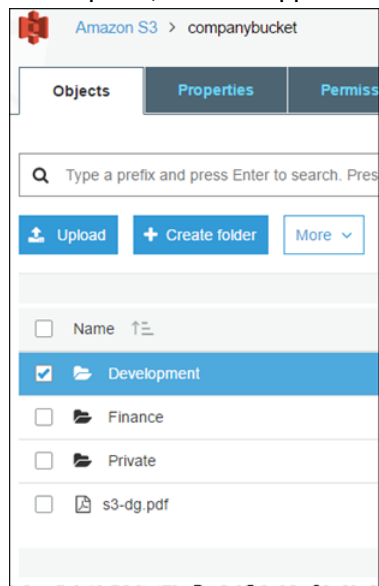
The console is using object keys to infer a logical hierarchy. Amazon S3 has no physical hierarchy; it only has buckets that contain objects in a flat file structure. When you create objects using the Amazon S3 API, you can use object keys that imply a logical hierarchy. When you create a logical hierarchy of objects, you can manage access to individual folders, as this walkthrough demonstrates.

Before you start, be sure that you are familiar with the concept of the *root-level* bucket content. Suppose that your `companybucket` bucket has the following objects:

- `Private/privDoc1.txt`
- `Private/privDoc2.zip`
- `Development/project1.xls`
- `Development/project2.xls`
- `Finance/Tax2011/document1.pdf`
- `Finance/Tax2011/document2.pdf`
- `s3-dg.pdf`

These object keys create a logical hierarchy with `Private`, `Development`, and the `Finance` as root-level folders and `s3-dg.pdf` as a root-level object. When you choose the bucket name on the Amazon S3 console, the root-level items appear as shown in the following image. The console shows the top-level

prefixes (`Private/`, `Development/`, and `Finance/`) as root-level folders. The object key `s3-dg.pdf` has no prefix, and so it appears as a root-level item.



Walkthrough Summary

In this walkthrough, you create a bucket with three folders (`Private`, `Development`, and `Finance`) in it.

You have two users, Alice and Bob. You want Alice to access only the `Development` folder, and you want Bob to access only the `Finance` folder. You want to keep the `Private` folder content private. In the walkthrough, you manage access by creating IAM users (the example uses the user names Alice and Bob) and granting them the necessary permissions.

IAM also supports creating user groups and granting group-level permissions that apply to all users in the group. This helps you better manage permissions. For this exercise, both Alice and Bob need some common permissions. So you also create a group named `Consultants` and then add both Alice and Bob to the group. You first grant permissions by attaching a group policy to the group. Then you add user-specific permissions by attaching policies to specific users.

Note

The walkthrough uses `companybucket` as the bucket name, Alice and Bob as the IAM users, and `Consultants` as the group name. Because Amazon S3 requires that bucket names be globally unique, you must replace the bucket name with a name that you create.

Preparing for the Walkthrough

In this example, you use your AWS account credentials to create IAM users. Initially, these users have no permissions. You incrementally grant these users permissions to perform specific Amazon S3 actions. To test these permissions, you sign in to the console with each user's credentials. As you incrementally grant permissions as an AWS account owner and test permissions as an IAM user, you need to sign in and out, each time using different credentials. You can do this testing with one browser, but the process will go faster if you can use two different browsers. Use one browser to connect to the AWS Management Console with your AWS account credentials and another to connect with the IAM user credentials.

To sign in to the AWS Management Console with your AWS account credentials, go to <https://console.aws.amazon.com/>. An IAM user cannot sign in using the same link. An IAM user must use an IAM-enabled sign-in page. As the account owner, you can provide this link to your users.

For more information about IAM, see [The AWS Management Console Sign-in Page](#) in the *IAM User Guide*.

To Provide a Sign-In Link for IAM Users

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the **Navigation** pane, choose **IAM Dashboard**.
3. Note the URL under **IAM users sign in link:**. You will give this link to IAM users to sign in to the console with their IAM user name and password.

Step 1: Create a Bucket

In this step, you sign in to the Amazon S3 console with your AWS account credentials, create a bucket, add folders (Development, Finance, and Private) to the bucket, and upload one or two sample documents in each folder.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create a bucket.

For step-by-step instructions, see [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

3. Upload one document to the bucket.

This exercise assumes that you have the `s3-dg.pdf` document at the root level of this bucket. If you upload a different document, substitute its file name for `s3-dg.pdf`.

4. Add three folders named `Private`, `Finance`, and `Development` to the bucket.

For step-by-step instructions to create a folder, see [Creating a Folder](#) in the *Amazon Simple Storage Service Console User Guide*.

5. Upload one or two documents to each folder.

For this exercise, assume that you have uploaded a couple of documents in each folder, resulting in the bucket having objects with the following keys:

- `Private/privDoc1.txt`
- `Private/privDoc2.zip`
- `Development/project1.xls`
- `Development/project2.xls`
- `Finance/Tax2011/document1.pdf`
- `Finance/Tax2011/document2.pdf`
- `s3-dg.pdf`

For step-by-step instructions, see [How Do I Upload Files and Folders to an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Step 2: Create IAM Users and a Group

Now use the IAM console to add two IAM users, Alice and Bob, to your AWS account. Also create an administrative group named `Consultants`, and then add both users to the group.

Warning

When you add users and a group, do not attach any policies that grant permissions to these users. At first, these users don't have any permissions. In the following sections, you grant permissions incrementally. First you must ensure that you have assigned passwords to these IAM

users. You use these user credentials to test Amazon S3 actions and verify that the permissions work as expected.

For step-by-step instructions for creating a new IAM user, see [Creating an IAM User in Your AWS Account](#) in the *IAM User Guide*. When you create the users for this walkthrough, select **AWS Management Console access** and clear **Programmatic access**.

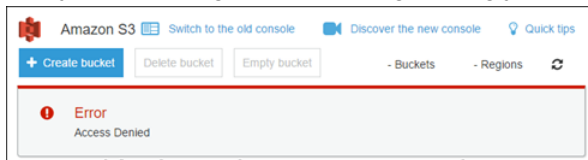
For step-by-step instructions for creating an administrative group, see [Creating Your First IAM Admin User and Group](#) in the *IAM User Guide*.

Step 3: Verify That IAM Users Have No Permissions

If you are using two browsers, you can now use the second browser to sign in to the console using one of the IAM user credentials.

1. Using the IAM user sign-in link (see [To Provide a Sign-In Link for IAM Users \(p. 388\)](#)), sign in to the AWS Management Console using either of the IAM user credentials.
2. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Verify the following console message telling you that access is denied.



Now, you can begin granting incremental permissions to the users. First, you attach a group policy that grants permissions that both users must have.

Step 4: Grant Group-Level Permissions

You want the users to be able to do the following:

- List all buckets owned by the parent account. To do so, Bob and Alice must have permission for the `s3:ListAllMyBuckets` action.
- List root-level items, folders, and objects in the `companybucket` bucket. To do so, Bob and Alice must have permission for the `s3:ListBucket` action on the `companybucket` bucket.

First, you create a policy that grants these permissions, and then you attach it to the `Consultants` group.

Step 4.1: Grant Permission to List All Buckets

In this step, you create a managed policy that grants the users minimum permissions to enable them to list all buckets owned by the parent account. Then you attach the policy to the `Consultants` group. When you attach the managed policy to a user or a group, you grant the user or group permission to obtain a list of buckets owned by the parent AWS account.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

Note

Because you are granting user permissions, sign in using your AWS account credentials, not as an IAM user.

2. Create the managed policy.

- a. In the navigation pane on the left, choose **Policies**, and then choose **Create Policy**.
- b. Choose the **JSON** tab.
- c. Copy the following access policy and paste it into the policy text field.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGroupToSeeBucketListInTheConsole",
      "Action": ["s3:ListAllMyBuckets"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::*"]
    }
  ]
}
```

A policy is a JSON document. In the document, a `Statement` is an array of objects, each describing a permission using a collection of name-value pairs. The preceding policy describes one specific permission. The `Action` specifies the type of access. In the policy, the `s3:ListAllMyBuckets` is a predefined Amazon S3 action. This action covers the Amazon S3 GET Service operation, which returns list of all buckets owned by the authenticated sender. The `Effect` element value determines whether specific permission is allowed or denied.

- d. Choose **Review Policy**. On the next page, enter `AllowGroupToSeeBucketListInTheConsole` in the **Name** field, and then choose **Create policy**.

Note

The **Summary** entry displays a message stating that the policy does not grant any permissions. For this walkthrough, you can safely ignore this message.

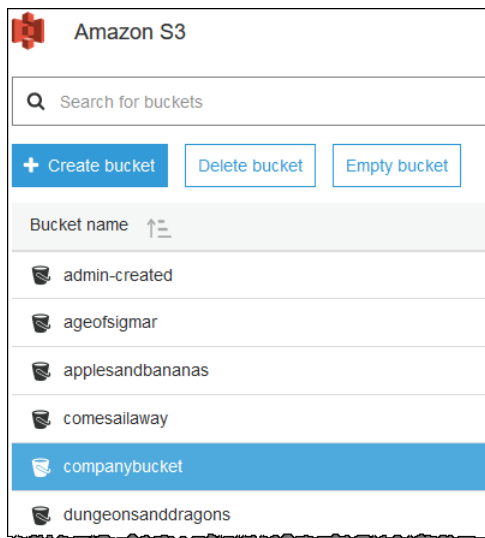
3. Attach the `AllowGroupToSeeBucketListInTheConsole` managed policy that you created to the `Consultants` group.

For step-by-step instructions for attaching a managed policy, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

You attach policy documents to IAM users and groups in the IAM console. Because you want both users to be able to list the buckets, you attach the policy to the group.

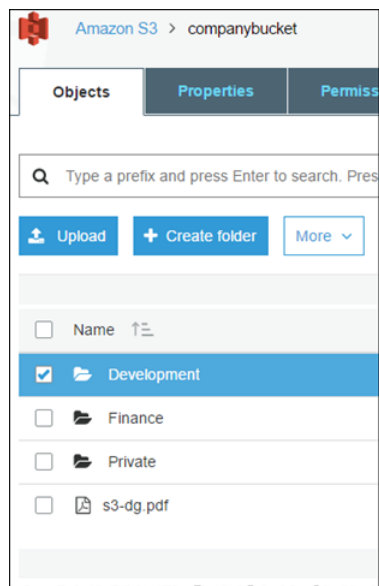
4. Test the permission.
 - a. Using the IAM user sign-in link (see [To Provide a Sign-In Link for IAM Users \(p. 388\)](#)), sign in to the console using any one of IAM user credentials.
 - b. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

The console should now list all the buckets but not the objects in any of the buckets.



Step 4.2: Enable Users to List Root-Level Content of a Bucket

Next, you allow all users in the `Consultants` group to list the root-level `companybucket` bucket items. When a user chooses the `companybucket` on the Amazon S3 console, the user can see the root-level items in the bucket.



Note

This example uses `companybucket` for illustration. You must use the name of the bucket that you created.

To understand the request that the console sends to Amazon S3 when you choose a bucket name, the response that Amazon S3 returns, and how the console interprets the response, it is necessary to examine it a little more closely.

When you choose a bucket name, the console sends the [GET Bucket \(List Objects\)](#) request to Amazon S3. This request includes the following parameters:

- The `prefix` parameter with an empty string as its value.
- The `delimiter` parameter with `/` as its value.

The following is an example request.

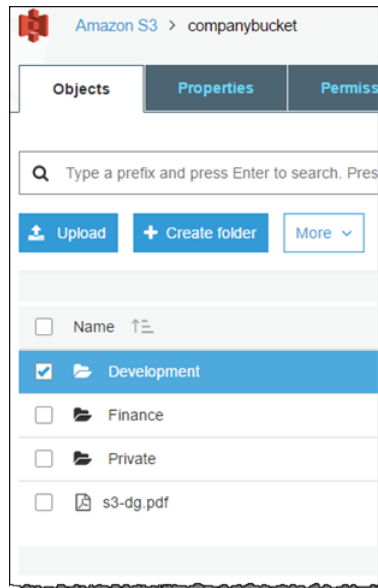
```
GET ?prefix=&delimiter=/ HTTP/1.1
Host: companybucket.s3.amazonaws.com
Date: Wed, 01 Aug 2012 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

Amazon S3 returns a response that includes the following `<ListBucketResult/>` element.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>companybucket</Name>
  <Prefix></Prefix>
  <Delimiter></Delimiter>
  ...
  <Contents>
    <Key>s3-dg.pdf</Key>
    ...
  </Contents>
  <CommonPrefixes>
    <Prefix>Development</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>Finance</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>Private</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

The key `s3-dg.pdf` object does not contain the slash (`/`) delimiter, and Amazon S3 returns the key in the `<Contents>` element. However, all other keys in the example bucket contain the `/` delimiter. Amazon S3 groups these keys and returns a `<CommonPrefixes>` element for each of the distinct prefix values `Development/`, `Finance/`, and `Private/` that is a substring from the beginning of these keys to the first occurrence of the specified `/` delimiter.

The console interprets this result and displays the root-level items as three folders and one object key.



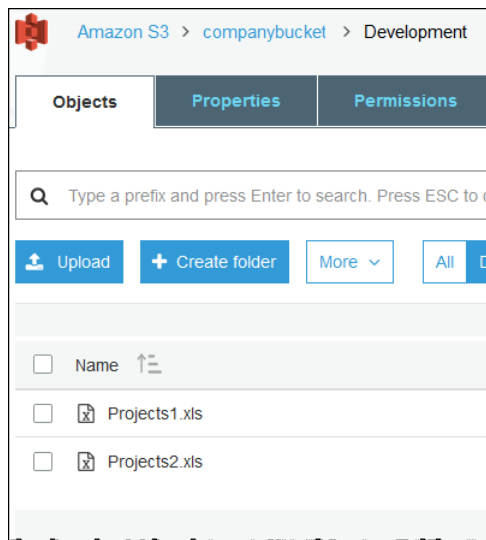
If Bob or Alice opens the **Development** folder, the console sends the [GET Bucket \(List Objects\)](#) request to Amazon S3 with the `prefix` and the `delimiter` parameters set to the following values:

- The `prefix` parameter with the value `Development/`.
- The `delimiter` parameter with the `"/` value.

In response, Amazon S3 returns the object keys that start with the specified prefix.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>companybucket</Name>
  <Prefix>Development</Prefix>
  <Delimiter>/</Delimiter>
  ...
  <Contents>
    <Key>Project1.xls</Key>
    ...
  </Contents>
  <Contents>
    <Key>Project2.xls</Key>
    ...
  </Contents>
</ListBucketResult>
```

The console shows the object keys.



Now, return to granting users permission to list the root-level bucket items. To list bucket content, users need permission to call the `s3:ListBucket` action, as shown in the following policy statement. To ensure that they see only the root-level content, you add a condition that users must specify an empty prefix in the request—that is, they are not allowed to double-click any of the root-level folders. Finally, you add a condition to require folder-style access by requiring user requests to include the `delimiter` parameter with the value `"/`.

```
{
  "Sid": "AllowRootLevelListingOfCompanyBucket",
  "Action": ["s3:ListBucket"],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::companybucket"],
  "Condition": {
    "StringEquals": {
      "s3:prefix": [""], "s3:delimiter": ["/"]
    }
  }
}
```

When you choose a bucket on the Amazon S3 console, the console first sends the [GET Bucket location](#) request to find the AWS Region where the bucket is deployed. Then the console uses the Region-specific endpoint for the bucket to send the [GET Bucket \(List Objects\)](#) request. As a result, if users are going to use the console, you must grant permission for the `s3:GetBucketLocation` action as shown in the following policy statement.

```
{
  "Sid": "RequiredByS3Console",
  "Action": ["s3:GetBucketLocation"],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::*"]
}
```

To enable users to list root-level bucket content

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

2. Replace the existing `AllowGroupToSeeBucketListInTheConsole` managed policy that is attached to the `Consultants` group with the following policy, which also allows the `s3:ListBucket` action. Remember to replace `companybucket` in the policy `Resource` with the name of your bucket.

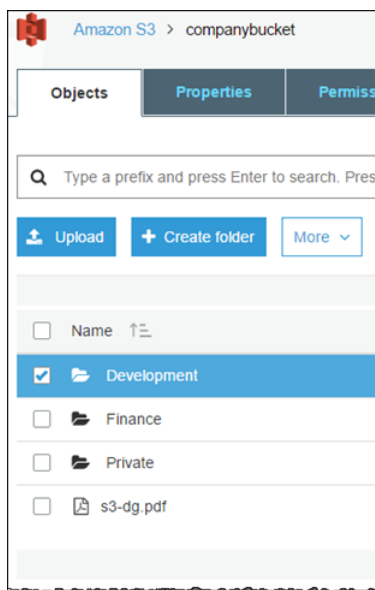
For step-by-step instructions, see [Editing IAM Policies](#) in the *IAM User Guide*. When following the step-by-step instructions, be sure to follow the steps for applying your changes to all principal entities that the policy is attached to.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":
"AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequiredForListBucket",
      "Action": [ "s3:ListAllMyBuckets", "s3:GetBucketLocation" ],
      "Effect": "Allow",
      "Resource": [ "arn:aws:s3::*" ]
    },
    {
      "Sid": "AllowRootLevelListingOfCompanyBucket",
      "Action": [ "s3:ListBucket" ],
      "Effect": "Allow",
      "Resource": [ "arn:aws:s3:::companybucket" ],
      "Condition": {
        "StringEquals": {
          "s3:prefix": [ "" ], "s3:delimiter": [ "/" ]
        }
      }
    }
  ]
}
```

3. Test the updated permissions.
 - a. Using the IAM user sign-in link (see [To Provide a Sign-In Link for IAM Users \(p. 388\)](#)), sign in to the AWS Management Console.

Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

- b. Choose the bucket that you created, and the console shows the root-level bucket items. If you choose any folders in the bucket, you won't be able to see the folder content because you haven't yet granted those permissions.



This test succeeds when users use the Amazon S3 console. When you choose a bucket on the console, the console implementation sends a request that includes the `prefix` parameter with an empty string as its value and the `delimiter` parameter with `"/` as its value.

Step 4.3: Summary of the Group Policy

The net effect of the group policy that you added is to grant the IAM users Alice and Bob the following minimum permissions:

- List all buckets owned by the parent account.
- See root-level items in the `companybucket` bucket.

However, the users still can't do much. Next, you grant user-specific permissions, as follows:

- Allow Alice to get and put objects in the `Development` folder.
- Allow Bob to get and put objects in the `Finance` folder.

For user-specific permissions, you attach a policy to the specific user, not to the group. In the following section, you grant Alice permission to work in the `Development` folder. You can repeat the steps to grant similar permission to Bob to work in the `Finance` folder.

Step 5: Grant IAM User Alice Specific Permissions

Now you grant additional permissions to Alice so that she can see the content of the `Development` folder and get and put objects in that folder.

Step 5.1: Grant IAM User Alice Permission to List the Development Folder Content

For Alice to list the `Development` folder content, you must apply a policy to the Alice user that grants permission for the `s3:ListBucket` action on the `companybucket` bucket, provided the request includes the prefix `Development/`. You want this policy to be applied only to the user Alice, so you use an inline policy. For more information about inline policies, see [Managed Policies and Inline Policies](#) in the *IAM User Guide*.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

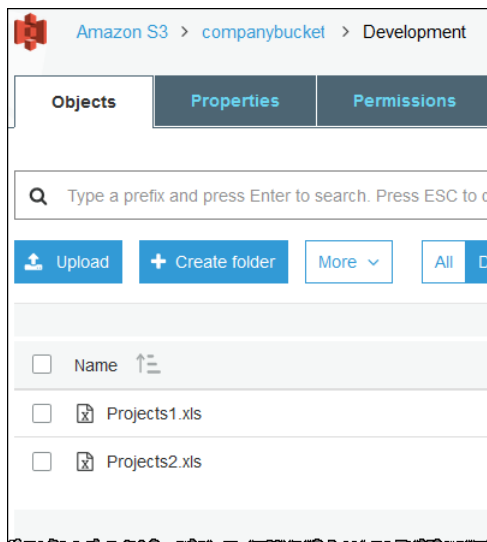
Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

2. Create an inline policy to grant the user Alice permission to list the Development folder content.
 - a. In the navigation pane on the left, choose **Users**.
 - b. Choose the user name **Alice**.
 - c. On the user details page, choose the **Permissions** tab and then choose **Add inline policy**.
 - d. Choose the **JSON** tab.
 - e. Copy the following policy and paste it into the policy text field.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition": {
        "StringLike": {
          "s3:prefix": ["Development/*"]
        }
      }
    }
  ]
}
```

- f. Choose **Review Policy**. On the next page, enter a name in the **Name** field, and then choose **Create policy**.
3. Test the change to Alice's permissions:
 - a. Using the IAM user sign-in link (see [To Provide a Sign-In Link for IAM Users \(p. 388\)](#)), sign in to the AWS Management Console.
 - b. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
 - c. On the Amazon S3 console, verify that Alice can see the list of objects in the Development/ folder in the bucket.

When the user chooses the /Development folder to see the list of objects in it, the Amazon S3 console sends the `ListObjects` request to Amazon S3 with the prefix `/Development`. Because the user is granted permission to see the object list with the prefix `Development` and delimiter `/`, Amazon S3 returns the list of objects with the key prefix `Development/`, and the console displays the list.



Step 5.2: Grant IAM User Alice Permissions to Get and Put Objects in the Development Folder

For Alice to get and put objects in the Development folder, she needs permission to call the `s3:GetObject` and `s3:PutObject` actions. The following policy statements grant these permissions, provided that the request includes the `prefix` parameter with a value of `Development/`.

```
{
  "Sid": "AllowUserToReadWriteObjectData",
  "Action": ["s3:GetObject", "s3:PutObject"],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::companybucket/Development/*"]
}
```

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

2. Edit the inline policy that you created in the previous step.
 - a. In the navigation pane on the left, choose **Users**.
 - b. Choose the user name Alice.
 - c. On the user details page, choose the **Permissions** tab and expand the **Inline Policies** section.
 - d. Next to the name of the policy that you created in the previous step, choose **Edit Policy**.
 - e. Copy the following policy and paste it into the policy text field, replacing the existing policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition": {
        "StringLike": {"s3:prefix": ["Development/*"]}
      }
    }
  ]
}
```



```
    },  
    {  
      "Sid": "AllowUserToReadWriteObjectDataInDevelopmentFolder",  
      "Action": ["s3:GetObject", "s3:PutObject"],  
      "Effect": "Allow",  
      "Resource": ["arn:aws:s3:::companybucket/Development/*"]  
    }  
  ]  
}
```

3. Test the updated policy:

- Using the IAM user sign-in link (see [To Provide a Sign-In Link for IAM Users \(p. 388\)](#)), sign into the AWS Management Console.
- Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- On the Amazon S3 console, verify that Alice can now add an object and download an object in the Development folder.

Step 5.3: Explicitly Deny IAM User Alice Permissions to Any Other Folders in the Bucket

User Alice can now list the root-level content in the companybucket bucket. She can also get and put objects in the Development folder. If you really want to tighten the access permissions, you could explicitly deny Alice access to any other folders in the bucket. If there is any other policy (bucket policy or ACL) that grants Alice access to any other folders in the bucket, this explicit deny overrides those permissions.

You can add the following statement to the user Alice policy that requires all requests that Alice sends to Amazon S3 to include the prefix parameter, whose value can be either Development/* or an empty string.

```
{  
  "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",  
  "Action": ["s3:ListBucket"],  
  "Effect": "Deny",  
  "Resource": ["arn:aws:s3:::companybucket"],  
  "Condition": {  
    "StringNotLike": { "s3:prefix": ["Development/*", ""] },  
    "Null": { "s3:prefix": false }  
  }  
}
```

There are two conditional expressions in the Condition block. The result of these conditional expressions is combined by using the logical AND. If both conditions are true, the result of the combined condition is true. Because the Effect in this policy is Deny, when the Condition evaluates to true, users can't perform the specified Action.

- The Null conditional expression ensures that requests from Alice include the prefix parameter.

The prefix parameter requires folder-like access. If you send a request without the prefix parameter, Amazon S3 returns all the object keys.

If the request includes the prefix parameter with a null value, the expression evaluates to true, and so the entire Condition evaluates to true. You must allow an empty string as value of the prefix parameter. From the preceding discussion, recall that allowing the null string allows Alice to retrieve root-level bucket items as the console does in the preceding discussion. For more information, see [Step 4.2: Enable Users to List Root-Level Content of a Bucket \(p. 391\)](#).

- The StringNotLike conditional expression ensures that if the value of the prefix parameter is specified and is not Development/*, the request fails.

Follow the steps in the preceding section and again update the inline policy that you created for user Alice.

Copy the following policy and paste it into the policy text field, replacing the existing policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition": {
        "StringLike": {"s3:prefix": ["Development/*"]}
      }
    },
    {
      "Sid": "AllowUserToReadWriteObjectDataInDevelopmentFolder",
      "Action": ["s3:GetObject", "s3:PutObject"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket/Development/*"]
    },
    {
      "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",
      "Action": ["s3:ListBucket"],
      "Effect": "Deny",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition": {
        "StringNotLike": {"s3:prefix": ["Development/*", "" ]},
        "Null": {"s3:prefix": false }
      }
    }
  ]
}
```

Step 6: Grant IAM User Bob Specific Permissions

Now you want to grant Bob permission to the `Finance` folder. Follow the steps that you used earlier to grant permissions to Alice, but replace the `Development` folder with the `Finance` folder. For step-by-step instructions, see [Step 5: Grant IAM User Alice Specific Permissions \(p. 396\)](#).

Step 7: Secure the Private Folder

In this example, you have only two users. You granted all the minimum required permissions at the group level and granted user-level permissions only when you really need to permissions at the individual user level. This approach helps minimize the effort of managing permissions. As the number of users increases, managing permissions can become cumbersome. For example, you don't want any of the users in this example to access the content of the `Private` folder. How do you ensure that you don't accidentally grant a user permission to it? You add a policy that explicitly denies access to the folder. An explicit deny overrides any other permissions.

To ensure that the `Private` folder remains private, you can add the following two deny statements to the group policy:

- Add the following statement to explicitly deny any action on resources in the `Private` folder (`companybucket/Private/*`).

```
{
  "Sid": "ExplicitDenyAccessToPrivateFolderToEveryoneInTheGroup",
  "Action": ["s3:*"],
  "Effect": "Deny",
```

```
}
  "Resource":["arn:aws:s3:::companybucket/Private/*"]
}
```

- You also deny permission for the list objects action when the request specifies the `Private/` prefix. On the console, if Bob or Alice opens the `Private` folder, this policy causes Amazon S3 to return an error response.

```
{
  "Sid": "DenyListBucketOnPrivateFolder",
  "Action": ["s3:ListBucket"],
  "Effect": "Deny",
  "Resource": ["arn:aws:s3:::*"],
  "Condition":{"
    "StringLike":{"s3:prefix":["Private/"]}
  }
}
```

Replace the `Consultants` group policy with an updated policy that includes the preceding deny statements. After the updated policy is applied, none of the users in the group can access the `Private` folder in your bucket.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

2. Replace the existing `AllowGroupToSeeBucketListInTheConsole` managed policy that is attached to the `Consultants` group with the following policy. Remember to replace `companybucket` in the policy with the name of your bucket.

For instructions, see [Editing Customer Managed Policies](#) in the *IAM User Guide*. When following the instructions, make sure to follow the directions for applying your changes to all principal entities that the policy is attached to.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":
"AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequiredForListBucket",
      "Action": ["s3:ListAllMyBuckets", "s3:GetBucketLocation"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::*"]
    },
    {
      "Sid": "AllowRootLevelListingOfCompanyBucket",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition":{"
        "StringEquals":{"s3:prefix":[""]}
      }
    },
    {
      "Sid": "RequireFolderStyleList",
      "Action": ["s3:ListBucket"],
      "Effect": "Deny",
      "Resource": ["arn:aws:s3:::*"],
      "Condition":{"
        "StringNotEquals":{"s3:delimiter":"/"}
      }
    }
  ]
}
```

```
{
  "Sid": "ExplicitDenyAccessToPrivateFolderToEveryoneInTheGroup",
  "Action": ["s3:*"],
  "Effect": "Deny",
  "Resource": ["arn:aws:s3:::companybucket/Private/*"]
},
{
  "Sid": "DenyListBucketOnPrivateFolder",
  "Action": ["s3:ListBucket"],
  "Effect": "Deny",
  "Resource": ["arn:aws:s3:::*"],
  "Condition": {
    "StringLike": {"s3:prefix": ["Private/"]}
  }
}
]
```

Step 8: Clean Up

To clean up, open the IAM console and remove the users Alice and Bob. For step-by-step instructions, see [Deleting an IAM User](#) in the *IAM User Guide*.

To ensure that you aren't charged further for storage, you should also delete the objects and the bucket that you created for this exercise.

Related Resources

- [Managing IAM Policies](#) in the *IAM User Guide*.

Managing Access with ACLs

Topics

- [Access Control List \(ACL\) Overview \(p. 403\)](#)
- [Managing ACLs \(p. 409\)](#)

Access control lists (ACLs) are one of the resource-based access policy options (see [Overview of Managing Access \(p. 302\)](#)) that you can use to manage access to your buckets and objects. You can use ACLs to grant basic read/write permissions to other AWS accounts. There are limits to managing permissions using ACLs. For example, you can grant permissions only to other AWS accounts; you cannot grant permissions to users in your account. You cannot grant conditional permissions, nor can you explicitly deny permissions. ACLs are suitable for specific scenarios. For example, if a bucket owner allows other AWS accounts to upload objects, permissions to these objects can only be managed using object ACL by the AWS account that owns the object.

The following introductory topics explain the basic concepts and options that are available for you to manage access to your Amazon S3 resources, and provide guidelines for when to use which access policy options.

- [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 301\)](#)
- [Guidelines for Using the Available Access Policy Options \(p. 312\)](#)

Access Control List (ACL) Overview

Topics

- [Who Is a Grantee? \(p. 404\)](#)
- [What Permissions Can I Grant? \(p. 405\)](#)
- [Sample ACL \(p. 407\)](#)
- [Canned ACL \(p. 408\)](#)
- [How to Specify an ACL \(p. 408\)](#)

Amazon S3 access control lists (ACLs) enable you to manage access to buckets and objects. Each bucket and object has an ACL attached to it as a subresource. It defines which AWS accounts or groups are granted access and the type of access. When a request is received against a resource, Amazon S3 checks the corresponding ACL to verify that the requester has the necessary access permissions.

When you create a bucket or an object, Amazon S3 creates a default ACL that grants the resource owner full control over the resource. This is shown in the following sample bucket ACL (the default object ACL has the same structure):

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="Canonical User">
```

```
<ID>*** Owner-Canonical-User-ID ***</ID>
<DisplayName>display-name</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

The sample ACL includes an `Owner` element that identifies the owner by the AWS account's canonical user ID. For instructions on finding your canonical user ID, see [Finding an AWS Account Canonical User ID](#) (p. 404). The `Grant` element identifies the grantee (either an AWS account or a predefined group) and the permission granted. This default ACL has one `Grant` element for the owner. You grant permissions by adding `Grant` elements, with each grant identifying the grantee and the permission.

Note

An ACL can have up to 100 grants.

Who Is a Grantee?

A grantee can be an AWS account or one of the predefined Amazon S3 groups. You grant permission to an AWS account using the email address or the canonical user ID. However, if you provide an email address in your grant request, Amazon S3 finds the canonical user ID for that account and adds it to the ACL. The resulting ACLs always contain the canonical user ID for the AWS account, not the AWS account's email address.

Important

Using email addresses to specify a grantee is only supported in the following AWS Regions:

- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- EU (Ireland)
- South America (São Paulo)

For a list of all the Amazon S3 supported regions and endpoints, see [Regions and Endpoints](#) in the *AWS General Reference*.

Warning

When you grant other AWS accounts access to your resources, be aware that the AWS accounts can delegate their permissions to users under their accounts. This is known as *cross-account access*. For information about using cross-account access, see [Creating a Role to Delegate Permissions to an IAM User](#) in the *IAM User Guide*.

Finding an AWS Account Canonical User ID

The canonical user ID is associated with your AWS account. It is a long string, such as `79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be`. For information about how to find the canonical user ID for your account, see [Finding Your Account Canonical User ID](#).

You can also look up the canonical user ID of an AWS account by reading the ACL of a bucket or an object to which the AWS account has access permissions. When an individual AWS account is granted permissions by a grant request, a grant entry is added to the ACL with the AWS account's canonical user ID.

Note

If you make your bucket public (not recommended) any unauthenticated user can upload objects to the bucket. These anonymous users don't have an AWS account. When an anonymous user uploads an object to your bucket Amazon S3 adds a special canonical user ID (65a011a29cdf8ec533ec3d1ccaae921c) as the object owner in the ACL. For more information, see [Amazon S3 Bucket and Object Ownership \(p. 302\)](#).

Amazon S3 Predefined Groups

Amazon S3 has a set of predefined groups. When granting account access to a group, you specify one of our URIs instead of a canonical user ID. We provide the following predefined groups:

- **Authenticated Users group** – Represented by `http://acs.amazonaws.com/groups/global/AuthenticatedUsers`.

This group represents all AWS accounts. **Access permission to this group allows any AWS account to access the resource.** However, all requests must be signed (authenticated).

Warning

When you grant access to the **Authenticated Users group** any AWS authenticated user in the world can access your resource.

- **All Users group** – Represented by `http://acs.amazonaws.com/groups/global/AllUsers`.

Access permission to this group allows anyone in the world access to the resource. The requests can be signed (authenticated) or unsigned (anonymous). Unsigned requests omit the Authentication header in the request.

Warning

We highly recommend that you never grant the **All Users group** `WRITE`, `WRITE_ACP`, or `FULL_CONTROL` permissions. For example, `WRITE` permissions allow anyone to store objects in your bucket, for which you are billed. It also allows others to delete objects that you might want to keep. For more details about these permissions, see the following section [What Permissions Can I Grant? \(p. 405\)](#).

- **Log Delivery group** – Represented by `http://acs.amazonaws.com/groups/s3/LogDelivery`.

`WRITE` permission on a bucket enables this group to write server access logs (see [Amazon S3 Server Access Logging \(p. 647\)](#)) to the bucket.

Note

When using ACLs, a grantee can be an AWS account or one of the predefined Amazon S3 groups. However, the grantee cannot be an IAM user. For more information about AWS users and permissions within IAM, go to [Using AWS Identity and Access Management](#).

What Permissions Can I Grant?

The following table lists the set of permissions that Amazon S3 supports in an ACL. The set of ACL permissions is the same for an object ACL and a bucket ACL. However, depending on the context (bucket ACL or object ACL), these ACL permissions grant permissions for specific buckets or object operations. The table lists the permissions and describes what they mean in the context of objects and buckets.

Permission	When granted on a bucket	When granted on an object
READ	Allows grantee to list the objects in the bucket	Allows grantee to read the object data and its metadata
WRITE	Allows grantee to create, overwrite, and delete any object in the bucket	Not applicable

Permission	When granted on a bucket	When granted on an object
READ_ACP	Allows grantee to read the bucket ACL	Allows grantee to read the object ACL
WRITE_ACP	Allows grantee to write the ACL for the applicable bucket	Allows grantee to write the ACL for the applicable object
FULL_CONTROL	Allows grantee the READ, WRITE, READ_ACP, and WRITE_ACP permissions on the bucket	Allows grantee the READ, READ_ACP, and WRITE_ACP permissions on the object

Warning

Use caution when granting access permissions to your S3 buckets and objects. For example, granting `WRITE` access to a bucket allows the grantee to create, overwrite, and delete any object in the bucket. We highly recommend that you read through this entire [Access Control List \(ACL\) Overview \(p. 403\)](#) section before granting permissions.

Mapping of ACL Permissions and Access Policy Permissions

As shown in the preceding table, an ACL allows only a finite set of permissions, compared to the number of permissions you can set in an access policy (see [Specifying Permissions in a Policy \(p. 345\)](#)). Each of these permissions allows one or more Amazon S3 operations.

The following table shows how each ACL permission maps to the corresponding access policy permissions. As you can see, access policy allows more permissions than ACL does. You use ACL primarily to grant basic read/write permissions, similar to file system permissions. For more information about when to use ACL, see [Guidelines for Using the Available Access Policy Options \(p. 312\)](#).

ACL permission	Corresponding access policy permissions when the ACL permission is granted on a bucket	Corresponding access policy permissions when the ACL permission is granted on an object
READ	<code>s3:ListBucket</code> , <code>s3:ListBucketVersions</code> , and <code>s3:ListBucketMultipartUploads</code>	<code>s3:GetObject</code> , <code>s3:GetObjectVersion</code> , and <code>s3:GetObjectTorrent</code>
WRITE	<code>s3:PutObject</code> and <code>s3:DeleteObject</code> . In addition, when the grantee is the bucket owner, granting <code>WRITE</code> permission in a bucket ACL allows the <code>s3:DeleteObjectVersion</code> action to be performed on any version in that bucket.	Not applicable
READ_ACP	<code>s3:GetBucketAcl</code>	<code>s3:GetObjectAcl</code> and <code>s3:GetObjectVersionAcl</code>
WRITE_ACP	<code>s3:PutBucketAcl</code>	<code>s3:PutObjectAcl</code> and <code>s3:PutObjectVersionAcl</code>
FULL_CONTROL	Equivalent to granting <code>READ</code> , <code>WRITE</code> , <code>READ_ACP</code> , and <code>WRITE_ACP</code> ACL permissions. Accordingly, this ACL permission maps to a combination of corresponding access policy permissions.	Equivalent to granting <code>READ</code> , <code>READ_ACP</code> , and <code>WRITE_ACP</code> ACL permissions. Accordingly, this ACL permission maps to a combination of corresponding access policy permissions.

Sample ACL

The following sample ACL on a bucket identifies the resource owner and a set of grants. The format is the XML representation of an ACL in the Amazon S3 REST API. The bucket owner has `FULL_CONTROL` of the resource. In addition, the ACL shows how permissions are granted on a resource to two AWS accounts, identified by canonical user ID, and two of the predefined Amazon S3 groups discussed in the preceding section.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>Owner-canonical-user-ID</ID>
    <DisplayName>display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>Owner-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>user1-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>user2-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

Canned ACL

Amazon S3 supports a set of predefined grants, known as canned ACLs. Each canned ACL has a predefined set of grantees and permissions. The following table lists the set of canned ACLs and the associated predefined grants.

Canned ACL	Applies to	Permissions added to ACL
private	Bucket and object	Owner gets <code>FULL_CONTROL</code> . No one else has access rights (default).
public-read	Bucket and object	Owner gets <code>FULL_CONTROL</code> . The <code>AllUsers</code> group (see Who Is a Grantee? (p. 404)) gets <code>READ</code> access.
public-read-write	Bucket and object	Owner gets <code>FULL_CONTROL</code> . The <code>AllUsers</code> group gets <code>READ</code> and <code>WRITE</code> access. Granting this on a bucket is generally not recommended.
aws-exec-read	Bucket and object	Owner gets <code>FULL_CONTROL</code> . Amazon EC2 gets <code>READ</code> access to <code>GET</code> an Amazon Machine Image (AMI) bundle from Amazon S3.
authenticated-read	Bucket and object	Owner gets <code>FULL_CONTROL</code> . The <code>AuthenticatedUsers</code> group gets <code>READ</code> access.
bucket-owner-read	Object	Object owner gets <code>FULL_CONTROL</code> . Bucket owner gets <code>READ</code> access. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.
bucket-owner-full-control	Object	Both the object owner and the bucket owner get <code>FULL_CONTROL</code> over the object. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.
log-delivery-write	Bucket	The <code>LogDelivery</code> group gets <code>WRITE</code> and <code>READ_ACP</code> permissions on the bucket. For more information about logs, see (Amazon S3 Server Access Logging (p. 647)) .

Note

You can specify only one of these canned ACLs in your request.

You specify a canned ACL in your request using the `x-amz-acl` request header. When Amazon S3 receives a request with a canned ACL in the request, it adds the predefined grants to the ACL of the resource.

How to Specify an ACL

Amazon S3 APIs enable you to set an ACL when you create a bucket or an object. Amazon S3 also provides API to set an ACL on an existing bucket or an object. These APIs provide the following methods to set an ACL:

- **Set ACL using request headers**— When you send a request to create a resource (bucket or object), you set an ACL using the request headers. Using these headers, you can either specify a canned ACL or specify grants explicitly (identifying grantee and permissions explicitly).
- **Set ACL using request body**— When you send a request to set an ACL on an existing resource, you can set the ACL either in the request header or in the body.

For more information, see [Managing ACLs \(p. 409\)](#).

Managing ACLs

Topics

- [Managing ACLs in the AWS Management Console \(p. 409\)](#)
- [Managing ACLs Using the AWS SDK for Java \(p. 409\)](#)
- [Managing ACLs Using the AWS SDK for .NET \(p. 411\)](#)
- [Managing ACLs Using the REST API \(p. 414\)](#)

There are several ways you can add grants to your resource ACL. You can use the AWS Management Console, which provides a UI to manage permissions without writing any code. You can use the REST API or one of the AWS SDKs. These libraries further simplify your programming tasks.

Managing ACLs in the AWS Management Console

AWS Management Console provides a UI for you to grant ACL-based access permissions to your buckets and objects. For information on setting ACL-based access permissions in the console, see [How Do I Set ACL Bucket Permissions?](#) and [How Do I Set Permissions on an Object?](#) in the *Amazon Simple Storage Service Console User Guide*.

Managing ACLs Using the AWS SDK for Java

This section provides examples of how to configure access control list (ACL) grants on buckets and objects. The first example creates a bucket with a canned ACL (see [Canned ACL \(p. 408\)](#)), creates a list of custom permission grants, and then replaces the canned ACL with an ACL containing the custom grants. The second example shows how to modify an ACL using the `AccessControlList.grantPermission()` method.

Setting ACL Grants

Example

This example creates a bucket. In the request, the example specifies a canned ACL that grants the Log Delivery group permission to write logs to the bucket.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.IOException;
import java.util.ArrayList;

public class CreateBucketWithACL {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String userEmailForReadPermission = "*** user@example.com ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .build();

            // Create a bucket with a canned ACL. This ACL will be replaced by the
            setBucketAcl()
```

```

        // calls below. It is included here for demonstration purposes.
        CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucketName,
clientRegion.getName())
            .withCannedAcl(CannedAccessControlList.LogDeliveryWrite);
        s3Client.createBucket(createBucketRequest);

        // Create a collection of grants to add to the bucket.
        ArrayList<Grant> grantCollection = new ArrayList<Grant>();

        // Grant the account owner full control.
        Grant grant1 = new Grant(new
CanonicalGrantee(s3Client.getS3AccountOwner().getId()), Permission.FullControl);
        grantCollection.add(grant1);

        // Grant the LogDelivery group permission to write to the bucket.
        Grant grant2 = new Grant(GroupGrantee.LogDelivery, Permission.Write);
        grantCollection.add(grant2);

        // Save grants by replacing all current ACL grants with the two we just
created.
        AccessControlList bucketAcl = new AccessControlList();
        bucketAcl.grantAllPermissions(grantCollection.toArray(new Grant[0]));
        s3Client.setBucketAcl(bucketName, bucketAcl);

        // Retrieve the bucket's ACL, add another grant, and then save the new ACL.
        AccessControlList newBucketAcl = s3Client.getBucketAcl(bucketName);
        Grant grant3 = new Grant(new EmailAddressGrantee(userEmailForReadPermission),
Permission.Read);
        newBucketAcl.grantAllPermissions(grant3);
        s3Client.setBucketAcl(bucketName, newBucketAcl);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it and returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
}
}

```

Configuring ACL Grants on an Existing Object

Example

This example updates the ACL on an object. The example performs the following tasks:

- Retrieves an object's ACL
- Clears the ACL by removing all existing permissions
- Adds two permissions: full access to the owner, and WRITE_ACP (see [What Permissions Can I Grant? \(p. 405\)](#)) to a user identified by an email address
- Saves the ACL to the object

```

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;

```

```
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.CanonicalGrantee;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
import com.amazonaws.services.s3.model.Permission;

import java.io.IOException;

public class ModifyACLExistingObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";
        String emailGrantee = "*** user@example.com ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Get the existing object ACL that we want to modify.
            AccessControlList acl = s3Client.getObjectAcl(bucketName, keyName);

            // Clear the existing list of grants.
            acl.getGrantsAsList().clear();

            // Grant a sample set of permissions, using the existing ACL owner for Full
            // Control permissions.
            acl.grantPermission(new CanonicalGrantee(acl.getOwner().getId()),
                Permission.FullControl);
            acl.grantPermission(new EmailAddressGrantee(emailGrantee),
                Permission.WriteAcp);

            // Save the modified ACL back to the object.
            s3Client.setObjectAcl(bucketName, keyName, acl);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Managing ACLs Using the AWS SDK for .NET

This section provides examples of configuring ACL grants on Amazon S3 buckets and objects.

Example 1: Creating a Bucket and Using a Canned ACL to Set Permissions

This C# example creates a bucket. In the request, the code also specifies a canned ACL that grants the Log Delivery group permissions to write the logs to the bucket.

For instructions on creating and testing a working example, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ManagingBucketACLTest
    {
        private const string newBucketName = "**** bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            CreateBucketUseCannedACLAsync().Wait();
        }

        private static async Task CreateBucketUseCannedACLAsync()
        {
            try
            {
                // Add bucket (specify canned ACL).
                PutBucketRequest putBucketRequest = new PutBucketRequest()
                {
                    BucketName = newBucketName,
                    BucketRegion = bucketRegion, // S3Region.US,
                                                // Add canned ACL.
                    CannedACL = S3CannedACL.LogDeliveryWrite
                };
                PutBucketResponse putBucketResponse = await
client.PutBucketAsync(putBucketRequest);

                // Retrieve bucket ACL.
                GetACLResponse getACLResponse = await client.GetACLAsync(new GetACLRequest
                {
                    BucketName = newBucketName
                });
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " +
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }
    }
}
```

Example 2: Configure ACL Grants on an Existing Object

This C# example updates the ACL on an existing object. The example performs the following tasks:

- Retrieves an object's ACL.
- Clears the ACL by removing all existing permissions.
- Adds two permissions: full access to the owner, and WRITE_ACP to a user identified by email address.
- Saves the ACL by sending a PutAcl request.

For instructions on creating and testing a working example, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ManagingObjectACLTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** object key name ***";
        private const string emailAddress = "*** email address ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;
        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            TestObjectACLTestAsync().Wait();
        }
        private static async Task TestObjectACLTestAsync()
        {
            try
            {
                // Retrieve the ACL for the object.
                GetACLResponse aclResponse = await client.GetACLAsync(new GetACLRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                });

                S3AccessControlList acl = aclResponse.AccessControlList;

                // Retrieve the owner (we use this to re-add permissions after we clear
                the ACL).
                Owner owner = acl.Owner;

                // Clear existing grants.
                acl.Grants.Clear();

                // Add a grant to reset the owner's full permission (the previous clear
                statement removed all permissions).
                S3Grant fullControlGrant = new S3Grant
                {
                    Grantee = new S3Grantee { CanonicalUser = owner.Id },
                    Permission = S3Permission.FULL_CONTROL
                };

                // Describe the grant for the permission using an email address.
                S3Grant grantUsingEmail = new S3Grant
                {
                    Grantee = new S3Grantee { EmailAddress = emailAddress },
                    Permission = S3Permission.WRITE_ACP
                };
                acl.Grants.AddRange(new List<S3Grant> { fullControlGrant,
                grantUsingEmail });

                // Set a new ACL.
                PutACLResponse response = await client.PutACLAsync(new PutACLRequest
```

```
        {
            BucketName = bucketName,
            Key = keyName,
            AccessControlList = acl
        });
    }
    catch (AmazonS3Exception amazonS3Exception)
    {
        Console.WriteLine("An AmazonS3Exception was thrown. Exception: " +
amazonS3Exception.ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine("Exception: " + e.ToString());
    }
}
}
```

Managing ACLs Using the REST API

For information on the REST API support for managing ACLs, see the following sections in the *Amazon Simple Storage Service API Reference*:

- [GET Bucket acl](#)
- [PUT Bucket acl](#)
- [GET Object acl](#)
- [PUT Object acl](#)
- [PUT Object](#)
- [PUT Bucket](#)
- [PUT Object - Copy](#)
- [Initiate Multipart Upload](#)

Using Amazon S3 Block Public Access

Amazon S3 provides *block public access* settings for buckets and accounts to help you manage public access to Amazon S3 resources. By default, new buckets and objects don't allow public access, but users can modify bucket policies or object permissions to allow public access. Amazon S3 block public access settings override these policies and permissions so that you can limit public access to these resources. With Amazon S3 block public access, account administrators and bucket owners can easily set up centralized controls to limit public access to their Amazon S3 resources that are enforced regardless of how the resources are created.

When Amazon S3 receives a request to access a bucket or an object, it determines whether the bucket or the bucket owner's account has a block public access setting applied. If there is an existing block public access setting that prohibits the requested access, then Amazon S3 rejects the request. Amazon S3 block public access provides four settings. These settings are independent and can be used in any combination. And each setting can be applied to a bucket or to an entire AWS account. If a bucket has block public access settings that are different from its owner's account, Amazon S3 applies the most restrictive combination of the bucket-level and account-level settings. When Amazon S3 evaluates whether an operation is prohibited by a block public access setting, it rejects any request that violates either a bucket-level or an account-level setting.

Note

- You can enable block public access settings only for buckets and AWS accounts. Amazon S3 doesn't support block public access settings on a per-object basis.

- When you apply block public access settings to an account, the settings apply to all AWS Regions globally. The settings might not take effect in all Regions immediately or simultaneously, but they eventually propagate to all Regions.

Topics

- [Enable Block Public Access on the Amazon S3 Console](#) (p. 415)
- [Block Public Access Settings](#) (p. 416)
- [The Meaning of "Public"](#) (p. 417)
- [Permissions](#) (p. 418)
- [Examples](#) (p. 419)

Enable Block Public Access on the Amazon S3 Console

Amazon S3 Block public access provides four settings. You can apply these settings in any combination to individual buckets or to entire AWS accounts. The image below shows how to enable block public access on the Amazon S3 console for your account. For more information, see [Setting Permissions: Block Public Access](#) in the *Amazon Simple Storage Service Console User Guide*.

Block public access (account settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, or both. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block *all* public access. These settings apply account-wide for all current and future buckets. AWS recommends that you turn on Block *all* public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☒ **Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

☐ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**

S3 will ignore all ACLs that grant public access to buckets and objects.

☐ **Block public access to buckets and objects granted through *new* public bucket policies**

S3 will block new bucket policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☐ **Block public and cross-account access to buckets and objects through *any* public bucket policies**

S3 will ignore public and cross-account access for buckets with policies that grant public access to buckets and objects.

Cancel

Save

Block Public Access Settings

Amazon S3 block public access provides four settings. You can apply these settings in any combination to individual buckets or to entire AWS accounts. If you apply a setting to an account, it applies to all buckets that are owned by that account.

The following table contains the available settings.

Name	Description
<code>BlockPublicAcls</code>	<p>Setting this option to <code>TRUE</code> causes the following behavior:</p> <ul style="list-style-type: none">• PUT Bucket acl and PUT Object acl calls fail if the specified access control list (ACL) is public.• PUT Object calls fail if the request includes a public ACL.• If this setting is applied to an account, then PUT Bucket calls fail if the request includes a public ACL. <p>When this setting is set to <code>TRUE</code>, the specified operations fail (whether made through the REST API, AWS CLI, or AWS SDKs). However, existing policies and ACLs for buckets and objects are not modified. This setting enables you to protect against public access while allowing you to audit, refine, or otherwise alter the existing policies and ACLs for your buckets and objects.</p>
<code>IgnorePublicAcls</code>	<p>Setting this option to <code>TRUE</code> causes Amazon S3 to ignore all public ACLs on a bucket and any objects that it contains. This setting enables you to safely block public access granted by ACLs while still allowing PUT Object calls that include a public ACL (as opposed to <code>BlockPublicAcls</code>, which rejects PUT Object calls that include a public ACL). Enabling this setting doesn't affect the persistence of any existing ACLs and doesn't prevent new public ACLs from being set.</p>
<code>BlockPublicPolicy</code>	<p>Setting this option to <code>TRUE</code> causes Amazon S3 to reject calls to PUT Bucket policy if the specified bucket policy allows public access. This setting enables you to allow users to manage bucket policies without allowing them to publicly share the bucket or the objects it contains. Enabling this setting doesn't affect existing bucket policies.</p> <p>Important</p> <p>To use this setting effectively, you should apply it at the <i>account</i> level. A bucket policy can allow users to alter a bucket's block public access settings. Therefore, users who have permission to change a bucket policy could insert a policy that allows them to disable the block public access settings for the bucket. If this setting is enabled for the entire account, rather than for a specific bucket, Amazon S3 blocks public policies even if a user alters the bucket policy to disable this setting.</p>
<code>RestrictPublicBuckets</code>	<p>Setting this option to <code>TRUE</code> restricts access to a bucket with a public policy to only AWS services and authorized users within the bucket owner's account. This setting blocks all cross-account access to the bucket (except by AWS services), while still allowing users within the account to manage the bucket.</p> <p>Enabling this setting doesn't affect existing bucket policies, except that Amazon S3 blocks public and cross-account access derived from any public bucket policy, including non-public delegation to specific accounts.</p>

Important

- Calls to GET Bucket acl and GET Object acl always return the effective permissions in place for the specified bucket or object. For example, suppose that a bucket has an ACL that grants public access, but the bucket also has the `IgnorePublicAcls` setting enabled. In this case, GET Bucket acl returns an ACL that reflects the access permissions that Amazon S3 is enforcing, rather than the actual ACL that is associated with the bucket.
- Block public access settings don't alter existing policies or ACLs. Therefore, removing a block public access setting causes a bucket or object with a public policy or ACL to again be publicly accessible.

The Meaning of "Public"

ACLs

Amazon S3 considers a bucket or object ACL public if it grants any permissions to members of the predefined `AllUsers` or `AuthenticatedUsers` groups. For more information about predefined groups, see [Amazon S3 Predefined Groups \(p. 405\)](#).

Policies

When evaluating a bucket policy, Amazon S3 begins by assuming that the policy is public. It then evaluates the policy to determine whether it qualifies as non-public. To be considered non-public, a bucket policy must grant access only to fixed values (values that don't contain a wildcard) of one or more of the following:

- A set of Classless Inter-Domain Routings (CIDRs), using `aws:SourceIp`. For more information about CIDR, see [RFC 4632](#) on the RFC Editor website.
- An AWS principal, user, role, or service principal (e.g. `aws:PrincipalOrgID`)
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:SourceOwner`
- `aws:SourceAccount`
- `s3:x-amz-server-side-encryption-aws-kms-key-id`
- `aws:userid`, outside the pattern `"AROLEID:*`

Under these rules, the following example policies are considered public:

```
{
  "Principal": { "Federated": "graph.facebook.com" },
  "Resource": "*",
  "Action": "s3:PutObject",
  "Effect": "Allow"
}
```

```
{
  "Principal": "*",
  "Resource": "*",
  "Action": "s3:PutObject",
  "Effect": "Allow"
}
```

```
{
  "Principal": "*",
  "Resource": "*",
  "Action": "s3:PutObject", API Version 2006-03-01
```

```
"Effect": "Allow",  
"Condition": { "StringLike": { "aws:SourceVpc": "vpc-*" } }  
}
```

You can make these policies non-public by including any of the condition keys listed previously, using a fixed value. For example, you can make the last policy preceding non-public by setting `aws:SourceVpc` to a fixed value, like this:

```
{  
  "Principal": "*",  
  "Resource": "*",  
  "Action": "s3:PutObject",  
  "Effect": "Allow",  
  "Condition": { "StringEquals": { "aws:SourceVpc": "vpc-91237329" } }  
}
```

For more information about bucket policies, see [Using Bucket Policies and User Policies \(p. 341\)](#).

Example

This example shows how Amazon S3 evaluates a policy that contains both public and non-public access grants.

Suppose that a bucket has a policy that grants access to a set of fixed principals. Under the previously described rules, this policy isn't public. Thus, if you enable the `RestrictPublicBuckets` setting, the policy remains in effect as written, because `RestrictPublicBuckets` only applies to buckets that have public policies. However, if you add a public statement to the policy, `RestrictPublicBuckets` takes effect on the bucket. It allows only AWS service principals and authorized users of the bucket owner's account to access the bucket.

As an example, suppose that a bucket owned by "Account-1" has a policy that contains the following:

1. A statement that grants access to AWS CloudTrail (which is an AWS service principal)
2. A statement that grants access to account "Account-2"
3. A statement that grants access to the public, for example by specifying `"Principal": "*" with no limiting Condition`

This policy qualifies as public because of the third statement. With this policy in place and `RestrictPublicBuckets` enabled, Amazon S3 allows access only by CloudTrail. Even though statement 2 isn't public, Amazon S3 disables access by "Account-2." This is because statement 3 renders the entire policy public, so `RestrictPublicBuckets` applies. As a result, Amazon S3 disables cross-account access, even though the policy delegates access to a specific account, "Account-2." But if you remove statement 3 from the policy, then the policy doesn't qualify as public, and `RestrictPublicBuckets` no longer applies. Thus, "Account-2" regains access to the bucket, even if you leave `RestrictPublicBuckets` enabled.

Permissions

To use Amazon S3 block public access features, you must have the following permissions.

Operation	Required Permissions
GET bucket policy status	<code>s3:GetBucketPolicyStatus</code>
GET bucket Block Public Access settings	<code>s3:GetBucketPublicAccessBlock</code>

Operation	Required Permissions
PUT bucket Block Public Access settings	s3:PutBucketPublicAccessBlock
DELETE bucket Block Public Access settings	s3:PutBucketPublicAccessBlock
GET account Block Public Access settings	s3:GetAccountPublicAccessBlock
PUT account Block Public Access settings	s3:PutAccountPublicAccessBlock
DELETE account Block Public Access settings	s3:PutAccountPublicAccessBlock

Note

The DELETE operations require the same permissions as the PUT operations. There are no separate permissions for the DELETE operations.

Examples

Using Block Public Access with the AWS CLI

You can use Amazon S3 block public access through the AWS CLI. The command you use depends on whether you want to perform a block public access call on a bucket or on an account. For more information about setting up and using the AWS CLI, see [What is the AWS Command Line Interface?](#)

Bucket

To perform block public access operations on a bucket, use the AWS CLI service `s3api`. The bucket-level operations that use this service are as follows:

- PUT PublicAccessBlock (for a bucket)
- GET PublicAccessBlock (for a bucket)
- DELETE PublicAccessBlock (for a bucket)
- GET BucketPolicyStatus

Account

To perform block public access operations on an account, use the AWS CLI service `s3control`. The account-level operations that use this service are as follows:

- PUT PublicAccessBlock (for an account)
- GET PublicAccessBlock (for an account)
- DELETE PublicAccessBlock (for an account)

Using Block Public Access with the AWS SDK for Java

The following examples show how to use Amazon S3 block public access with the AWS SDK for Java. For instructions on how to create and test a working sample, see [Using the AWS SDK for Java \(p. 676\)](#).

Example 1

This example shows how to set a public access block configuration on an S3 bucket using the AWS SDK for Java.

```
AmazonS3 client = AmazonS3ClientBuilder.standard()
    .withCredentials(<credentials>)
    .build();

client.setPublicAccessBlock(new SetPublicAccessBlockRequest())
```

```
.withBucketName(<bucket-name>)  
.withPublicAccessBlockConfiguration(new PublicAccessBlockConfiguration()  
    .withBlockPublicAcls(<value>)  
    .withIgnorePublicAcls(<value>)  
    .withBlockPublicPolicy(<value>)  
    .withRestrictPublicBuckets(<value>));
```

Important

This example pertains only to bucket-level operations, which use the `AmazonS3` client class. For account-level operations, see the following example.

Example 2

This example shows how to put a public access block configuration on an Amazon S3 account using the AWS SDK for Java.

```
AWSS3ControlClientBuilder controlClientBuilder = AWSS3ControlClientBuilder.standard();  
controlClientBuilder.setRegion(<region>);  
controlClientBuilder.setCredentials(<credentials>);  
  
AWSS3Control client = controlClientBuilder.build();  
client.putPublicAccessBlock(new PutPublicAccessBlockRequest()  
    .withAccountId(<account-id>)  
    .withPublicAccessBlockConfiguration(new PublicAccessBlockConfiguration()  
        .withIgnorePublicAcls(<value>)  
        .withBlockPublicAcls(<value>)  
        .withBlockPublicPolicy(<value>)  
        .withRestrictPublicBuckets(<value>)));
```

Important

This example pertains only to account-level operations, which use the `AWSS3Control` client class. For bucket-level operations, see the preceding example.

Using Block Public Access with Other AWS SDKs

For information about using the other AWS SDKs, see [Using the AWS SDKs, CLI, and Explorers \(p. 669\)](#).

Using Block Public Access with the REST APIs

For information about using Amazon S3 block public access through the REST APIs, see the following topics in the *Amazon Simple Storage Service API Reference*.

- Account-level operations
 - [PUT PublicAccessBlock](#)
 - [GET PublicAccessBlock](#)
 - [DELETE PublicAccessBlock](#)
- Bucket-level operations
 - [PUT PublicAccessBlock](#)
 - [GET PublicAccessBlock](#)
 - [DELETE PublicAccessBlock](#)
 - [GET BucketPolicyStatus](#)

Logging and Monitoring in Amazon S3

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon S3 and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon S3 resources and responding to potential incidents:

Amazon CloudWatch Alarms

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms do not invoke actions because they are in a particular state. Rather the state must have changed and been maintained for a specified number of periods. For more information, see [Monitoring Metrics with Amazon CloudWatch \(p. 611\)](#).

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon S3. Using the information collected by CloudTrail, you can determine the request that was made to Amazon S3, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see [Logging Amazon S3 API Calls by Using AWS CloudTrail \(p. 621\)](#).

Amazon S3 Access Logs

Server access logs provide detailed records about requests that are made to a bucket. Server access logs are useful for many applications. For example, access log information can be useful in security and access audits. For more information, see [Amazon S3 Server Access Logging \(p. 647\)](#).

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

Trusted Advisor has the following Amazon S3-related checks:

- Logging configuration of Amazon S3 buckets.
- Security checks for Amazon S3 buckets that have open access permissions.
- Fault tolerance checks for Amazon S3 buckets that don't have versioning enabled, or have versioning suspended.

For more information, see [AWS Trusted Advisor](#) in the *AWS Support User Guide*.

The following security best practices also address logging and monitoring:

- [Identify and audit all your Amazon S3 buckets](#)
- [Implement monitoring using AWS monitoring tools](#)
- [Enable AWS Config](#)
- [Enable Amazon S3 server access logging](#)
- [Use AWS CloudTrail](#)
- [Monitor AWS security advisories](#)

Compliance Validation for Amazon S3

The security and compliance of Amazon S3 is assessed by third-party auditors as part of multiple AWS compliance programs, including the following:

- System and Organization Controls (SOC)
- Payment Card Industry Data Security Standard (PCI DSS)
- Federal Risk and Authorization Management Program (FedRAMP)
- Health Insurance Portability and Accountability Act (HIPAA)

AWS provides a frequently updated list of AWS services in scope of specific compliance programs at [AWS Services in Scope by Compliance Program](#).

Third-party audit reports are available for you to download using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

For more information about AWS compliance programs, see [AWS Compliance Programs](#).

Your compliance responsibility when using Amazon S3 is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. If your use of Amazon S3 is subject to compliance with standards like HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- [Security and Compliance Quick Start Guides](#) that discuss architectural considerations and steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance](#) whitepaper that outlines how companies use AWS to help them meet HIPAA requirements.
- [AWS Compliance Resources](#) provide several different workbooks and guides that might apply to your industry and location.
- [AWS Config](#) can be used to assess how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) provides you with a comprehensive view of your security state within AWS and helps you check your compliance with security industry standards and best practices.
- [Amazon S3 Object Lock](#) can help you meet technical requirements of financial services regulators (such as the SEC, FINRA, and CFTC) that require write once, read many (WORM) data storage for certain types of books and records information.
- [Amazon S3 Inventory \(p. 422\)](#) can help you audit and report on the replication and encryption status of your objects for business, compliance, and regulatory needs.

Amazon S3 Inventory

Amazon S3 inventory is one of the tools Amazon S3 provides to help manage your storage. You can use it to audit and report on the replication and encryption status of your objects for business, compliance, and regulatory needs. You can also simplify and speed up business workflows and big data jobs using Amazon S3 inventory, which provides a scheduled alternative to the Amazon S3 synchronous `List` API operation.

Amazon S3 inventory provides comma-separated values (CSV), [Apache optimized row columnar \(ORC\)](#) or [Apache Parquet \(Parquet\)](#) output files that list your objects and their corresponding metadata on a daily or weekly basis for an S3 bucket or a shared prefix (that is, objects that have names that begin with a common string). For information about Amazon S3 inventory pricing, see [Amazon S3 Pricing](#).

You can configure multiple inventory lists for a bucket. You can configure what object metadata to include in the inventory, whether to list all object versions or only current versions, where to store the

inventory list file output, and whether to generate the inventory on a daily or weekly basis. You can also specify that the inventory list file be encrypted.

You can query Amazon S3 inventory using standard SQL by using [Amazon Athena](#), Amazon Redshift Spectrum, and other tools such as [Presto](#), [Apache Hive](#), and [Apache Spark](#). It's easy to use Athena to run queries on your inventory files. You can use Athena for Amazon S3 inventory queries in all Regions where Athena is available.

Topics

- [How Do I Set Up Amazon S3 Inventory?](#) (p. 423)
- [What's Included in an Amazon S3 Inventory?](#) (p. 425)
- [Where Are Inventory Lists Located?](#) (p. 426)
- [How Do I Know When an Inventory Is Complete?](#) (p. 429)
- [Querying Inventory with Amazon Athena](#) (p. 429)
- [Amazon S3 Inventory REST APIs](#) (p. 430)

How Do I Set Up Amazon S3 Inventory?

This section describes how to set up an inventory, including details about the inventory source and destination buckets.

Amazon S3 Inventory Source and Destination Buckets

The bucket that the inventory lists the objects for is called the *source bucket*. The bucket where the inventory list file is stored is called the *destination bucket*.

Source Bucket

The inventory lists the objects that are stored in the source bucket. You can get inventory lists for an entire bucket or filtered by (object key name) prefix.

The source bucket:

- Contains the objects that are listed in the inventory.
- Contains the configuration for the inventory.

Destination Bucket

Amazon S3 inventory list files are written to the destination bucket. To group all the inventory list files in a common location in the destination bucket, you can specify a destination (object key name) prefix in the inventory configuration.

The destination bucket:

- Contains the inventory file lists.
- Contains the manifest files that list all the file inventory lists that are stored in the destination bucket. For more information, see [What Is an Inventory Manifest?](#) (p. 427)
- Must have a bucket policy to give Amazon S3 permission to verify ownership of the bucket and permission to write files to the bucket.
- Must be in the same AWS Region as the source bucket.
- Can be the same as the source bucket.
- Can be owned by a different AWS account than the account that owns the source bucket.

Setting Up Amazon S3 Inventory

Amazon S3 inventory helps you manage your storage by creating lists of the objects in an S3 bucket on a defined schedule. You can configure multiple inventory lists for a bucket. The inventory lists are published to CSV, ORC, or Parquet files in a destination bucket.

The easiest way to set up an inventory is by using the AWS Management Console, but you can also use the REST API, AWS CLI, or AWS SDKs. The console performs the first step of the following procedure for you: adding a bucket policy to the destination bucket.

To set up Amazon S3 inventory for an S3 bucket

1. Add a bucket policy for the destination bucket.

You must create a bucket policy on the destination bucket to grant permissions to Amazon S3 to write objects to the bucket in the defined location. For an example policy, see [Granting Permissions for Amazon S3 Inventory and Amazon S3 Analytics \(p. 377\)](#).

2. Configure an inventory to list the objects in a source bucket and publish the list to a destination bucket.

When you configure an inventory list for a source bucket, you specify the destination bucket where you want the list to be stored, and whether you want to generate the list daily or weekly. You can also configure what object metadata to include and whether to list all object versions or only current versions.

You can specify that the inventory list file be encrypted by using Amazon S3-managed keys (SSE-S3) or keys stored in AWS KMS (SSE-KMS). For more information about SSE-S3 and SSE-KMS, see [Protecting Data Using Server-Side Encryption \(p. 265\)](#). If you plan to use SSE-KMS encryption, see Step 3.

- For information about how to use the console to configure an inventory list, see [How Do I Configure Amazon S3 Inventory?](#) in the *Amazon Simple Storage Service Console User Guide*.
- To use the Amazon S3 API to configure an inventory list, use the [PUT Bucket inventory configuration](#) REST API, or the equivalent from the AWS CLI or AWS SDKs.

3. To encrypt the inventory list file with SSE-KMS, grant Amazon S3 permission to use the AWS KMS key.

You can configure encryption for the inventory list file by using the AWS Management Console, REST API, AWS CLI, or AWS SDKs. Whichever way you choose, you must grant Amazon S3 permission to use the AWS KMS customer master key (CMK) to encrypt the inventory file. You grant Amazon S3 permission by modifying the key policy for the AWS KMS CMK that is being used to encrypt the inventory file. For more information, see the next section, [Grant Amazon S3 Permission to Encrypt Using Your AWS KMS Key \(p. 424\)](#).

Grant Amazon S3 Permission to Encrypt Using Your AWS KMS Key

You must grant Amazon S3 permission to encrypt using your AWS KMS key with a key policy. The following procedure describes how to use the AWS Identity and Access Management (IAM) console to modify the key policy for the AWS KMS CMK that is used to encrypt the inventory file.

To grant permissions to encrypt using your AWS KMS key

1. Sign in to the AWS Management Console using the AWS account that owns the AWS KMS CMK, and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Encryption keys**.
3. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (upper-right corner).

4. Choose the alias of the CMK that you want to encrypt inventory with.
5. In the **Key Policy** section of the page, choose **Switch to policy view**.
6. Using the **Key Policy** editor, insert following key policy into the existing policy and then choose **Save Changes**. You might want to copy the policy to the end of the existing policy.

```
{
  "Sid": "Allow Amazon S3 use of the key",
  "Effect": "Allow",
  "Principal": {
    "Service": "s3.amazonaws.com"
  },
  "Action": [
    "kms:GenerateDataKey*"
  ],
  "Resource": "*"
}
```

You can also use the AWS KMS PUT key policy API [PutKeyPolicy](#) to copy the key policy to the CMK that is being used to encrypt the inventory file. For more information about creating and editing AWS KMS CMKs, see [Getting Started](#) in the *AWS Key Management Service Developer Guide*.

What's Included in an Amazon S3 Inventory?

An inventory list file contains a list of the objects in the source bucket and metadata for each object. The inventory lists are stored in the destination bucket as a CSV file compressed with GZIP, as an Apache optimized row columnar (ORC) file compressed with ZLIB, or as an Apache Parquet (Parquet) file compressed with Snappy.

The inventory list contains a list of the objects in an S3 bucket and the following metadata for each listed object:

- **Bucket name** – The name of the bucket that the inventory is for.
- **Key name** – Object key name (or key) that uniquely identifies the object in the bucket. When using the CSV file format, the key name is URL-encoded and must be decoded before you can use it.
- **Version ID** – Object version ID. When you enable versioning on a bucket, Amazon S3 assigns a version number to objects that are added to the bucket. For more information, see [Object Versioning \(p. 108\)](#). (This field is not included if the list is only for the current version of objects.)
- **IsLatest** – Set to `True` if the object is the current version of the object. (This field is not included if the list is only for the current version of objects.)
- **Size** – Object size in bytes.
- **Last modified date** – Object creation date or the last modified date, whichever is the latest.
- **ETag** – The entity tag is a hash of the object. The ETag reflects changes only to the contents of an object, not its metadata. The ETag may or may not be an MD5 digest of the object data. Whether it is depends on how the object was created and how it is encrypted.
- **Storage class** – Storage class used for storing the object. For more information, see [Amazon S3 Storage Classes \(p. 103\)](#).
- **Multipart upload flag** – Set to `True` if the object was uploaded as a multipart upload. For more information, see [Multipart Upload Overview \(p. 175\)](#).
- **Delete marker** – Set to `True`, if the object is a delete marker. For more information, see [Object Versioning \(p. 108\)](#). (This field is automatically added to your report if you've configured the report to include all versions of your objects).
- **Replication status** – Set to `PENDING`, `COMPLETED`, `FAILED`, or `REPLICA`. For more information, see [Replication Status Information \(p. 594\)](#).

- **Encryption status** – Set to SSE-S3, SSE-C, SSE-KMS, or NOT-SSE. The server-side encryption status for SSE-S3, SSE-KMS, and SSE with customer-provided keys (SSE-C). A status of NOT-SSE means that the object is not encrypted with server-side encryption. For more information, see [Protecting Data Using Encryption](#) (p. 264).
- **Object lock Retain until date** – The date until which the locked object cannot be deleted. For more information, see [Locking Objects Using Amazon S3 Object Lock](#) (p. 453).
- **Object lock Mode** – Set to Governance or Compliance for objects that are locked. For more information, see [Locking Objects Using Amazon S3 Object Lock](#) (p. 453).
- **Object lock Legal hold status** – Set to On if a legal hold has been applied to an object; otherwise it is set to Off. For more information, see [Locking Objects Using Amazon S3 Object Lock](#) (p. 453).

The following is an example CSV inventory list opened in a spreadsheet application. The heading row is shown only to help clarify the example; it is not included in the actual list.

Bucket	Key	VersionId	IsLatest	IsDeleteMarker	Size	LastModifiedDate	Etag	StorageClass	MultipartUploaded	ReplicationStatus
example-bucket	object1			FALSE	2.4E+08	2016-08-11T01:19	e80d8eda4	STANDARD	TRUE	
example-bucket	object2			FALSE		0 2016-08-10T22:23	d41d8cd98	STANDARD	FALSE	
example-bucket	object3			FALSE	9	2016-08-10T20:18	9090441e4	STANDARD_IA	FALSE	
example-bucket	object4			FALSE	9	2016-08-10T20:36	9090441e4	STANDARD_IA	FALSE	
example-bucket	object1			FALSE	22	2016-08-10T20:35	9090441e4	STANDARD	FALSE	
example-bucket	object1			FALSE		2016-08-10T20:34	9090441e4	REDUCED_RED	FALSE	
example-bucket	object1			FALSE		2016-08-10T21:13	9090441e4	GLACIER	FALSE	

We recommend that you create a lifecycle policy that deletes old inventory lists. For more information, see [Object Lifecycle Management](#) (p. 119).

Inventory Consistency

All of your objects might not appear in each inventory list. The inventory list provides eventual consistency for PUTs of both new objects and overwrites, and DELETES. Inventory lists are a rolling snapshot of bucket items, which are eventually consistent (that is, the list might not include recently added or deleted objects).

To validate the state of the object before you take action on the object, we recommend that you perform a HEAD Object REST API request to retrieve metadata for the object, or check the object's properties in the Amazon S3 console. You can also check object metadata with the AWS CLI or the AWS SDKs. For more information, see [HEAD Object](#) in the *Amazon Simple Storage Service API Reference*.

Where Are Inventory Lists Located?

When an inventory list is published, the manifest files are published to the following location in the destination bucket.

```
destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/manifest.json
destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/manifest.checksum
destination-prefix/source-bucket/config-ID/hive/dt=YYYY-MM-DD-HH-MM/symlink.txt
```

- *destination-prefix* is the (object key name) prefix set in the inventory configuration, which can be used to group all the inventory list files in a common location within the destination bucket.
- *source-bucket* is the source bucket that the inventory list is for. It is added to prevent collisions when multiple inventory reports from different source buckets are sent to the same destination bucket.
- *config-ID* is added to prevent collisions with multiple inventory reports from the same source bucket that are sent to the same destination bucket. The *config-ID* comes from the inventory report configuration, and is the name for the report that is defined on setup.

- `YYYY-MM-DDTHH-MMZ` is the timestamp that consists of the start time and the date when the inventory report generation begins scanning the bucket; for example, `2016-11-06T21-32Z`. Storage added after the timestamp is not in the report.
- `manifest.json` is the manifest file.
- `manifest.checksum` is the MD5 of the content of the `manifest.json` file.
- `symlink.txt` is the Apache Hive-compatible manifest file.

The inventory lists are published daily or weekly to the following location in the destination bucket.

```
destination-prefix/source-bucket/config-ID/example-file-name.csv.gz
...
destination-prefix/source-bucket/config-ID/example-file-name-1.csv.gz
```

- `destination-prefix` is the (object key name) prefix set in the inventory configuration. It can be used to group all the inventory list files in a common location in the destination bucket.
- `source-bucket` is the source bucket that the inventory list is for. It is added to prevent collisions when multiple inventory reports from different source buckets are sent to the same destination bucket.
- `example-file-name.csv.gz` is one of the CSV inventory files. ORC inventory names end with the file name extension `.orc`, and Parquet inventory names end with the file name extension `.parquet`.

What Is an Inventory Manifest?

The manifest files `manifest.json` and `symlink.txt` describe where the inventory files are located. Whenever a new inventory list is delivered, it is accompanied by a new set of manifest files.

Each manifest contained in the `manifest.json` file provides metadata and other basic information about an inventory. This information includes the following:

- Source bucket name
- Destination bucket name
- Version of the inventory
- Creation timestamp in the epoch date format that consists of the start time and the date when the inventory report generation begins scanning the bucket
- Format and schema of the inventory files
- Actual list of the inventory files that are in the destination bucket

Whenever a `manifest.json` file is written, it is accompanied by a `manifest.checksum` file that is the MD5 of the content of `manifest.json` file.

The following is an example of a manifest in a `manifest.json` file for a CSV-formatted inventory.

```
{
  "sourceBucket": "example-source-bucket",
  "destinationBucket": "arn:aws:s3:::example-inventory-destination-bucket",
  "version": "2016-11-30",
  "creationTimestamp" : "1514944800000",
  "fileFormat": "CSV",
  "fileSchema": "Bucket, Key, VersionId, IsLatest, IsDeleteMarker, Size,
LastModifiedDate, ETag, StorageClass, IsMultipartUploaded, ReplicationStatus,
EncryptionStatus, ObjectLockRetainUntilDate, ObjectLockMode, ObjectLockLegalHoldStatus",
```

```

    "files": [
      {
        "key": "Inventory/example-source-bucket/2016-11-06T21-32Z/
files/939c6d46-85a9-4ba8-87bd-9db705a579ce.csv.gz",
        "size": 2147483647,
        "MD5checksum": "f11166069f1990abeb9c97ace9cdfabc"
      }
    ]
  }
}

```

The following is an example of a manifest in a `manifest.json` file for an ORC-formatted inventory.

```

{
  "sourceBucket": "example-source-bucket",
  "destinationBucket": "arn:aws:s3:::example-destination-bucket",
  "version": "2016-11-30",
  "creationTimestamp": "1514944800000",
  "fileFormat": "ORC",
  "fileSchema":
    "struct<bucket:string,key:string,version_id:string,is_latest:boolean,is_delete_marker:boolean,size:big
    "files": [
      {
        "key": "inventory/example-source-bucket/data/
d794c570-95bb-4271-9128-26023c8b4900.orc",
        "size": 56291,
        "MD5checksum": "5925f4e78e1695c2d020b9f6eexample"
      }
    ]
  }
}

```

The following is an example of a manifest in a `manifest.json` file for a Parquet-formatted inventory.

```

{
  "sourceBucket": "example-source-bucket",
  "destinationBucket": "arn:aws:s3:::example-destination-bucket",
  "version": "2016-11-30",
  "creationTimestamp": "1514944800000",
  "fileFormat": "Parquet",
  "fileSchema": "message s3.inventory { required binary bucket (UTF8); required binary
key (UTF8); optional binary version_id (UTF8); optional boolean is_latest; optional
boolean is_delete_marker; optional int64 size; optional int64 last_modified_date
(TIMESTAMP_MILLIS); optional binary e_tag (UTF8); optional binary storage_class (UTF8);
optional boolean is_multipart_uploaded; optional binary replication_status (UTF8);
optional binary encryption_status (UTF8);}"
  "files": [
    {
      "key": "inventory/example-source-bucket/data/
d754c470-85bb-4255-9218-47023c8b4910.parquet",
      "size": 56291,
      "MD5checksum": "5825f2e18e1695c2d030b9f6eexample"
    }
  ]
}

```

The `symlink.txt` file is an Apache Hive-compatible manifest file that allows Hive to automatically discover inventory files and their associated data files. The Hive-compatible manifest works with the Hive-compatible services Athena and Amazon Redshift Spectrum. It also works with Hive-compatible applications, including [Presto](#), [Apache Hive](#), [Apache Spark](#), and many others.

Important

The `symlink.txt` Apache Hive-compatible manifest file does not currently work with AWS Glue.

Reading `symlink.txt` with [Apache Hive](#) and [Apache Spark](#) is not supported for ORC and Parquet-formatted inventory files.

How Do I Know When an Inventory Is Complete?

You can set up an Amazon S3 event notification to receive notice when the manifest checksum file is created, which indicates that an inventory list has been added to the destination bucket. The manifest is an up-to-date list of all the inventory lists at the destination location.

Amazon S3 can publish events to an Amazon Simple Notification Service (Amazon SNS) topic, an Amazon Simple Queue Service (Amazon SQS) queue, or an AWS Lambda function. For more information, see [Configuring Amazon S3 Event Notifications \(p. 530\)](#).

The following notification configuration defines that all `manifest.checksum` files newly added to the destination bucket are processed by the AWS Lambda `cloud-function-list-write`.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>destination-prefix/source-bucket</Value>
        </FilterRule>
        <FilterRule>
          <Name>suffix</Name>
          <Value>checksum</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Cloudcode>arn:aws:lambda:us-west-2:222233334444:cloud-function-list-write</Cloudcode>
    <Event>s3:ObjectCreated:*</Event>
  </QueueConfiguration>
</NotificationConfiguration>
```

For more information, see [Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

Querying Inventory with Amazon Athena

You can query Amazon S3 inventory using standard SQL by using Amazon Athena in all Regions where Athena is available. To check for AWS Region availability, see the [AWS Region Table](#).

Athena can query Amazon S3 inventory files in ORC, Parquet, or CSV format. When you use Athena to query inventory, we recommend that you use ORC-formatted or Parquet-formatted inventory files. ORC and Parquet formats provide faster query performance and lower query costs. ORC and Parquet are self-describing type-aware columnar file formats designed for [Apache Hadoop](#). The columnar format lets the reader read, decompress, and process only the columns that are required for the current query. The ORC and Parquet formats for Amazon S3 inventory are available in all AWS Regions.

To get started using Athena to query Amazon S3 inventory

1. Create an Athena table. For information about creating a table, see [Creating Tables in Amazon Athena](#) in the *Amazon Athena User Guide*.

The following sample query includes all optional fields in an ORC-formatted inventory report. Drop any optional field that you did not choose for your inventory so that the query corresponds to the fields chosen for your inventory. Also, you must use your bucket name and the location. The location points to your inventory destination path; for example, `s3://destination-prefix/source-bucket/config-ID/hive/`.

```
CREATE EXTERNAL TABLE your_table_name(
  `bucket` string,
  key string,
  version_id string,
  is_latest boolean,
  is_delete_marker boolean,
  size bigint,
  last_modified_date timestamp,
  e_tag string,
  storage_class string,
  is_multipart_uploaded boolean,
  replication_status string,
  encryption_status string,
  object_lock_retain_until_date timestamp,
  object_lock_mode string,
  object_lock_legal_hold_status string
)
PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
LOCATION 's3://destination-prefix/source-bucket/config-ID/hive/';
```

When using Athena to query a Parquet-formatted inventory report, use the following Parquet SerDe in place of the ORC SerDe in the ROW FORMAT SERDE statement.

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
```

2. To add new inventory lists to your table, use the following MSCK REPAIR TABLE command.

```
MSCK REPAIR TABLE your-table-name;
```

3. After performing the first two steps, you can run ad hoc queries on your inventory, as shown in the following example.

```
SELECT encryption_status, count(*) FROM your-table-name GROUP BY encryption_status;
```

For more information about using Athena, see [Amazon Athena User Guide](#).

Amazon S3 Inventory REST APIs

The following are the REST operations used for Amazon S3 inventory.

- [DELETE Bucket Inventory](#)
- [GET Bucket Inventory](#)
- [List Bucket Inventory](#)
- [PUT Bucket Inventory](#)

Resilience in Amazon S3

The AWS global infrastructure is built around Regions and Availability Zones. AWS Regions provide multiple, physically separated and isolated Availability Zones that are connected with low latency, high throughput, and highly redundant networking. These Availability Zones offer you an effective way to design and operate applications and databases. They are more highly available, fault tolerant, and scalable than traditional single data center infrastructures or multi-data center infrastructures. If you specifically need to replicate your data over greater geographic distances, you can use [Replication \(p. 551\)](#), which enables automatic, asynchronous copying of objects across buckets in different AWS Regions.

Each AWS Region has multiple Availability Zones. You can deploy your applications across multiple Availability Zones in the same Region for fault tolerance and low latency. Availability Zones are connected to each other with fast, private fiber-optic networking, enabling you to easily architect applications that automatically fail over between Availability Zones without interruption.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

In addition to the AWS global infrastructure, Amazon S3 offers several features to help support your data resiliency and backup needs.

Lifecycle configuration

A lifecycle configuration is a set of rules that define actions that Amazon S3 applies to a group of objects. With lifecycle configuration rules, you can tell Amazon S3 to transition objects to less expensive storage classes, archive them, or delete them. For more information, see [Object Lifecycle Management \(p. 119\)](#).

Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. For more information, see [Using Versioning \(p. 432\)](#).

Amazon S3 object lock

You can use Amazon S3 object lock to store objects using a *write once, read many* (WORM) model. Using Amazon S3 object lock, you can prevent an object from being deleted or overwritten for a fixed amount of time or indefinitely. Amazon S3 object lock enables you to meet regulatory requirements that require WORM storage or simply to add an additional layer of protection against object changes and deletion. For more information, see [Locking Objects Using Amazon S3 Object Lock \(p. 453\)](#).

Storage classes

Amazon S3 offers a range of storage classes for the objects that you store. Two of these storage classes (STANDARD_IA and ONEZONE_IA) are designed for long-lived and infrequently accessed data, such as backups. You can also use the GLACIER storage class to archive objects that you don't need to access in real time. For more information, see [Amazon S3 Storage Classes \(p. 103\)](#).

The following security best practices also address resilience:

- [Enable versioning](#)
- [Consider Amazon S3 cross-region replication](#)
- [Identify and audit all your Amazon S3 buckets](#)

Encryption of Amazon S3 Backups

If you are storing backups using Amazon S3, the encryption of your backups depends on the configuration of those buckets. Amazon S3 provides a way to set the default encryption behavior for an S3 bucket. You can set default encryption on a bucket so that all objects are encrypted when they are stored in the bucket. The default encryption supports keys stored in AWS KMS (SSE-KMS). For more information, see [Amazon S3 Default Encryption for S3 Buckets \(p. 66\)](#).

Using Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures.

In one bucket, for example, you can have two objects with the same key, but different version IDs, such as `photo.gif` (version 111111) and `photo.gif` (version 121212).



Versioning-enabled buckets enable you to recover objects from accidental deletion or overwrite. For example:

- If you delete an object, instead of removing it permanently, Amazon S3 inserts a delete marker, which becomes the current object version. You can always restore the previous version. For more information, see [Deleting Object Versions \(p. 444\)](#).
- If you overwrite an object, it results in a new object version in the bucket. You can always restore the previous version.

Important

If you have an object expiration lifecycle policy in your non-versioned bucket and you want to maintain the same permanent delete behavior when you enable versioning, you must add a noncurrent expiration policy. The noncurrent expiration lifecycle policy will manage the deletes of the noncurrent object versions in the version-enabled bucket. (A version-enabled bucket maintains one current and zero or more noncurrent object versions.) For more information, see [How Do I Create a Lifecycle Policy for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Buckets can be in one of three states: unversioned (the default), versioning-enabled, or versioning-suspended.

Important

Once you version-enable a bucket, it can never return to an unversioned state. You can, however, suspend versioning on that bucket.

The versioning state applies to all (never some) of the objects in that bucket. The first time you enable a bucket for versioning, objects in it are thereafter always versioned and given a unique version ID. Note the following:

- Objects stored in your bucket before you set the versioning state have a version ID of `null`. When you enable versioning, existing objects in your bucket do not change. What changes is how Amazon S3

handles the objects in future requests. For more information, see [Managing Objects in a Versioning-Enabled Bucket \(p. 437\)](#).

- The bucket owner (or any user with appropriate permissions) can suspend versioning to stop accruing object versions. When you suspend versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles objects in future requests. For more information, see [Managing Objects in a Versioning-Suspended Bucket \(p. 451\)](#).

How to Configure Versioning on a Bucket

You can configure bucket versioning using any of the following methods:

- Configure versioning using the Amazon S3 console.
- Configure versioning programmatically using the AWS SDKs.

Both the console and the SDKs call the REST API that Amazon S3 provides to manage versioning.

Note

If you need to, you can also make the Amazon S3 REST API calls directly from your code. However, this can be cumbersome because it requires you to write code to authenticate your requests.

Each bucket you create has a *versioning* subresource (see [Bucket Configuration Options \(p. 56\)](#)) associated with it. By default, your bucket is unversioned, and accordingly the versioning subresource stores empty versioning configuration.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
</VersioningConfiguration>
```

To enable versioning, you send a request to Amazon S3 with a versioning configuration that includes a status.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

To suspend versioning, you set the status value to *Suspended*.

The bucket owner, an AWS account that created the bucket (root account), and authorized users can configure the versioning state of a bucket. For more information about permissions, see [Identity and Access Management in Amazon S3 \(p. 301\)](#).

For an example of configuring versioning, see [Examples of Enabling Bucket Versioning \(p. 434\)](#).

MFA Delete

You can optionally add another layer of security by configuring a bucket to enable MFA (multi-factor authentication) Delete, which requires additional authentication for either of the following operations:

- Change the versioning state of your bucket
- Permanently delete an object version

MFA Delete requires two forms of authentication together:

- Your security credentials

- The concatenation of a valid serial number, a space, and the six-digit code displayed on an approved authentication device

MFA Delete thus provides added security in the event, for example, your security credentials are compromised.

To enable or disable MFA Delete, you use the same API that you use to configure versioning on a bucket. Amazon S3 stores the MFA Delete configuration in the same *versioning* subresource that stores the bucket's versioning status.

MFA Delete can help prevent accidental bucket deletions by doing the following:

- Requiring the user who initiates the delete action to prove physical possession of an MFA device with an MFA code.
- Adding an extra layer of friction and security to the delete action.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>VersioningState</Status>
  <MfaDelete>MfaDeleteState</MfaDelete>
</VersioningConfiguration>
```

Note

The bucket owner, the AWS account that created the bucket (root account), and all authorized IAM users can enable versioning, but only the bucket owner (root account) can enable MFA Delete. For more information, see the AWS blog post on [MFA Delete and Versioning](#).

To use MFA Delete, you can use either a hardware or virtual MFA device to generate an authentication code. The following example shows a generated authentication code displayed on a hardware device.



Note

MFA Delete and MFA-protected API access are features intended to provide protection for different scenarios. You configure MFA Delete on a bucket to ensure that data in your bucket cannot be accidentally deleted. MFA-protected API access is used to enforce another authentication factor (MFA code) when accessing sensitive Amazon S3 resources. You can require any operations against these Amazon S3 resources be done with temporary credentials created using MFA. For an example, see [Adding a Bucket Policy to Require MFA \(p. 375\)](#). For more information on how to purchase and activate an authentication device, see <https://aws.amazon.com/iam/details/mfa/>.

Related Topics

For more information, see the following topics:

- [Examples of Enabling Bucket Versioning \(p. 434\)](#)
- [Managing Objects in a Versioning-Enabled Bucket \(p. 437\)](#)
- [Managing Objects in a Versioning-Suspended Bucket \(p. 451\)](#)
- [Significant Increases in HTTP 503 Responses to Amazon S3 Requests to Buckets with Versioning Enabled \(p. 643\)](#)

Examples of Enabling Bucket Versioning

Topics

- [Using the Amazon S3 Console \(p. 435\)](#)
- [Using the AWS SDK for Java \(p. 435\)](#)
- [Using the AWS SDK for .NET \(p. 436\)](#)
- [Using Other AWS SDKs \(p. 437\)](#)

This section provides examples of enabling versioning on a bucket. The examples first enable versioning on a bucket and then retrieve versioning status. For an introduction, see [Using Versioning \(p. 432\)](#).

Using the Amazon S3 Console

For more information about enabling versioning on a bucket using the Amazon S3 console, see [How Do I Enable or Suspend Versioning for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Using the AWS SDK for Java

Example

For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

```
import java.io.IOException;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class BucketVersioningConfigurationExample {
    public static String bucketName = "*** bucket name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
        s3Client.setRegion(Region.getRegion(Regions.US_EAST_1));
        try {

            // 1. Enable versioning on the bucket.
            BucketVersioningConfiguration configuration =
                new BucketVersioningConfiguration().withStatus("Enabled");

            SetBucketVersioningConfigurationRequest setBucketVersioningConfigurationRequest =
                new SetBucketVersioningConfigurationRequest(bucketName, configuration);

            s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigurationRequest);

            // 2. Get bucket versioning configuration information.
            BucketVersioningConfiguration conf =
                s3Client.getBucketVersioningConfiguration(bucketName);
            System.out.println("bucket versioning configuration status:    " + conf.getStatus());

        } catch (AmazonS3Exception amazonS3Exception) {
            System.out.format("An Amazon S3 error occurred. Exception: %s",
                amazonS3Exception.toString());
        } catch (Exception ex) {
            System.out.format("Exception: %s", ex.toString());
        }
    }
}
```

```
}
```

Using the AWS SDK for .NET

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Example

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class BucketVersioningConfiguration
    {
        static string bucketName = "**** bucket name ****";

        public static void Main(string[] args)
        {
            using (var client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                try
                {
                    EnableVersioningOnBucket(client);
                    string bucketVersioningStatus =
RetrieveBucketVersioningConfiguration(client);
                }
                catch (AmazonS3Exception amazonS3Exception)
                {
                    if (amazonS3Exception.ErrorCode != null &&
                        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                        ||
                        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                    {
                        Console.WriteLine("Check the provided AWS Credentials.");
                        Console.WriteLine(
                            "To sign up for service, go to http://aws.amazon.com/s3");
                    }
                    else
                    {
                        Console.WriteLine(
                            "Error occurred. Message:'{0}' when listing objects",
                            amazonS3Exception.Message);
                    }
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void EnableVersioningOnBucket(IAmazonS3 client)
        {
            PutBucketVersioningRequest request = new PutBucketVersioningRequest
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig
                {
                    Status = VersionStatus.Enabled
                }
            };
        }
    }
}
```

```

        PutBucketVersioningResponse response = client.PutBucketVersioning(request);
    }

    static string RetrieveBucketVersioningConfiguration(IAmazonS3 client)
    {
        GetBucketVersioningRequest request = new GetBucketVersioningRequest
        {
            BucketName = bucketName
        };

        GetBucketVersioningResponse response = client.GetBucketVersioning(request);
        return response.VersioningConfig.Status;
    }
}

```

Using Other AWS SDKs

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Managing Objects in a Versioning-Enabled Bucket

Topics

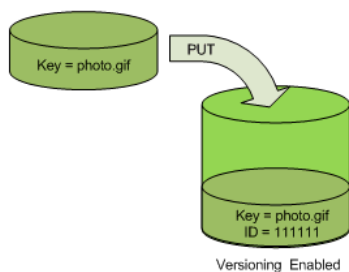
- [Adding Objects to Versioning-Enabled Buckets \(p. 437\)](#)
- [Listing Objects in a Versioning-Enabled Bucket \(p. 438\)](#)
- [Retrieving Object Versions \(p. 443\)](#)
- [Deleting Object Versions \(p. 444\)](#)
- [Transitioning Object Versions \(p. 449\)](#)
- [Restoring Previous Versions \(p. 449\)](#)
- [Versioned Object Permissions \(p. 450\)](#)

Objects that are stored in your bucket before you set the versioning state have a version ID of `null`. When you enable versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles the objects in future requests. The topics in this section explain various object operations in a versioning-enabled bucket.

Adding Objects to Versioning-Enabled Buckets

Once you enable versioning on a bucket, Amazon S3 automatically adds a unique version ID to every object stored (using `PUT`, `POST`, or `COPY`) in the bucket.

The following figure shows that Amazon S3 adds a unique version ID to an object when it is added to a versioning-enabled bucket.



Topics

- [Using the Console \(p. 438\)](#)
- [Using the AWS SDKs \(p. 438\)](#)
- [Using the REST API \(p. 438\)](#)

Using the Console

For instructions, see [How Do I Upload an Object to an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Using the AWS SDKs

For examples of uploading objects using the AWS SDKs for Java, .NET, and PHP, see [Uploading Objects \(p. 169\)](#). The examples for uploading objects in nonversioned and versioning-enabled buckets are the same, although in the case of versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Using the REST API

Adding Objects to Versioning-Enabled Buckets

1	Enable versioning on a bucket using a <code>PUT Bucket versioning</code> request. For more information, see PUT Bucket versioning .
2	Send a <code>PUT</code> , <code>POST</code> , or <code>COPY</code> request to store an object in the bucket.

When you add an object to a versioning-enabled bucket, Amazon S3 returns the version ID of the object in the `x-amz-version-id` response header, for example:

```
x-amz-version-id: 3/L4kqtJlcpXroDTdMJ+rmSpXd3dIbrHY
```

Note

Normal Amazon S3 rates apply for every version of an object stored and transferred. Each version of an object is the entire object; it is not just a diff from the previous version. Thus, if you have three versions of an object stored, you are charged for three objects.

Note

The version ID values that Amazon S3 assigns are URL safe (can be included as part of a URI).

Listing Objects in a Versioning-Enabled Bucket

Topics

- [Using the Console \(p. 438\)](#)
- [Using the AWS SDKs \(p. 439\)](#)
- [Using the REST API \(p. 441\)](#)

This section provides an example of listing object versions from a versioning-enabled bucket. Amazon S3 stores object version information in the *versions* subresource (see [Bucket Configuration Options \(p. 56\)](#)) that is associated with the bucket.

Using the Console

For information about listing object versions using the Amazon S3 console, see [How Do I See the Versions of an S3 Object?](#) in the *Amazon Simple Storage Service Console User Guide*.

Using the AWS SDKs

The examples in this section show how to retrieve an object listing from a versioning-enabled bucket. Each request returns up to 1,000 versions, unless you specify a lower number. If the bucket contains more versions than this limit, you send a series of requests to retrieve the list of all versions. This process of returning results in "pages" is called *pagination*. To show how pagination works, the examples limit each response to two object versions. After retrieving the first page of results, each example checks to determine whether the version list was truncated. If it was, the example continues retrieving pages until all versions have been retrieved.

Note

The following examples also work with a bucket that isn't versioning-enabled, or for objects that don't have individual versions. In those cases, Amazon S3 returns the object listing with a version ID of null.

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Using the AWS SDK for Java

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.VersionListing;

public class ListKeysVersioningEnabledBucket {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Retrieve the list of versions. If the bucket contains more versions
            // than the specified maximum number of results, Amazon S3 returns
            // one page of results per request.
            ListVersionsRequest request = new ListVersionsRequest()
                .withBucketName(bucketName)
                .withMaxResults(2);
            VersionListing versionListing = s3Client.listVersions(request);
            int numVersions = 0, numPages = 0;
            while (true) {
                numPages++;
                for (S3VersionSummary objectSummary :
                    versionListing.getVersionSummaries()) {
                    System.out.printf("Retrieved object %s, version %s\n",
                        objectSummary.getKey(),
                        objectSummary.getVersionId());
                    numVersions++;
                }
            }
        }
```

```

        // Check whether there are more pages of versions to retrieve. If
        // there are, retrieve them. Otherwise, exit the loop.
        if (versionListing.isTruncated()) {
            versionListing = s3Client.listNextBatchOfVersions(versionListing);
        } else {
            break;
        }
    }
    System.out.println(numVersions + " object versions retrieved in " + numPages +
" pages");
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
}
}

```

Using the AWS SDK for .NET

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Example

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ListObjectsVersioningEnabledBucketTest
    {
        static string bucketName = "**** bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main(string[] args)
        {
            s3Client = new AmazonS3Client(bucketRegion);
            GetObjectListWithAllVersionsAsync().Wait();
        }

        static async Task GetObjectListWithAllVersionsAsync()
        {
            try
            {
                ListVersionsRequest request = new ListVersionsRequest()
                {
                    BucketName = bucketName,
                    // You can optionally specify key name prefix in the request
                    // if you want list of object versions of a specific object.

                    // For this example we limit response to return list of 2 versions.
                    MaxKeys = 2
                };
            }
        }
    }
}

```

```

        do
        {
            ListVersionsResponse response = await
s3Client.ListVersionsAsync(request);
            // Process response.
            foreach (S3ObjectVersion entry in response.Versions)
            {
                Console.WriteLine("key = {0} size = {1}",
                    entry.Key, entry.Size);
            }

            // If response is truncated, set the marker to get the next
            // set of keys.
            if (response.IsTruncated)
            {
                request.KeyMarker = response.NextKeyMarker;
                request.VersionIdMarker = response.NextVersionIdMarker;
            }
            else
            {
                request = null;
            }
        } while (request != null);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}
}
}

```

Using the REST API

To list all the versions of all the objects in a bucket, you use the versions subresource in a GET Bucket request. Amazon S3 can retrieve only a maximum of 1,000 objects, and each object version counts fully as an object. Therefore, if a bucket contains two keys (for example, `photo.gif` and `picture.jpg`), and the first key has 990 versions and the second key has 400 versions, a single request would retrieve all 990 versions of `photo.gif` and only the most recent 10 versions of `picture.jpg`.

Amazon S3 returns object versions in the order in which they were stored, with the most recently stored returned first.

To list all object versions in a bucket

- In a GET Bucket request, include the versions subresource.

```

GET /?versions HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=

```

Retrieving a Subset of Objects in a Bucket

This section discusses the following two example scenarios:

- You want to retrieve a subset of all object versions in a bucket, for example, retrieve all versions of a specific object.
- The number of object versions in the response exceeds the value for `max-key` (1000 by default), so that you have to submit a second request to retrieve the remaining object versions.

To retrieve a subset of object versions, you use the request parameters for GET Bucket. For more information, see [GET Bucket](#).

Example 1: Retrieving All Versions of Only a Specific Object

You can retrieve all versions of an object using the `versions` subresource and the `prefix` request parameter using the following process. For more information about `prefix`, see [GET Bucket](#).

Retrieving All Versions of a Key

1	Set the <code>prefix</code> parameter to the key of the object you want to retrieve.
2	Send a GET Bucket request using the <code>versions</code> subresource and <code>prefix</code> . GET /?versions&prefix=objectName HTTP/1.1

Example Retrieving Objects Using a Prefix

The following example retrieves objects whose key is or begins with `myObject`.

```
GET /?versions&prefix=myObject HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

You can use the other request parameters to retrieve a subset of all versions of the object. For more information, see [GET Bucket](#).

Example 2: Retrieving a Listing of Additional Objects if the Response Is Truncated

If the number of objects that could be returned in a GET request exceeds the value of `max-keys`, the response contains `<isTruncated>true</isTruncated>`, and includes the first key (in `NextKeyMarker`) and the first version ID (in `NextVersionIdMarker`) that satisfy the request, but were not returned. You use those returned values as the starting position in a subsequent request to retrieve the additional objects that satisfy the GET request.

Use the following process to retrieve additional objects that satisfy the original GET Bucket versions request from a bucket. For more information about `key-marker`, `version-id-marker`, `NextKeyMarker`, and `NextVersionIdMarker`, see [GET Bucket](#).

Retrieving Additional Responses that Satisfy the Original GET Request

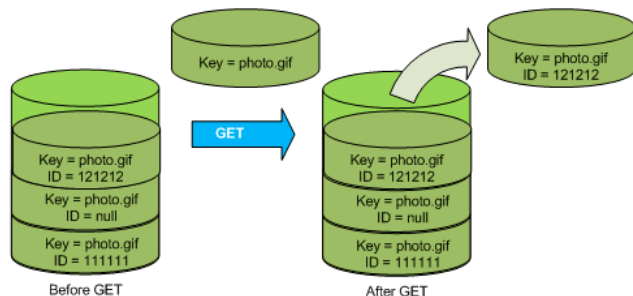
1	Set the value of <code>key-marker</code> to the key returned in <code>NextKeyMarker</code> in the previous response.
2	Set the value of <code>version-id-marker</code> to the version ID returned in <code>NextVersionIdMarker</code> in the previous response.
3	Send a GET Bucket versions request using <code>key-marker</code> and <code>version-id-marker</code> .

Example Retrieving Objects Starting with a Specified Key and Version ID

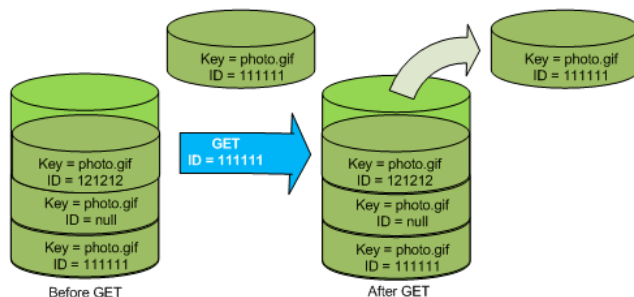
```
GET /?versions&key-marker=myObject&version-id-marker=298459348571 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Retrieving Object Versions

A simple `GET` request retrieves the current version of an object. The following figure shows how `GET` returns the current version of the object, `photo.gif`.



To retrieve a specific version, you have to specify its version ID. The following figure shows that a `GET versionId` request retrieves the specified version of the object (not necessarily the current one).



Using the Console

For instructions see, [How Do I See the Versions of an S3 Object?](#) in the Amazon Simple Storage Service Console User Guide.

Using the AWS SDKs

For examples of uploading objects using AWS SDKs for Java, .NET, and PHP, see [Getting Objects \(p. 161\)](#). The examples for uploading objects in a nonversioned and versioning-enabled buckets are the same, although in the case of versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Using REST

To retrieve a specific object version

1. Set `versionId` to the ID of the version of the object you want to retrieve.
2. Send a `GET Object versionId` request.

Example Retrieving a Versioned Object

The following request retrieves version L4kqtJlcpXroDTDmpUMLUo of `my-image.jpg`.

```
GET /my-image.jpg?versionId=L4kqtJlcpXroDTDmpUMLUo HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Related Topics

[Retrieving the Metadata of an Object Version \(p. 444\)](#)

Retrieving the Metadata of an Object Version

If you only want to retrieve the metadata of an object (and not its content), you use the `HEAD` operation. By default, you get the metadata of the most recent version. To retrieve the metadata of a specific object version, you specify its version ID.

To retrieve the metadata of an object version

1. Set `versionId` to the ID of the version of the object whose metadata you want to retrieve.
2. Send a `HEAD Object versionId` request.

Example Retrieving the Metadata of a Versioned Object

The following request retrieves the metadata of version 3HL4kqCxf3vjVBH40Nrjfk of `my-image.jpg`.

```
HEAD /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfk HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

The following shows a sample response.

```
HTTP/1.1 200 OK
x-amz-id-2: ef8yU9AS1ed4OpIszj7UDNEHGran
x-amz-request-id: 318BC8BC143432E5
x-amz-version-id: 3HL4kqtJlcpXroDTDmjVBH40Nrjfk
Date: Wed, 28 Oct 2009 22:32:00 GMT
Last-Modified: Sun, 1 Jan 2006 12:00:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 434234
Content-Type: text/plain
Connection: close
Server: AmazonS3
```

Deleting Object Versions

You can delete object versions whenever you want. In addition, you can also define lifecycle configuration rules for objects that have a well-defined lifecycle to request Amazon S3 to expire current object versions or permanently remove noncurrent object versions. When your bucket is version-enabled or versioning is suspended, the lifecycle configuration actions work as follows:

- The `Expiration` action applies to the current object version and instead of deleting the current object version, Amazon S3 retains the current version as a noncurrent version by adding a delete marker, which then becomes the current version.

- The `NoncurrentVersionExpiration` action applies to noncurrent object versions, and Amazon S3 permanently removes these object versions. You cannot recover permanently removed objects.

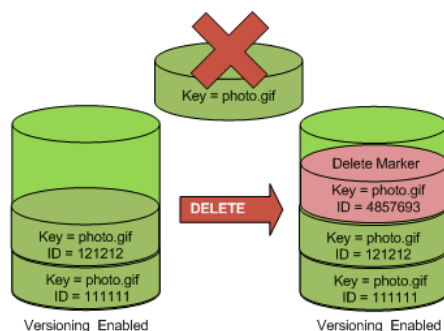
For more information, see [Object Lifecycle Management \(p. 119\)](#).

A `DELETE` request has the following use cases:

- When versioning is enabled, a simple `DELETE` cannot permanently delete an object.

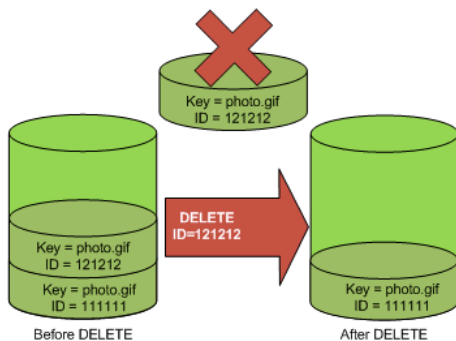
Instead, Amazon S3 inserts a delete marker in the bucket, and that marker becomes the current version of the object with a new ID. When you try to `GET` an object whose current version is a delete marker, Amazon S3 behaves as though the object has been deleted (even though it has not been erased) and returns a 404 error.

The following figure shows that a simple `DELETE` does not actually remove the specified object. Instead, Amazon S3 inserts a delete marker.



- To permanently delete versioned objects, you must use `DELETE Object versionId`.

The following figure shows that deleting a specified object version permanently removes that object.



Using the Console

For instructions see, [How Do I See the Versions of an S3 Object?](#) in the Amazon Simple Storage Service Console User Guide.

Using the AWS SDKs

For examples of deleting objects using the AWS SDKs for Java, .NET, and PHP, see [Deleting Objects \(p. 227\)](#). The examples for deleting objects in nonversioned and versioning-enabled buckets are the same, although in the case of versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Using REST

To delete a specific version of an object

- In a `DELETE`, specify a version ID.

Example Deleting a Specific Version

The following example shows how to delete version `UIORUnfnd89493jJfJ` of `photo.gif`.

```
DELETE /photo.gif?versionId=UIORUnfnd89493jJfJ HTTP/1.1

Host: bucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 0
```

Related Topics

[Using MFA Delete \(p. 446\)](#)

[Working with Delete Markers \(p. 446\)](#)

[Removing Delete Markers \(p. 448\)](#)

[Using Versioning \(p. 432\)](#)

Using MFA Delete

If a bucket's versioning configuration is MFA Delete-enabled, the bucket owner must include the `x-amz-mfa` request header in requests to permanently delete an object version or change the versioning state of the bucket. Requests that include `x-amz-mfa` must use HTTPS. The header's value is the concatenation of your authentication device's serial number, a space, and the authentication code displayed on it. If you do not include this request header, the request fails.

For more information about authentication devices, see <https://aws.amazon.com/iam/details/mfa/>.

Example Deleting an Object from an MFA Delete Enabled Bucket

The following example shows how to delete `my-image.jpg` (with the specified version), which is in a bucket configured with MFA Delete enabled. Note the space between `[SerialNumber]` and `[AuthenticationCode]`. For more information, see [DELETE Object](#).

```
DELETE /my-image.jpg?versionId=3HL4kqCxf3vjVBH40NrjfkD HTTPS/1.1
Host: bucketName.s3.amazonaws.com
x-amz-mfa: 20899872 301749
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

For more information about enabling MFA delete, see [MFA Delete \(p. 433\)](#).

Working with Delete Markers

A delete marker is a placeholder (marker) for a versioned object that was named in a simple `DELETE` request. Because the object was in a versioning-enabled bucket, the object was not deleted. The delete marker, however, makes Amazon S3 behave as if it had been deleted.

A delete marker has a key name (or key) and version ID like any other object. However, a delete marker differs from other objects in the following ways:

- It does not have data associated with it.
- It is not associated with an access control list (ACL) value.
- It does not retrieve anything from a `GET` request because it has no data; you get a 404 error.
- The only operation you can use on a delete marker is `DELETE`, and only the bucket owner can issue such a request.

Delete markers accrue a nominal charge for storage in Amazon S3. The storage size of a delete marker is equal to the size of the key name of the delete marker. A key name is a sequence of Unicode characters. The UTF-8 encoding adds from 1 to 4 bytes of storage to your bucket for each character in the name. For more information about key names, see [Object Keys \(p. 99\)](#). For information about deleting a delete marker, see [Removing Delete Markers \(p. 448\)](#).

Only Amazon S3 can create a delete marker, and it does so whenever you send a `DELETE` `Object` request on an object in a versioning-enabled or suspended bucket. The object named in the `DELETE` request is not actually deleted. Instead, the delete marker becomes the current version of the object. (The object's key name (or key) becomes the key of the delete marker.) If you try to get an object and its current version is a delete marker, Amazon S3 responds with:

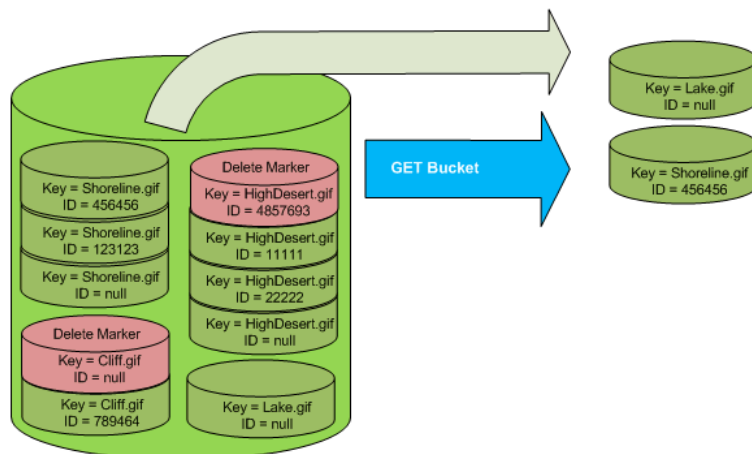
- A 404 (Object not found) error
- A response header, `x-amz-delete-marker: true`

The response header tells you that the object accessed was a delete marker. This response header never returns `false`; if the value is `false`, Amazon S3 does not include this response header in the response.

The following figure shows how a simple `GET` on an object, whose current version is a delete marker, returns a 404 No Object Found error.



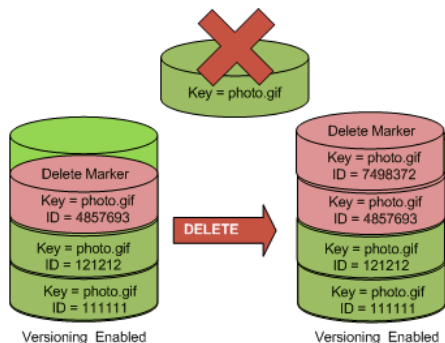
The only way to list delete markers (and other versions of an object) is by using the `versions` subresource in a `GET` `Bucket versions` request. A simple `GET` does not retrieve delete marker objects. The following figure shows that a `GET` `Bucket` request does not return objects whose current version is a delete marker.



Removing Delete Markers

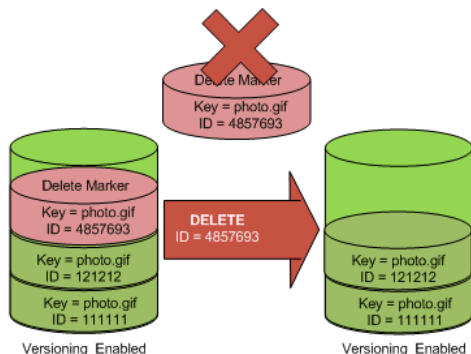
To delete a delete marker, you must specify its version ID in a `DELETE Object versionId` request. If you use a `DELETE` request to delete a delete marker (without specifying the version ID of the delete marker), Amazon S3 does not delete the delete marker, but instead, inserts another delete marker.

The following figure shows how a simple `DELETE` on a delete marker removes nothing, but adds a new delete marker to a bucket.



In a versioning-enabled bucket, this new delete marker would have a unique version ID. So, it's possible to have multiple delete markers of the same object in one bucket.

To permanently delete a delete marker, you must include its version ID in a `DELETE Object versionId` request. The following figure shows how a `DELETE Object versionId` request permanently removes a delete marker. Only the owner of a bucket can permanently remove a delete marker.



The effect of removing the delete marker is that a simple GET request will now retrieve the current version (121212) of the object.

To permanently remove a delete marker

1. Set `versionId` to the ID of the version to the delete marker you want to remove.
2. Send a `DELETE Object versionId` request.

Example Removing a Delete Marker

The following example removes the delete marker for `photo.gif` version 4857693.

```
DELETE /photo.gif?versionId=4857693 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

When you delete a delete marker, Amazon S3 includes in the response:

```
204 NoContent
x-amz-version-id: versionID
x-amz-delete-marker: true
```

Transitioning Object Versions

You can define lifecycle configuration rules for objects that have a well-defined lifecycle to transition object versions to the `GLACIER` storage class at a specific time in the object's lifetime. For more information, see [Object Lifecycle Management \(p. 119\)](#).

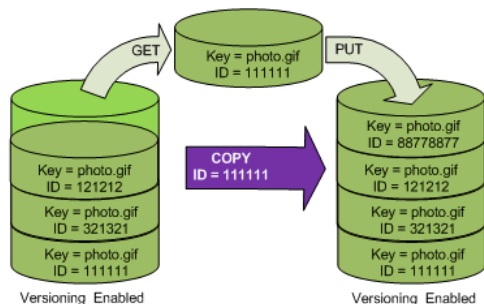
Restoring Previous Versions

One of the value propositions of versioning is the ability to retrieve previous versions of an object. There are two approaches to doing so:

- Copy a previous version of the object into the same bucket
 - The copied object becomes the current version of that object and all object versions are preserved.
- Permanently delete the current version of the object

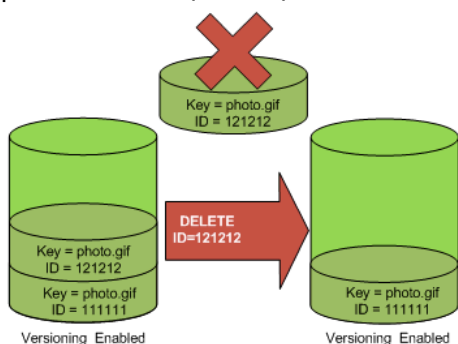
When you delete the current object version, you, in effect, turn the previous version into the current version of that object.

Because all object versions are preserved, you can make any earlier version the current version by copying a specific version of the object into the same bucket. In the following figure, the source object (ID = 111111) is copied into the same bucket. Amazon S3 supplies a new ID (88778877) and it becomes the current version of the object. So, the bucket has both the original object version (111111) and its copy (88778877).



A subsequent `GET` will retrieve version 88778877.

The following figure shows how deleting the current version (121212) of an object, which leaves the previous version (111111) as the current object.



A subsequent `GET` will retrieve version 111111.

Versioned Object Permissions

Permissions are set at the version level. Each version has its own object owner; an AWS account that creates the object version is the owner. So, you can set different permissions for different versions of the same object. To do so, you must specify the version ID of the object whose permissions you want to set in a `PUT Object versionId acl` request. For a detailed description and instructions on using ACLs, see [Identity and Access Management in Amazon S3 \(p. 301\)](#).

Example Setting Permissions for an Object Version

The following request sets the permission of the grantee, `BucketOwner@amazon.com`, to `FULL_CONTROL` on the key, `my-image.jpg`, version ID, `3HL4kqtJvjVBH40Nrjfk`.

```
PUT /my-image.jpg?acl&versionId=3HL4kqtJvjVBH40Nrjfk HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
Content-Length: 124

<AccessControlPolicy>
  <Owner>
    <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>
    <DisplayName>mtd@amazon.com</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
```

```
<ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>  
<DisplayName>BucketOwner@amazon.com</DisplayName>  
</Grantee>  
<Permission>FULL_CONTROL</Permission>  
</Grant>  
</AccessControlList>  
</AccessControlPolicy>
```

Likewise, to get the permissions of a specific object version, you must specify its version ID in a `GET Object versionId acl` request. You need to include the version ID because, by default, `GET Object acl` returns the permissions of the current version of the object.

Example Retrieving the Permissions for a Specified Object Version

In the following example, Amazon S3 returns the permissions for the key, `my-image.jpg`, version ID, `DVBH40Nr8X8gUMLUo`.

```
GET /my-image.jpg?versionId=DVBH40Nr8X8gUMLUo&acl HTTP/1.1  
Host: bucket.s3.amazonaws.com  
Date: Wed, 28 Oct 2009 22:32:00 GMT  
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU
```

For more information, see [GET Object acl](#).

Managing Objects in a Versioning-Suspended Bucket

Topics

- [Adding Objects to Versioning-Suspended Buckets \(p. 451\)](#)
- [Retrieving Objects from Versioning-Suspended Buckets \(p. 452\)](#)
- [Deleting Objects from Versioning-Suspended Buckets \(p. 452\)](#)

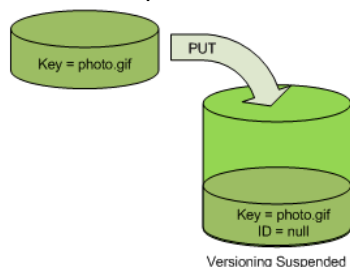
You suspend versioning to stop accruing new versions of the same object in a bucket. You might do this because you only want a single version of an object in a bucket, or you might not want to accrue charges for multiple versions.

When you suspend versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles objects in future requests. The topics in this section explain various object operations in a versioning-suspended bucket.

Adding Objects to Versioning-Suspended Buckets

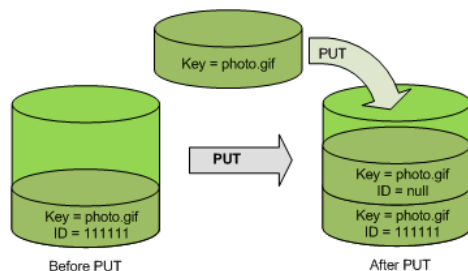
Once you suspend versioning on a bucket, Amazon S3 automatically adds a `null` version ID to every subsequent object stored thereafter (using `PUT`, `POST`, or `COPY`) in that bucket.

The following figure shows how Amazon S3 adds the version ID of `null` to an object when it is added to a version-suspended bucket.

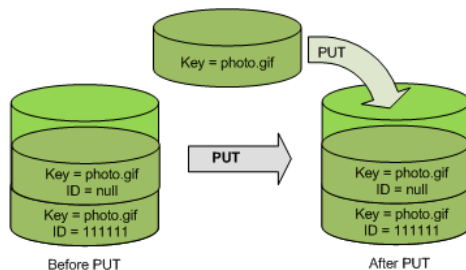


If a null version is already in the bucket and you add another object with the same key, the added object overwrites the original null version.

If there are versioned objects in the bucket, the version you `PUT` becomes the current version of the object. The following figure shows how adding an object to a bucket that contains versioned objects does not overwrite the object already in the bucket. In this case, version 111111 was already in the bucket. Amazon S3 attaches a version ID of null to the object being added and stores it in the bucket. Version 111111 is not overwritten.



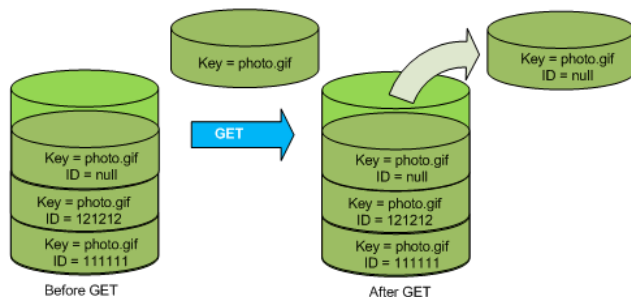
If a null version already exists in a bucket, the null version is overwritten, as shown in the following figure.



Note that although the key and version ID (`null`) of null version are the same before and after the `PUT`, the contents of the null version originally stored in the bucket is replaced by the contents of the object `PUT` into the bucket.

Retrieving Objects from Versioning-Suspended Buckets

A `GET` Object request returns the current version of an object whether you've enabled versioning on a bucket or not. The following figure shows how a simple `GET` returns the current version of an object.



Deleting Objects from Versioning-Suspended Buckets

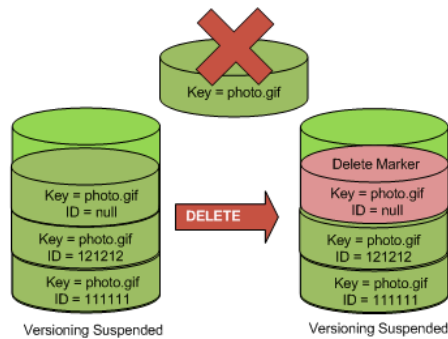
If versioning is suspended, a `DELETE` request:

- Can only remove an object whose version ID is `null`

Doesn't remove anything if there isn't a null version of the object in the bucket.

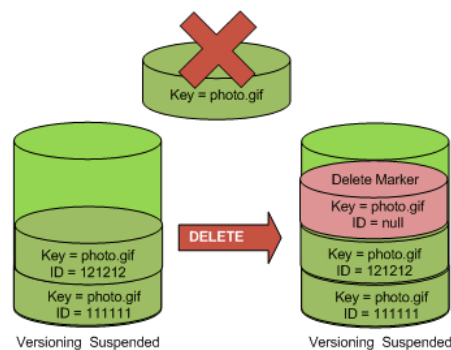
- Inserts a delete marker into the bucket.

The following figure shows how a simple `DELETE` removes a null version and Amazon S3 inserts a delete marker in its place with a version ID of `null`.

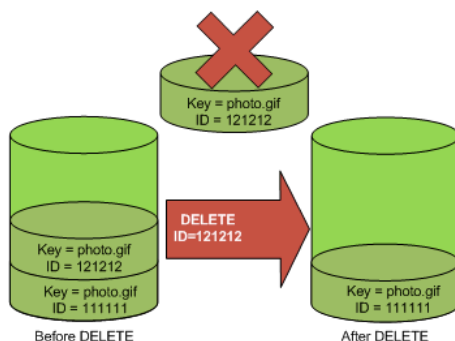


Remember that a delete marker doesn't have content, so you lose the content of the null version when a delete marker replaces it.

The following figure shows a bucket that doesn't have a null version. In this case, the `DELETE` removes nothing; Amazon S3 just inserts a delete marker.



Even in a versioning-suspended bucket, the bucket owner can permanently delete a specified version. The following figure shows that deleting a specified object version permanently removes that object. Only the bucket owner can delete a specified object version.



Locking Objects Using Amazon S3 Object Lock

With Amazon S3 object lock, you can store objects using a *write-once-read-many* (WORM) model. You can use it to prevent an object from being deleted or overwritten for a fixed amount of time or

indefinitely. Amazon S3 object lock helps you meet regulatory requirements that require WORM storage, or simply add another layer of protection against object changes and deletion.

Amazon S3 object lock has been assessed by Cohasset Associates for use in environments that are subject to SEC 17a-4, CTCC, and FINRA regulations. For more information about how Amazon S3 object lock relates to these regulations, see the [Cohasset Associates Compliance Assessment](#).

Amazon S3 object lock provides two ways to manage object retention: retention periods and legal holds.

- A *retention period* specifies a fixed period of time during which an object remains locked. During this period, your object is WORM-protected and can't be overwritten or deleted.
- A *legal hold* provides the same protection as a retention period, but it has no expiration date. Instead, a legal hold remains in place until you explicitly remove it. Legal holds are independent from retention periods.

An object version can have both a retention period and a legal hold, one but not the other, or neither. For more information, see [Amazon S3 Object Lock Overview \(p. 454\)](#).

Amazon S3 object lock works only in versioned buckets, and retention periods and legal holds apply to individual object versions. When you lock an object version, Amazon S3 stores the lock information in the metadata for that object version. Placing a retention period or legal hold on an object protects only the version specified in the request. It doesn't prevent new versions of the object from being created. If you put an object into a bucket that has the same key name as an existing, protected object, Amazon S3 creates a new version of that object, stores it in the bucket as requested, and reports the request as completed successfully. The existing, protected version of the object remains locked according to its retention configuration.

To use Amazon S3 object lock, follow these basic steps:

1. Create a new bucket with Amazon S3 object lock enabled.
2. (Optional) Configure a default retention period for objects placed in the bucket.
3. Place the objects that you want to lock in the bucket.
4. Apply a retention period, a legal hold, or both, to the objects that you want to protect.

For information about using Amazon S3 object lock on the AWS Management Console, see [How Do I Lock an Amazon S3 Object?](#) in the *Amazon Simple Storage Service Console User Guide*.

Topics

- [Amazon S3 Object Lock Overview \(p. 454\)](#)
- [Managing Amazon S3 Object Locks \(p. 457\)](#)

Amazon S3 Object Lock Overview

You can use Amazon S3 object lock to store objects using a *write-once-read-many* (WORM) model. It can help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. You can use Amazon S3 object lock to meet regulatory requirements that require WORM storage, or add an extra layer of protection against object changes and deletion.

For information about managing the lock status of your Amazon S3 objects, see [the section called "Managing Object Locks" \(p. 457\)](#).

The following sections describe the main features of Amazon S3 object lock.

Topics

- [Retention Modes \(p. 455\)](#)
- [Retention Periods \(p. 455\)](#)
- [Legal Holds \(p. 456\)](#)
- [Bucket Configuration \(p. 456\)](#)
- [Required Permissions \(p. 457\)](#)

Retention Modes

Amazon S3 object lock provides two *retention modes*:

- Governance mode
- Compliance mode

These retention modes apply different levels of protection to your objects. You can apply either retention mode to any object version that is protected by Amazon S3 object lock.

In *governance* mode, users can't overwrite or delete an object version or alter its lock settings unless they have special permissions. With governance mode, you protect objects against being deleted by most users, but you can still grant some users permission to alter the retention settings or delete the object if necessary. You can also use governance mode to test retention-period settings before creating a compliance-mode retention period. To override or remove governance-mode retention settings, a user must have the `s3:BypassGovernanceRetention` permission and must explicitly include `x-amz-bypass-governance-retention:true` as a request header with any request that requires overriding governance mode.

Note

The Amazon S3 console by default includes the `x-amz-bypass-governance-retention:true` header. If you try to delete objects protected by *governance* mode and have `s3:BypassGovernanceMode` and `s3:GetObjectLockConfiguration` or, `s3:GetObjectRetention` permissions, the operation will succeed.

In *compliance* mode, a protected object version can't be overwritten or deleted by any user, including the root user in your AWS account. When an object is locked in compliance mode, its retention mode can't be changed, and its retention period can't be shortened. Compliance mode ensures that an object version can't be overwritten or deleted for the duration of the retention period.

Note

Updating an object version's metadata, as occurs when you place or alter an object lock, doesn't overwrite the object version or reset its `Last-Modified` timestamp.

Retention Periods

A *retention period* protects an object version for a fixed amount of time. When you place a retention period on an object version, Amazon S3 stores a timestamp in the object version's metadata to indicate when the retention period expires. After the retention period expires, the object version can be overwritten or deleted unless you also placed a legal hold on the object version.

You can place a retention period on an object version either explicitly or through a bucket default setting. When you apply a retention period to an object version explicitly, you specify a *Retain Until Date* for the object version. Amazon S3 stores the Retain Until Date setting in the object version's metadata and protects the object version until the retention period expires.

When you use bucket default settings, you don't specify a Retain Until Date. Instead, you specify a duration, in either days or years, for which every object version placed in the bucket should be protected. When you place an object in the bucket, Amazon S3 calculates a Retain Until Date for the object version by adding the specified duration to the object version's creation timestamp. It stores the Retain Until

Date in the object version's metadata. The object version is then protected exactly as though you explicitly placed a lock with that retention period on the object version.

Note

If your request to place an object version in a bucket contains an explicit retention mode and period, those settings override any bucket default settings for that object version.

Like all other Amazon S3 object lock settings, retention periods apply to individual object versions. Different versions of a single object can have different retention modes and periods.

For example, suppose that you have an object that is 15 days into a 30-day retention period, and you `PUT` an object into Amazon S3 with the same name and a 60-day retention period. In this case, your `PUT` succeeds, and Amazon S3 creates a new version of the object with a 60-day retention period. The older version maintains its original retention period and becomes deletable in 15 days.

You can extend a retention period after you've applied a retention setting to an object version. To do this, submit a new lock request for the object version with a `Retain Until Date` that is later than the one currently configured for the object version. Amazon S3 replaces the existing retention period with the new, longer period. Any user with permissions to place an object retention period can extend a retention period for an object version locked in either mode.

Legal Holds

Amazon S3 object lock also enables you to place a *legal hold* on an object version. Like a retention period, a legal hold prevents an object version from being overwritten or deleted. However, a legal hold doesn't have an associated retention period and remains in effect until removed. Legal holds can be freely placed and removed by any user who has the `s3:PutObjectLegalHold` permission.

Legal holds are independent from retention periods. As long as the bucket that contains the object has Amazon S3 object lock enabled, you can place and remove legal holds regardless of whether the specified object version has a retention period set. Placing a legal hold on an object version doesn't affect the retention mode or retention period for that object version. For example, suppose that you place a legal hold on an object version while the object version is also protected by a retention period. If the retention period expires, the object doesn't lose its WORM protection. Rather, the legal hold continues to protect the object until an authorized user explicitly removes it. Similarly, if you remove a legal hold while an object version has a retention period in effect, the object version remains protected until the retention period expires.

Bucket Configuration

To use Amazon S3 object lock, you must enable it for a bucket. You can also optionally configure a default retention mode and period that applies to new objects that are placed in the bucket.

Enabling object lock

Before you can lock any objects, you have to configure a bucket to use Amazon S3 object lock. To do this, you specify when you create the bucket that you want to enable Amazon S3 object lock. After you configure a bucket for Amazon S3 object lock, you can lock objects in that bucket using retention periods, legal holds, or both.

Note

- You can only enable Amazon S3 object lock for new buckets. If you want to turn on Amazon S3 object lock for an existing bucket, contact AWS Support.
- When you create a bucket with Amazon S3 object lock enabled, Amazon S3 automatically enables versioning for the bucket.
- Once you create a bucket with Amazon S3 object lock enabled, you can't disable object lock or suspend versioning for the bucket.

For information about enabling Amazon S3 object lock on the console, see [How Do I Lock an Amazon S3 Object?](#) in the *Amazon Simple Storage Service Console User Guide*.

Default Retention Settings

When you turn on Amazon S3 object lock for a bucket, the bucket can store protected objects. However, the setting doesn't automatically protect objects that you put into the bucket. If you want to automatically protect object versions that are placed in the bucket, you can configure a default retention period. Default settings apply to all new objects that are placed in the bucket, unless you explicitly specify a different retention mode and period for an object when you create it.

Tip

If you want to enforce the bucket default retention mode and period for all new object versions placed in a bucket, set the bucket defaults and deny users permission to configure object retention settings. Amazon S3 then applies the default retention mode and period to new object versions placed in the bucket, and rejects any request to put an object that includes a retention mode and setting.

Bucket default settings require both a mode and a period. A bucket default mode is either *governance* or *compliance*. For more information, see [Retention Modes \(p. 455\)](#).

A default retention period is described not as a timestamp, but as a period either in days or in years. When you place an object version in a bucket with a default retention period, Amazon S3 object lock calculates a *Retain Until Date*. It does this by adding the default retention period to the creation timestamp for the object version. Amazon S3 stores the resulting timestamp as the object version's Retain Until Date, as if you had calculated the timestamp manually and placed it on the object version yourself.

Default settings apply only to new objects that are placed in the bucket. Placing a default retention setting on a bucket doesn't place any retention settings on objects that already exist in the bucket.

Important

Object locks apply to individual object versions only. If you place an object in a bucket that has a default retention period, and you don't explicitly specify a retention period for that object, Amazon S3 creates the object with a retention period that matches the bucket default. After the object is created, its retention period is independent from the bucket's default retention period. Changing a bucket's default retention period doesn't change the existing retention period for any objects in that bucket.

Note

If you configure a default retention period on a bucket, requests to upload objects in such a bucket must include the `Content-MD5` header. For more information, see [Put Object](#) in the *Amazon Simple Storage Service API Reference*.

Required Permissions

Amazon S3 object lock operations require specific permissions. For more information, see [Permissions for Object Operations \(p. 345\)](#).

Managing Amazon S3 Object Locks

Amazon S3 object lock lets you store objects in Amazon S3 using a *write once, read many* (WORM) model. You can use it to view, configure, and manage the object lock status of your Amazon S3 objects. For more information about Amazon S3 object lock capabilities, see [Amazon S3 Object Lock Overview \(p. 454\)](#).

Topics

- [Viewing the Lock Information for an Object \(p. 458\)](#)
- [Bypassing Governance Mode \(p. 458\)](#)
- [Configuring Events and Notifications \(p. 458\)](#)

- [Setting Retention Limits \(p. 459\)](#)
- [Managing Delete Markers and Object Lifecycles \(p. 459\)](#)
- [Using Object Lock with Replication \(p. 459\)](#)

Viewing the Lock Information for an Object

You can view the object lock status of an Amazon S3 object version using the `GET Object` or `HEAD Object` commands. Both commands return the retention mode, `Retain Until Date`, and the legal-hold status for the specified object version.

To view an object version's retention mode and retention period, you must have the `s3:GetObjectRetention` permission. To view an object version's legal hold status, you must have the `s3:GetObjectLegalHold` permission. If you `GET` or `HEAD` an object version but don't have the necessary permissions to view its lock status, the request succeeds. However, it doesn't return information that you don't have permission to view.

To view a bucket's default retention configuration (if it has one), request the bucket's object lock configuration. To do this, you must have the `s3:GetBucketObjectLockConfiguration` permission. If you make a request for an object lock configuration against a bucket that doesn't have Amazon S3 object lock enabled, Amazon S3 returns an error.

You can configure Amazon S3 inventory reports on your buckets to include the `Retain Until Date`, `object lock Mode`, and `Legal Hold Status` for all objects in a bucket. For more information, see [Amazon S3 Inventory \(p. 422\)](#).

Bypassing Governance Mode

You can perform operations on object versions that are locked in governance mode as if they were unprotected if you have the `s3:BypassGovernanceRetention` permission. These operations include deleting an object version, shortening the retention period, or removing the object lock by placing a new lock with empty parameters. To bypass governance mode, you must explicitly indicate in your request that you want to bypass this mode. To do this, include the `x-amz-bypass-governance-retention:true` header with your request, or use the equivalent parameter with requests made through the AWS CLI, or AWS SDKs. The AWS Management Console automatically applies this header for requests made through the console if you have the permission required to bypass governance mode.

Note

Bypassing governance mode doesn't affect an object version's legal hold status. If an object version has a legal hold enabled, the legal hold remains in force and prevents requests to overwrite or delete the object version.

Configuring Events and Notifications

You can configure Amazon S3 events for object-level operations in an object lock bucket. When `PUT Object`, `HEAD Object`, and `GET Object` calls include object lock metadata, events for these calls include those metadata values. When object lock metadata is added to or updated for an object, those actions also trigger events. These events occur whenever you `PUT` or `GET` object retention or legal-hold information.

For more information about Amazon S3 events, see [Configuring Amazon S3 Event Notifications \(p. 530\)](#).

You can use Amazon S3 event notifications to track access and changes to your object lock configurations and data using AWS CloudTrail. For information about CloudTrail, see the [AWS CloudTrail Documentation](#).

You can also use Amazon CloudWatch to generate alerts based on this data. For information about CloudWatch, see the [Amazon CloudWatch Documentation](#).

Setting Retention Limits

You can set minimum and maximum allowable retention periods for a bucket using a bucket policy. You do this using the `s3:object-lock-remaining-retention-days` condition key. The following example shows a bucket policy that sets a maximum retention period of 10 days.

```
{
  "Version": "2012-10-17",
  "Id": "<Policy1436912751980>",
  "Statement": [
    {
      "Sid": "<Stmnt1436912698057>",
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "s3:PutObjectRetention"
      ],
      "Resource": "arn:aws:s3:::<example-bucket>/*",
      "Condition": {
        "NumericGreaterThan": {
          "s3:object-lock-remaining-retention-days": "10"
        }
      }
    }
  ]
}
```

Note

If your bucket is the destination bucket for a replication policy and you want to set up minimum and maximum allowable retention periods for object replicas that are created using replication, you must include the `s3:ReplicateObject` action in your bucket policy.

For more information about using bucket policies, see [Using Bucket Policies and User Policies \(p. 341\)](#).

Managing Delete Markers and Object Lifecycles

Although you can't delete a protected object version, you can still create a delete marker for that object. Placing a delete marker on an object doesn't delete any object version. However, it makes Amazon S3 behave in most ways as though the object has been deleted. For more information, see [Working with Delete Markers \(p. 446\)](#).

Note

Delete markers are not WORM-protected, regardless of any retention period or legal hold in place on the underlying object.

Object lifecycle management configurations continue to function normally on protected objects, including placing delete markers. However, protected object versions remain safe from being deleted or overwritten by a lifecycle configuration. For more information about managing object lifecycles, see [Object Lifecycle Management \(p. 119\)](#).

Using Object Lock with Replication

You can use Amazon S3 object lock with replication to enable automatic, asynchronous copying of locked objects and their retention metadata, across S3 buckets in different or the same AWS Regions. When you use replication, objects in a *source bucket* are replicated to a *destination bucket*. For more information, see [Replication \(p. 551\)](#).

To set up object lock with replication, you can choose one of the following options.

Option 1: Enable object lock first.

1. Enable object lock on the destination bucket, or on both the source and the destination bucket.
2. Set up replication between the source and the destination buckets.

Option 2: Set up replication first.

1. Set up replication between the source and destination buckets.
2. Enable object lock on just the destination bucket, or on both the source and destination buckets.

To complete step 2 in the preceding options, you must contact [AWS Support](#). This is required to make sure that replication is configured correctly.

Before you contact AWS Support, review the following requirements for setting up object lock with replication:

- The Amazon S3 destination bucket must have object lock enabled on it.
- You must grant two new permissions on the source S3 bucket in the AWS Identity and Access Management (IAM) role that you use to set up replication. The two new permissions are `s3:GetObjectRetention` and `s3:GetObjectLegalHold`. If the role has an `s3:Get*` permission, it satisfies the requirement. For more information, see [Setting Up Permissions for Replication \(p. 564\)](#).

For more information about Amazon S3 object lock, see [Locking Objects Using Amazon S3 Object Lock \(p. 453\)](#).

Infrastructure Security in Amazon S3

As a managed service, Amazon S3 is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

Access to Amazon S3 via the network is through AWS published APIs. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2. Clients must also support cipher suites with Perfect Forward Secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Additionally, requests must be signed using AWS Signature V4 or AWS Signature V2, requiring valid credentials to be provided.

These APIs are callable from any network location. However, Amazon S3 does support resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon S3 bucket policies to control access to buckets from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints, or specific VPCs. Effectively, this isolates network access to a given Amazon S3 bucket from only the specific VPC within the AWS network. For more information, see [Example Bucket Policies for VPC Endpoints for Amazon S3 \(p. 378\)](#).

The following security best practices also address infrastructure security in Amazon S3:

- [Consider VPC endpoints for Amazon S3 access](#)
- [Identify and audit all your Amazon S3 buckets](#)

Configuration and Vulnerability Analysis in Amazon S3

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- [Compliance Validation for Amazon S3 \(p. 422\)](#)
- [Shared Responsibility Model](#)
- [Amazon Web Services: Overview of Security Processes](#) (whitepaper)

The following security best practices also address configuration and vulnerability analysis in Amazon S3:

- [Identify and audit all your Amazon S3 buckets](#)
- [Enable AWS Config](#)

Security Best Practices for Amazon S3

Amazon S3 provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Topics

- [Amazon S3 Preventative Security Best Practices \(p. 463\)](#)
- [Amazon S3 Monitoring and Auditing Best Practices \(p. 465\)](#)

Amazon S3 Preventative Security Best Practices

The following best practices for Amazon S3 can help prevent security incidents.

Ensure that your Amazon S3 buckets use the correct policies and are not publicly accessible

Unless you explicitly require anyone on the internet to be able to read or write to your S3 bucket, you should ensure that your S3 bucket is not public. The following are some of the steps you can take:

- Use Amazon S3 block public access. With Amazon S3 block public access, account administrators and bucket owners can easily set up centralized controls to limit public access to their Amazon S3 resources that are enforced regardless of how the resources are created. For more information, see [Using Amazon S3 Block Public Access \(p. 414\)](#).
- Identify Amazon S3 bucket policies that allow a wildcard identity such as Principal "*" (which effectively means "anyone") or allows a wildcard action "*" (which effectively allows the user to perform any action in the Amazon S3 bucket).
- Similarly, note Amazon S3 bucket access control lists (ACLs) that provide read, write, or full-access to "Everyone" or "Any authenticated AWS user."
- Use the `ListBuckets` API to scan all of your Amazon S3 buckets. Then use `GetBucketAcl`, `GetBucketWebsite`, and `GetBucketPolicy` to determine whether the bucket has compliant access controls and configuration.
- Use [AWS Trusted Advisor](#) to inspect your Amazon S3 implementation.
- Consider implementing on-going detective controls using the [s3-bucket-public-read-prohibited](#) and [s3-bucket-public-write-prohibited](#) managed AWS Config Rules.

For more information, see [Setting Bucket and Object Access Permissions](#) in the *Amazon Simple Storage Service Console User Guide*.

Implement least privilege access

When granting permissions, you decide who is getting what permissions to which Amazon S3 resources. You enable specific actions that you want to allow on those resources. Therefore you should grant only the permissions that are required to perform a task. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

The following tools are available to implement least privilege access:

- [IAM user policies](#) and [Permissions Boundaries for IAM Entities](#)
- [Amazon S3 bucket policies](#)
- [Amazon S3 access control lists \(ACLs\)](#)
- [Service Control Policies](#)

For guidance on what to consider when choosing one or more of the preceding mechanisms, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 301\)](#).

Use IAM roles for applications and AWS services that require Amazon S3 access

For applications on Amazon EC2 or other AWS services to access Amazon S3 resources, they must include valid AWS credentials in their AWS API requests. You should not store AWS credentials directly in the application or Amazon EC2 instance. These are long-term credentials that are not automatically rotated and could have a significant business impact if they are compromised.

Instead, you should use an IAM role to manage temporary credentials for applications or services that need to access Amazon S3. When you use a role, you don't have to distribute long-term credentials (such as a user name and password or access keys) to an Amazon EC2 instance or AWS service such as AWS Lambda. The role supplies temporary permissions that applications can use when they make calls to other AWS resources.

For more information, see the following topics in the *IAM User Guide*:

- [IAM Roles](#)
- [Common Scenarios for Roles: Users, Applications, and Services](#)

Enable multi-factor authentication (MFA) Delete

MFA Delete can help prevent accidental bucket deletions. If MFA Delete is not enabled, any user with the password of a sufficiently privileged root or IAM user could permanently delete an Amazon S3 object.

MFA Delete requires additional authentication for either of the following operations:

- Changing the versioning state of your bucket
- Permanently deleting an object version

For more information, see [MFA Delete \(p. 433\)](#).

Consider encryption of data at rest

You have the following options for protecting data at rest in Amazon S3:

- **Server-Side Encryption** – Request Amazon S3 to encrypt your object before saving it on disks in its data centers and then decrypt it when you download the objects. Server-side encryption can help reduce risk to your data by encrypting the data with a key that is stored in a different mechanism than the mechanism that stores the data itself.

Amazon S3 provides multiple server-side encryption options. For more information, see [Protecting Data Using Server-Side Encryption \(p. 265\)](#).

- **Client-Side Encryption** – Encrypt data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and related tools. As with server-side encryption, client-side encryption can help reduce risk by encrypting the data with a key that is stored in a different mechanism than the mechanism that stores the data itself.

Amazon S3 provides multiple client-side encryption options. For more information, see [Protecting Data Using Client-Side Encryption \(p. 293\)](#).

Enforce encryption of data in transit

You can use HTTPS (TLS) to help prevent potential attackers from eavesdropping on or manipulating network traffic using person-in-the-middle or similar attacks. You should allow only encrypted connections over HTTPS (TLS) using the [aws:SecureTransport](#) condition on Amazon S3 bucket policies.

Also consider implementing on-going detective controls using the [s3-bucket-ssl-requests-only](#) managed AWS Config rule.

Consider Amazon S3 Object Lock

[Amazon S3 Object Lock](#) enables you to store objects using a "Write Once Read Many" (WORM) model. Amazon S3 Object Lock can help prevent accidental or inappropriate deletion of data. For example, you could use Amazon S3 Object Lock to help protect your AWS CloudTrail logs.

Enable versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures.

Also consider implementing on-going detective controls using the [s3-bucket-versioning-enabled](#) managed AWS Config rule.

For more information, see [Using Versioning \(p. 432\)](#).

Consider Amazon S3 cross-region replication

Although Amazon S3 stores your data across multiple geographically diverse Availability Zones by default, compliance requirements might dictate that you store data at even greater distances. Cross-region replication (CRR) allows you to replicate data between distant AWS Regions to help satisfy these requirements. CRR enables automatic, asynchronous copying of objects across buckets in different AWS Regions. For more information, see [Replication \(p. 551\)](#).

Note

CRR requires that both source and destination S3 buckets have versioning enabled.

Also consider implementing on-going detective controls using the [s3-bucket-replication-enabled](#) managed AWS Config rule.

Consider VPC endpoints for Amazon S3 access

A VPC endpoint for Amazon S3 is a logical entity within an Amazon Virtual Private Cloud (Amazon VPC) that allows connectivity only to Amazon S3. You can use Amazon S3 bucket policies to control access to buckets from specific Amazon VPC endpoints, or specific VPCs. A VPC endpoint can help prevent traffic from potentially traversing the open internet and being subject to open internet environment.

VPC endpoints for Amazon S3 provide two ways to control access to your Amazon S3 data:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint.
- You can control which VPCs or VPC endpoints have access to your S3 buckets by using S3 bucket policies.
- You can help prevent data exfiltration by using a VPC that does not have an internet gateway.

For more information, see [Example Bucket Policies for VPC Endpoints for Amazon S3 \(p. 378\)](#).

Amazon S3 Monitoring and Auditing Best Practices

The following best practices for Amazon S3 can help detect potential security weaknesses and incidents.

Identify and audit all your Amazon S3 buckets

Identification of your IT assets is a crucial aspect of governance and security. You need to have visibility of all your Amazon S3 resources to assess their security posture and take action on potential areas of weakness.

Use Tag Editor to identify security-sensitive or audit-sensitive resources, then use those tags when you need to search for these resources. For more information, see [Searching for Resources to Tag](#).

Use Amazon S3 inventory to audit and report on the replication and encryption status of your objects for business, compliance, and regulatory needs. For more information, see [Amazon S3 Inventory](#) (p. 422).

Create resource groups for your Amazon S3 resources. For more information, see [What Is AWS Resource Groups?](#)

Implement monitoring using AWS monitoring tools

Monitoring is an important part of maintaining the reliability, security, availability, and performance of Amazon S3 and your AWS solutions. AWS provides several tools and services to help you monitor Amazon S3 and your other AWS services. For example, you can monitor CloudWatch metrics for Amazon S3, particularly `PutRequests`, `GetRequests`, `4xxErrors`, and `DeleteRequests`. For more information, see [Monitoring Metrics with Amazon CloudWatch](#) (p. 611). For more information, see [Monitoring Amazon S3](#) (p. 610).

For a second example, see [Example: Amazon S3 Bucket Activity](#). This example describes how to create an Amazon CloudWatch alarm that is triggered when an Amazon S3 API call is made to PUT or DELETE bucket policy, bucket lifecycle, or bucket replication, or to PUT a bucket ACL.

Enable Amazon S3 server access logging

Server access logging provides detailed records of the requests that are made to a bucket. Server access logs can assist you in security and access audits, help you learn about your customer base, and understand your Amazon S3 bill. For instructions on enabling server access logging, see [Amazon S3 Server Access Logging](#) (p. 647).

Also consider implementing on-going detective controls using the [s3-bucket-logging-enabled](#) AWS Config managed rule.

Use AWS CloudTrail

AWS CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon S3. Using the information collected by CloudTrail, you can determine the request that was made to Amazon S3, the IP address from which the request was made, who made the request, when it was made, and additional details. For example, you can identify CloudTrail entries for Put actions that impact data access, in particular `PutBucketAcl`, `PutObjectAcl`, `PutBucketPolicy`, and `PutBucketWebsite`. For more information, see [Logging Amazon S3 API Calls by Using AWS CloudTrail](#) (p. 621).

You can also use CloudTrail in conjunction with Amazon S3 server access logging. Amazon S3 server access logging provides an access log of requests made to the bucket, but does not provide visibility into API operations at the object-level. CloudTrail supports Amazon S3 object-level API operations such as `GetObject`, `DeleteObject`, and `PutObject`. Monitoring these events, called *data events*, can provide valuable insight into operations involving your Amazon S3 objects. For more information, see [Data Events](#) in the *AWS CloudTrail User Guide*.

Enable AWS Config

Several of the best practices listed in this topic suggest creating AWS Config rules. AWS Config enables you to assess, audit, and evaluate the configurations of your AWS resources. AWS Config monitors resource configurations, allowing you to evaluate the recorded configurations against the desired secure configurations. Using AWS Config, you can review changes in configurations and relationships between AWS resources, investigate detailed resource configuration histories, and determine your overall compliance against the configurations specified in your internal guidelines. This can help you simplify compliance auditing, security analysis, change management, and operational troubleshooting. For more information, see [Setting Up AWS Config with the Console](#) in the *AWS Config Developer Guide*. When specifying the resource types to record, ensure that you include Amazon S3 resources.

For an example of how to use AWS Config to monitor for and respond to Amazon S3 buckets that allow public access, see [How to Use AWS Config to Monitor for and Respond to Amazon S3 Buckets Allowing Public Access](#) on the *AWS Security Blog*.

Consider using Amazon Macie with Amazon S3

Macie uses machine learning to automatically discover, classify, and protect sensitive data in AWS. Macie recognizes sensitive data such as personally identifiable information (PII) or intellectual property. It provides you with dashboards and alerts that give visibility into how this data is being accessed or moved. For more information, see [What Is Amazon Macie?](#)

Monitor AWS security advisories

You should regularly check security advisories posted in Trusted Advisor for your AWS account. In particular, note warnings about Amazon S3 buckets with “open access permissions.” You can do this programmatically using [describe-trusted-advisor-checks](#).

Further, actively monitor the primary email address registered to each of your AWS accounts. AWS will contact you, using this email address, about emerging security issues that might affect you.

AWS operational issues with broad impact are posted on the [AWS Service Health Dashboard](#). Operational issues are also posted to individual accounts via the Personal Health Dashboard. For more information, see the [AWS Health Documentation](#).

Performing Batch Operations

You can use Amazon S3 batch operations to perform large-scale batch operations on Amazon S3 objects. Amazon S3 batch operations can execute a single operation on lists of Amazon S3 objects that you specify. A single job can perform the specified operation on billions of objects containing exabytes of data. Amazon S3 tracks progress, sends notifications, and stores a detailed completion report of all actions, providing a fully managed, auditable, serverless experience. You can use Amazon S3 batch operations through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

Use Amazon S3 batch operations to copy objects and set object tags or access control lists (ACLs). You can also initiate object restores from Amazon S3 Glacier or invoke an AWS Lambda function to perform custom actions using your objects. You can perform these operations on a custom list of objects, or you can use an Amazon S3 inventory report to make generating even the largest lists of objects easy. Amazon S3 batch operations use the same Amazon S3 APIs that you already use with Amazon S3, so you'll find the interface familiar.

Topics

- [Terminology \(p. 468\)](#)
- [The Basics: Amazon S3 Batch Operations Jobs \(p. 468\)](#)
- [Creating an Amazon S3 Batch Operations Job \(p. 470\)](#)
- [Operations \(p. 475\)](#)
- [Managing Batch Operations Jobs \(p. 485\)](#)
- [Amazon S3 Batch Operations Examples \(p. 489\)](#)

Terminology

This section uses the terms *jobs*, *operations*, and *tasks*, which are defined as follows:

Job

A job is the basic unit of work for Amazon S3 batch operations. A job contains all of the information necessary to execute the specified operation on the objects listed in the manifest. After you provide this information and request that the job begin, the job executes the operation for each object in the manifest.

Operation

An operation is a single command that you want a job to execute. Each job contains only one type of operation with one set of parameters, which Amazon S3 batch operations execute for each object.

Task

A task is the unit of execution for a job. A task represents a single call to an Amazon S3 or AWS Lambda API operation to perform the job's operation on a single object. Over the course of a job's lifetime, Amazon S3 batch operations create one task for each object specified in the manifest.

The Basics: Amazon S3 Batch Operations Jobs

You can use Amazon S3 batch operations to perform large-scale batch operations on Amazon S3 objects. Amazon S3 batch operations can execute a single operation on lists of Amazon S3 objects that you specify.

Topics

- [How an Amazon S3 Batch Operations Job Works \(p. 469\)](#)
- [Specifying a Manifest \(p. 469\)](#)

How an Amazon S3 Batch Operations Job Works

A job is the basic unit of work for Amazon S3 batch operations. A job contains all of the information necessary to execute the specified operation on a list of objects.

To create a job, you give Amazon S3 batch operations a list of objects and specify the action to perform on those objects. Amazon S3 batch operations support the following operations:

- [PUT copy object](#)
- [PUT object tagging](#)
- [PUT object ACL](#)
- [Initiate Glacier restore](#)
- [Invoke an AWS Lambda function](#)

The objects that you want a job to act on are listed in a manifest object. A job performs the specified operation on each object that is included in its manifest. You can use a CSV-formatted [Amazon S3 Inventory \(p. 422\)](#) report as a manifest, which makes it easy to create large lists of objects located in a bucket. You can also specify a manifest in a simple CSV format that enables you to perform batch operations on a customized list of objects contained within a single bucket.

After you create a job, Amazon S3 processes the list of objects in the manifest and executes the specified operation against each object. While a job is executing, you can monitor its progress programmatically or through the Amazon S3 console. You can also configure a job to generate a completion report when it finishes. The completion report describes the results of each task that was executed by the job. For more information about monitoring jobs, see [Managing Batch Operations Jobs \(p. 485\)](#).

Specifying a Manifest

A manifest is an Amazon S3 object that lists object keys that you want Amazon S3 to act upon. To create a manifest for a job, you specify the manifest object key, ETag, and optional version ID. The contents of the manifest must be URL encoded. Manifests that use server-side encryption with customer-provided keys (SSE-C) and server-side encryption with AWS KMS managed keys (SSE-KMS) are not supported. Your manifest must contain the bucket name, object key, and optionally, the object version for each object. Any other fields in the manifest are not used by Amazon S3 batch operations.

You can specify a manifest in a create job request using one of the following two formats.

- Amazon S3 inventory report — must be a CSV-formatted Amazon S3 inventory report. You must specify the `manifest.json` file that is associated with the inventory report. For more information about inventory reports, see [Amazon S3 Inventory \(p. 422\)](#). If the inventory report includes version IDs, Amazon S3 batch operations operate on the specific object versions.
- CSV file — Each row in the file must include the bucket name, object key, and optionally, the object version. Object keys must be URL-encoded, as shown in the following examples. The manifest must either include version IDs for all objects or omit version IDs for all objects. For more information about the CSV manifest format, see [JobManifestSpec](#) in the *Amazon Simple Storage Service API Reference*.

The following is an example manifest in CSV format without version IDs:

```
Examplebucket,objectkey1  
Examplebucket,objectkey2
```

```
Examplebucket,objectkey3
Examplebucket,photos/jpgs/objectkey4
Examplebucket,photos/jpgs/newjersey/objectkey5
Examplebucket,object%20key%20with%20spaces
```

The following is an example manifest in CSV format including version IDs:

```
Examplebucket,objectkey1,PZ9ibn9D5lP6p298B7S9_ceqx1n5EJ0p
Examplebucket,objectkey2,YY_ouuAJByNW1LRBfFMfxMge7XQWxMBF
Examplebucket,objectkey3,jbo9_jhdPEyB4RrmOxWS0kU0EoNrU_oI
Examplebucket,photos/jpgs/objectkey4,6EqlikJJxLTsHsnbZbSRffn24_eh5Ny4
Examplebucket,photos/jpgs/newjersey/objectkey5,imHf3FAiRsvBW_EHB8GOu.NHunHO1gVs
Examplebucket,object%20key%20with%20spaces,9HkPvDaZY5MVbMhn6TMn1YTb5ArQAo3w
```

Important

If the objects in your manifest are in a versioned bucket, you should specify the version IDs for the objects. When you create a job, Amazon S3 batch operations parse the entire manifest before running the job. However, it doesn't take a "snapshot" of the state of the bucket. Because manifests can contain billions of objects, jobs might take a long time to run. If you overwrite an object with a new version while a job is running, and you didn't specify a version ID for that object, Amazon S3 performs the operation on the latest version of the object, and not the version that existed when you created the job. The only way to avoid this behavior is to specify version IDs for the objects that are listed in the manifest.

Creating an Amazon S3 Batch Operations Job

This section describes the information that you need to create an Amazon S3 batch operations job. It also describes the results of a `Create Job` request.

Creating a Job Request

To create a job, you must provide the following information:

Operation

Specify the operation that you want Amazon S3 batch operations to execute against the objects in the manifest. Each operation type accepts parameters that are specific to that operation, which enables you to perform the same tasks as if you performed the operation one-by-one on each object.

Manifest

The manifest is a list of all of the objects that you want Amazon S3 batch operations to execute the specified action on. You can use a CSV-formatted [Amazon S3 Inventory \(p. 422\)](#) report as a manifest or use your own customized CSV list of objects. For more information about manifests, see [Specifying a Manifest \(p. 469\)](#).

Priority

Use job priorities to indicate the relative priority of this job to others running in your account. A higher number indicates higher priority.

Job priorities only have meaning relative to the priorities set for other jobs in the same account and Region, so you can choose whatever numbering system works for you. For example, you might want to assign all `Initiate Restore Object` jobs a priority of 1, all `PUT Object Copy` jobs a priority of 2, and all `Put Object ACL` jobs a priority of 3. Batch operations prioritize jobs according to priority numbers, but strict ordering isn't guaranteed. Thus, you shouldn't use job priorities to ensure

that any one job will start or finish before any other job. If you need to ensure strict ordering, wait until one job has finished before starting the next.

RoleArn

You must specify an IAM role to run the job. The IAM role that you use must have sufficient permissions to perform the operation that is specified in the job. For example, to run an `PUT Object Copy` job, the IAM role must have `s3:GetObject` permissions for the source bucket and `s3:PutObject` permissions for the destination bucket. The role also needs permissions to read the manifest and write the job-completion report. For more information about IAM roles, see [IAM Roles](#). For more information about Amazon S3 permissions, see [Specifying Permissions in a Policy \(p. 345\)](#).

Report

Specify whether you want Amazon S3 batch operations to generate a completion report. If you request a job-completion report, then you must also provide the parameters for the report in this element. The necessary information includes the bucket where you want to store the report, the format of the report, whether you want the report to include the details of all tasks or only failed tasks, and an optional prefix string.

Description (Optional)

You can also provide a description of up to 256 characters to help you track and monitor your job. Amazon S3 includes this description whenever it returns information about a job or displays job details on the Amazon S3 console. You can then easily sort and filter jobs according to the descriptions that you assigned. Descriptions don't need to be unique, so you can use descriptions as categories (for example, "Weekly Log Copy Jobs") to help you track groups of similar jobs.

Creating a Job Response

If the `Create Job` request succeeds, Amazon S3 returns a job ID. The job ID is a unique identifier that Amazon S3 generates automatically so that you can identify your batch operations job and monitor its status.

When you create a job through the AWS CLI, AWS SDKs, or REST API, you can set Amazon S3 batch operations to begin processing the job automatically. The job runs as soon as it's ready and not waiting behind higher-priority jobs. When you create a job through the AWS Management Console, you must review the job details and confirm that you want to run it before batch operations can begin to process it. After you confirm that you want to run the job, it progresses as though you had created it through one of the other methods. If a job remains in the suspended state for over 30 days, it will fail.

Granting Permissions for Amazon S3 Batch Operations

This section describes how to grant the necessary permissions required for creating and performing batch operations jobs.

Topics

- [Required Permissions for Creating an Amazon S3 Batch Operations Job \(p. 471\)](#)
- [Creating an Amazon S3 Batch Operations IAM Role \(p. 472\)](#)

Required Permissions for Creating an Amazon S3 Batch Operations Job

To create an Amazon S3 batch operations job, the `s3:CreateJob` permission is required. The same entity creating the job must also have the `iam:PassRole` permission to pass the AWS Identity and

Access Management (IAM) role specified for the job to Amazon S3 batch operations. For information about creating this IAM role, see the next topic [Creating an Amazon S3 Batch Operations IAM Role](#) (p. 472).

Creating an Amazon S3 Batch Operations IAM Role

Amazon S3 must have your permissions to perform batch operations on your behalf. You grant these permissions through an AWS Identity and Access Management (IAM) role. This section provides examples of the trust and permissions policies you use when creating an IAM role. For more information, see [IAM Roles](#).

Trust Policy

You attach the following trust policy to the IAM role to allow the Amazon S3 batch operations service principal to assume the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "batchoperations.s3.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Permissions Policies

Depending on the type of operations, you can attach one of the following policies.

Note

- Regardless of the operation, Amazon S3 needs permissions to read your manifest object from your S3 bucket and optionally write a report to your bucket. Therefore, all of the following policies include these permissions.
- For Amazon S3 inventory report manifests, Amazon S3 batch operations require permission to read the manifest.json object and all associated CSV data files.
- Version-specific permissions such as `s3:GetObjectVersion` are only required when you are specifying the version ID of the objects.

- PUT copy object

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl",
        "s3:PutObjectTagging"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::{DestinationBucket}/*"
    },
    {
      "Action": [
```

```

        "s3:GetObject",
        "s3:GetObjectAcl",
        "s3:GetObjectTagging"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:s3:::{SourceBucket}/*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetBucketLocation"
    ],
    "Resource": [
        "arn:aws:s3:::{ManifestBucket}/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation"
    ],
    "Resource": [
        "arn:aws:s3:::{ReportBucket}/*"
    ]
}
]
}

```

- PUT object tagging

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectTagging",
                "s3:PutObjectVersionTagging"
            ],
            "Resource": "arn:aws:s3:::{TargetResource}/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::{ManifestBucket}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::{ReportBucket}/*"
            ]
        }
    ]
}

```

```
]
}
```

- PUT object ACL

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObjectAcl",
        "s3:PutObjectVersionAcl"
      ],
      "Resource": "arn:aws:s3:::{TargetResource}/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::{ManifestBucket}/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::{ReportBucket}/*"
      ]
    }
  ]
}
```

- Initiate Glacier restore

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:RestoreObject"
      ],
      "Resource": "arn:aws:s3:::{TargetResource}/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::{ManifestBucket}/*"
      ]
    }
  ]
}
```

```
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation"
    ],
    "Resource": [
        "arn:aws:s3:::{ReportBucket}/*"
    ]
}
]
```

Related Resources

- [The Basics: Amazon S3 Batch Operations Jobs \(p. 468\)](#)
- [Operations \(p. 475\)](#)
- [Managing Batch Operations Jobs \(p. 485\)](#)

Operations

Amazon S3 batch operations support five different operations. The topics in this section describe each of the operations.

Topics

- [PUT Object Copy \(p. 475\)](#)
- [Initiate Restore Object \(p. 476\)](#)
- [Invoking a Lambda Function from Amazon S3 Batch Operations \(p. 477\)](#)
- [Put Object ACL \(p. 484\)](#)
- [Put Object Tagging \(p. 484\)](#)

PUT Object Copy

The PUT object copy operation copies each object specified in the manifest. You can copy objects to a different bucket in the same AWS Region or to a bucket in a different Region. Amazon S3 batch operations support most options available through Amazon S3 for copying objects. These options include setting object metadata, setting permissions, and changing an object's storage class. For more information about the functionality available through Amazon S3 for copying objects, see [Copying Objects \(p. 210\)](#).

Restrictions and Limitations

- All source objects must be in one bucket.
- All destination objects must be in one bucket.
- You must have read permissions for the source bucket and write permissions for the destination bucket.
- Objects to be copied can be up to 5 GB in size.
- PUT Object Copy jobs must be created in the destination region, i.e. the region you intend to copy the objects to.
- All PUT Object Copy options are supported except for conditional checks on ETags and server-side encryption with customer-provided encryption keys.

- If the buckets are unversioned, you will overwrite objects with the same key names.
- Objects are not necessarily copied in the same order as they are listed in the manifest. So for versioned buckets, if preserving current/non-current version order is important, you should copy all non-current versions first and later copy the current versions in a subsequent job after the first job is complete.

Initiate Restore Object

The `InitiateRestore` operation sends a restore request to Amazon S3 Glacier for each object that is specified in the manifest. To create an Initiate Restore Object job, you must include two elements with your request:

- **ExpirationInDays**

When you restore an object from S3 Glacier, the restored object is only a temporary copy, which Amazon S3 deletes after a fixed period of time. This element specifies how long the temporary copy will remain available in Amazon S3. After the temporary copy expires, you can only retrieve the object by restoring it from S3 Glacier again. For more information about object restoration, see [Restoring Archived Objects \(p. 248\)](#).

- **GlacierJobTier**

Amazon S3 can restore objects from S3 Glacier according to three different retrieval tiers: Expedited, Standard, and Bulk. Amazon S3 batch operations support only the Standard and Bulk tiers. For more information about S3 Glacier retrieval tiers, see [Archive Retrieval Options \(p. 249\)](#). For more information about pricing for each tier, see the "Retrieval pricing" section at [Amazon S3 Glacier pricing](#).

Important

The Initiate Restore Object job only initiates the request to restore objects. Amazon S3 batch operations report the job as complete for each object after the request has been initiated for that object. Amazon S3 doesn't update the job or otherwise notify you when the objects have been restored. However, you can use event notifications to receive notifications when the objects are available in Amazon S3. For more information, see [Configuring Amazon S3 Event Notifications \(p. 530\)](#).

Overlapping Restores

If your Initiate Restore Object job tries to restore an object that is already in the process of being restored, Amazon S3 batch operations will behave as follows:

The restore operation succeeds for the object if either of the following conditions are true:

- Compared to the restoration request already in progress, this job's `ExpirationInDays` is the same and `GlacierJobTier` is faster.
- The previous restoration request has already completed and the object is currently available in Reduced Redundancy Storage mode. In this case, Amazon S3 batch operations update the expiration date of the restored object to match the `ExpirationInDays` specified in this job.

The restore operation fails for the object if any of the following conditions are true:

- The restoration request already in progress has not yet completed and the restoration duration for this job (specified by `ExpirationInDays`) is different than the restoration duration that is specified in the in-progress restoration request.
- The restoration tier for this job (specified by `GlacierJobTier`) is the same or slower than the restoration tier that is specified in the in-progress restoration request.

Limitations

Initiate Restore Object jobs have the following limitations:

- You must create an Initiate Restore Object job in the same Region as the archived objects.
- Amazon S3 batch operations do not support S3 Glacier SELECT.
- Amazon S3 batch operations do not support the Expedited retrieval tier.

Invoking a Lambda Function from Amazon S3 Batch Operations

Amazon S3 batch operations can invoke AWS Lambda functions to perform custom actions on objects that are listed in a manifest. This section describes how to create a Lambda function to use with Amazon S3 batch operations and how to create a job to invoke the function. The Amazon S3 batch operations job uses the `LambdaInvoke` operation to run a Lambda function on each object listed in a manifest.

You can work with Amazon S3 batch operations for Lambda using the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST APIs. For more information about using Lambda, see [Getting Started with AWS Lambda](#) in the *AWS Lambda Developer Guide*.

The following sections explain how you can get started using Amazon S3 batch operations with Lambda.

Topics

- [Using Lambda with Amazon S3 Batch Operations](#) (p. 477)
- [Creating a Lambda Function to Use with Amazon S3 Batch Operations](#) (p. 478)
- [Creating an Amazon S3 Batch Operations Job That Invokes a Lambda Function](#) (p. 481)
- [Providing Task-Level Information in Lambda Manifests](#) (p. 482)

Using Lambda with Amazon S3 Batch Operations

When using Amazon S3 batch operations with AWS Lambda, you must create new Lambda functions specifically for use with Amazon S3 batch operations. You can't reuse existing Amazon S3 event-based functions with Amazon S3 batch operations. Event functions can only receive messages; they don't return messages. The Lambda functions that are used with Amazon S3 batch operations must accept and return messages. For more information about using Lambda with Amazon S3 events, see [Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

You create an Amazon S3 batch operations job that invokes your Lambda function. The job runs the same Lambda function on all of the objects listed in your manifest. You can control what versions of your Lambda function to use while processing the objects in your manifest. Amazon S3 batch operations support unqualified Amazon Resource Names (ARNs), aliases, and specific versions. For more information, see [Introduction to AWS Lambda Versioning](#) in the *AWS Lambda Developer Guide*.

If you provide the Amazon S3 batch operations job with a function ARN that uses an alias or the `$LATEST` qualifier, and you update the version that either of those points to, Amazon S3 batch operations starts calling the new version of your Lambda function. This can be useful when you want to update functionality part of the way through a large job. If you don't want Amazon S3 batch operations to change the version that is used, provide the specific version in the `FunctionARN` parameter when you create your job.

Response and Result Codes

There are two levels of codes that Amazon S3 batch operations expect from Lambda functions. The first is the response code for the entire request, and the second is a per-task result code. The following table contains the response codes.

Response Code	Description
Succeeded	The task completed normally. If you requested a job completion report, the task's result string is included in the report.
TemporaryFailure	The task suffered a temporary failure and will be redriven before the job completes. The result string is ignored. If this is the final redrive, the error message is included in the final report.
PermanentFailure	The task suffered a permanent failure. If you requested a job-completion report, the task is marked as <code>Failed</code> and includes the error message string. Result strings from failed tasks are ignored.

Creating a Lambda Function to Use with Amazon S3 Batch Operations

This section provides example AWS Identity and Access Management (IAM) permissions that you must use with your Lambda function. It also contains an example Lambda function to use with Amazon S3 batch operations. If you have never created a Lambda function before, see [Tutorial: Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

You must create Lambda functions specifically for use with Amazon S3 batch operations. You can't reuse existing Amazon S3 event-based Lambda functions. This is because Lambda functions that are used for Amazon S3 batch operations must accept and return special data fields.

Example IAM Permissions

The following are examples of the IAM permissions that are necessary to use a Lambda function with Amazon S3 batch operations.

Example — Amazon S3 batch operations trust policy

The following is an example of the trust policy that you can use for the execution role. It gives Lambda permission to execute the function invoked by an Amazon S3 batch operations job.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "batchoperations.s3.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```


Example — Lambda IAM policy

The following is an example of an IAM policy that gives Amazon S3 batch operations permission to invoke the Lambda function and read the input manifest.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "BatchOperationsLambdaPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:PutObject",
        "lambda:InvokeFunction"
      ],
      "Resource": "*"
    }
  ]
}
```

Example Request and Response

This section provides request and response examples for the Lambda function.

Example Request

The following is a JSON example of a request for the Lambda function.

```
{
  "invocationSchemaVersion": "1.0",
  "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "job": {
    "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
  },
  "tasks": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "s3Key": "customerImage1.jpg",
      "s3VersionId": "1",
      "s3BucketArn": "arn:aws:s3:us-east-1:0123456788:awsexamplebucket"
    }
  ]
}
```

Example Response

The following is a JSON example of a response for the Lambda function.

```
{
  "invocationSchemaVersion": "1.0",
  "treatMissingKeysAs": "PermanentFailure",
  "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "results": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "resultCode": "Succeeded",
      "resultString": "[\"Mary Major\", \"John Stiles\"]"
    }
  ]
}
```

```
]
}
```

Example Lambda Function for Amazon S3 Batch Operations

The following example Python Lambda function iterates through the manifest, copying and renaming each object.

As the example shows, keys from Amazon S3 batch operations are URL encoded. To use Amazon S3 with other AWS services, it's important that you URL decode the key that is passed from Amazon S3 batch operations.

```
import boto3
import urllib
from botocore.exceptions import ClientError

def lambda_handler(event, context):
    # Instantiate boto client
    s3Client = boto3.client('s3')

    # Parse job parameters from Amazon S3 batch operations
    jobId = event['job']['id']
    invocationId = event['invocationId']
    invocationSchemaVersion = event['invocationSchemaVersion']

    # Prepare results
    results = []

    # Parse Amazon S3 Key, Key Version, and Bucket ARN
    taskId = event['tasks'][0]['taskId']
    s3Key = urllib.unquote(event['tasks'][0]['s3Key']).decode('utf8')
    s3VersionId = event['tasks'][0]['s3VersionId']
    s3BucketArn = event['tasks'][0]['s3BucketArn']
    s3Bucket = s3BucketArn.split(':::')[1]

    # Construct CopySource with VersionId
    copySrc = {'Bucket': s3Bucket, 'Key': s3Key}
    if s3VersionId is not None:
        copySrc['VersionId'] = s3VersionId

    # Copy object to new bucket with new key name
    try:
        # Prepare result code and string
        resultCode = None
        resultString = None

        # Construct New Key
        newKey = rename_key(s3Key)
        newBucket = 'destination-bucket-name'

        # Copy Object to New Bucket
        response = s3Client.copy_object(
            CopySource = copySrc,
            Bucket = newBucket,
            Key = newKey
        )

        # Mark as succeeded
        resultCode = 'Succeeded'
        resultString = str(response)
    except ClientError as e:
        # If request timed out, mark as a temp failure
        # and Amazon S3 batch operations will make the task for retry. If
        # any other exceptions are received, mark as permanent failure.
```

```

        errorCode = e.response['Error']['Code']
        errorMessage = e.response['Error']['Message']
        if errorCode == 'RequestTimeout':
            resultCode = 'TemporaryFailure'
            resultString = 'Retry request to Amazon S3 due to timeout.'
        else:
            resultCode = 'PermanentFailure'
            resultString = '{}: {}'.format(errorCode, errorMessage)
    except Exception as e:
        # Catch all exceptions to permanently fail the task
        resultCode = 'PermanentFailure'
        resultString = 'Exception: {}'.format(e.message)
    finally:
        results.append({
            'taskId': taskId,
            'resultCode': resultCode,
            'resultString': resultString
        })

    return {
        'invocationSchemaVersion': invocationSchemaVersion,
        'treatMissingKeysAs': 'PermanentFailure',
        'invocationId': invocationId,
        'results': results
    }

def rename_key(s3Key):
    # Rename the key by adding additional suffix
    return s3Key + '_new_suffix'

```

Creating an Amazon S3 Batch Operations Job That Invokes a Lambda Function

When creating an Amazon S3 batch operations job to invoke a Lambda function, you must provide the following:

- The ARN of your Lambda function (which might include the function alias or a specific version number)
- An IAM role with permission to invoke the function
- The action parameter `LambdaInvokeFunction`

For more information about creating an Amazon S3 batch operations job, see [Creating an Amazon S3 Batch Operations Job \(p. 470\)](#) and [Operations \(p. 475\)](#).

The following example creates an Amazon S3 batch operations job that invokes a Lambda function using the AWS CLI.

```

aws s3control create-job
  --account-id <AccountID>
  --operation '{"LambdaInvoke": { "FunctionArn":
    "arn:aws:lambda:Region:AccountID:function:LambdaFunctionName" } }'
  --manifest '{"Spec":{"Format":"S3BatchOperations_CSV_20180820","Fields":
    ["Bucket","Key"]},"Location":
    {"ObjectArn":"arn:aws:s3:::ManifestLocation","ETag":"ManifestETag"}}'
  --report
    '{"Bucket":"arn:aws:s3:::awsexamplebucket","Format":"Report_CSV_20180820","Enabled":true,"Prefix":"Rep
  --priority 2
  --role-arn arn:aws:iam::AccountID:role/BatchOperationsRole
  --region Region
  --description "Lambda Function"

```

Providing Task-Level Information in Lambda Manifests

When you use AWS Lambda functions with Amazon S3 batch operations, you might want additional data to accompany each task/key that is operated on. For example, you might want to have both a source object key and new object key provided. Your Lambda function could then copy the source key to a new S3 bucket under a new name. By default, Amazon S3 batch operations let you specify only the destination bucket and a list of source keys in the input manifest to your job. The following describes how you can include additional data in your manifest so that you can run more complex Lambda functions.

To specify per-key parameters in your Amazon S3 batch operations manifest to use in your Lambda function's code, use the following URL-encoded JSON format. The `key` field is passed to your Lambda function as if it were an Amazon S3 object key. But it can be interpreted by the Lambda function to contain other values or multiple keys, as shown following.

Note

The maximum number of characters for the `key` field in the manifest is 1,024.

Example — Manifest substituting the "Amazon S3 keys" with JSON strings

The URL-encoded version must be provided to Amazon S3 batch operations.

```
my-bucket,{"origKey": "object1key", "newKey": "newObject1Key"}
my-bucket,{"origKey": "object2key", "newKey": "newObject2Key"}
my-bucket,{"origKey": "object3key", "newKey": "newObject3Key"}
```

Example — Manifest URL-encoded

This URL-encoded version must be provided to Amazon S3 batch operations. The non-URL-encoded version does not work.

```
my-bucket,%7B%22origKey%22%3A%20%22object1key%22%2C%20%22newKey%22%3A%20%22newObject1Key%22%7D
my-bucket,%7B%22origKey%22%3A%20%22object2key%22%2C%20%22newKey%22%3A%20%22newObject2Key%22%7D
my-bucket,%7B%22origKey%22%3A%20%22object3key%22%2C%20%22newKey%22%3A%20%22newObject3Key%22%7D
```

Example — Lambda function with manifest format writing results to the job report

```
import json
from urllib.parse import unquote_plus

# This example Lambda function shows how to parse JSON that is encoded into the Amazon S3
# batch
# operations manifest containing lines like this:
#
# bucket,encoded-json
# bucket,encoded-json
# bucket,encoded-json
#
# For example, if we wanted to send the following JSON to this Lambda function:
#
# bucket,{"origKey": "object1key", "newKey": "newObject1Key"}
# bucket,{"origKey": "object2key", "newKey": "newObject2Key"}
# bucket,{"origKey": "object3key", "newKey": "newObject3Key"}
```

```
#
# We would simply URL-encode the JSON like this to create the real manifest to create a
# batch
# operations job with:
#
# my-bucket,%7B%22origKey%22%3A%20%22object1key%22%2C%20%22newKey%22%3A%20%22newObject1Key%22%7D
# my-bucket,%7B%22origKey%22%3A%20%22object2key%22%2C%20%22newKey%22%3A%20%22newObject2Key%22%7D
# my-bucket,%7B%22origKey%22%3A%20%22object3key%22%2C%20%22newKey%22%3A%20%22newObject3Key%22%7D
#
def lambda_handler(event, context):
    # Parse job parameters from Amazon S3 batch operations
    jobId = event['job']['id']
    invocationId = event['invocationId']
    invocationSchemaVersion = event['invocationSchemaVersion']

    # Prepare results
    results = []

    # S3 batch operations currently only passes a single task at a time in the array of
    # tasks.
    task = event['tasks'][0]

    # Extract the task values we might want to use
    taskId = task['taskId']
    s3Key = task['s3Key']
    s3VersionId = task['s3VersionId']
    s3BucketArn = task['s3BucketArn']
    s3BucketName = s3BucketArn.split(':')[3]

    try:
        # Assume it will succeed for now
        resultCode = 'Succeeded'
        resultString = ''

        # Decode the JSON string that was encoded into the S3 Key value and convert the
        # resulting string into a JSON structure.
        s3Key_decoded = unquote_plus(s3Key)
        keyJson = json.loads(s3Key_decoded)

        # Extract some values from the JSON that we might want to operate on. In this
        # example
        # we won't do anything except return the concatenated string as a fake result.
        newKey = keyJson['newKey']
        origKey = keyJson['origKey']
        resultString = origKey + " --> " + newKey

    except Exception as e:
        # If we run into any exceptions, fail this task so batch operations does retry it
        # and
        # return the exception string so we can see the failure message in the final report
        # created by batch operations.
        resultCode = 'PermanentFailure'
        resultString = 'Exception: {}'.format(e)
    finally:
        # Send back the results for this task.
        results.append({
            'taskId': taskId,
            'resultCode': resultCode,
            'resultString': resultString
        })

    return {
        'invocationSchemaVersion': invocationSchemaVersion,
```

```
'treatMissingKeysAs': 'PermanentFailure',  
'invocationId': invocationId,  
'results': results  
}
```

Put Object ACL

The Put Object Acl operation replaces the Amazon S3 access control lists (ACLs) for each object that is listed in the manifest. Using ACLs, you can define who can access an object and what actions they can perform.

Amazon S3 batch operations support custom ACLs that you define and canned ACLs that Amazon S3 provides with a predefined set of access permissions.

If the objects in your manifest are in a versioned bucket, you can apply the ACLs to specific versions of each object. You do this by specifying a version ID for each object in the manifest. If you don't include a version ID for any object, then Amazon S3 batch operations applies the ACL to the latest version of the object.

Note

If you want to limit public access to all objects in a bucket, you should use Amazon S3 block public access instead of Amazon S3 batch operations. Block public access can limit public access on a per-bucket or account-wide basis with a single, simple operation that takes effect quickly. This makes it a better choice when your goal is to control public access to all objects in a bucket or account. Use Amazon S3 batch operations when you need to apply a customized ACL to each object in the manifest. For more information about Amazon S3 block public access, see [Using Amazon S3 Block Public Access \(p. 414\)](#).

Restrictions and Limitations

- The role that you specify to run the Put Object Acl job must have permissions to perform the underlying Amazon S3 PUT Object acl operation. For more information about the permissions required, see [PUT Object acl](#) in the *Amazon Simple Storage Service API Reference*.
- Amazon S3 batch operations use the Amazon S3 PUT Object acl operation to apply the specified ACL to each object in the manifest. Therefore, all restrictions and limitations that apply to the underlying PUT Object acl operation also apply to Amazon S3 batch operations Put Object Acl jobs. For more information, see the [Related Resources \(p. 484\)](#) section of this page.

Related Resources

- [Managing Access with ACLs \(p. 403\)](#)
- [GET Object ACL](#) in the *Amazon Simple Storage Service API Reference*

Put Object Tagging

The Put Object Tagging operation replaces the Amazon S3 object tags of each object listed in the manifest. An Amazon S3 object tag is a key-value pair of strings that you can use to store metadata about an object.

To create a Put Object Tagging job, you provide a set of tags that you want to apply. Amazon S3 batch operations apply the same set of tags to each object. The tag set that you provide replaces whatever tag sets are already associated with the objects in the manifest. Amazon S3 batch operations do not support adding tags to objects while leaving the existing tags in place.

If the objects in your manifest are in a versioned bucket, you can apply the tag set to specific versions of each object. You do this by specifying a version ID for each object in the manifest. If you don't include a version ID for any object, then Amazon S3 batch operations will apply the tag set to the latest version of each object.

Restrictions and Limitations

- The role that you specify to run the Put Object Tagging job must have permissions to perform the underlying Amazon S3 PUT Object tagging operation. For more information about the permissions required, see [PUT Object tagging](#) in the *Amazon Simple Storage Service API Reference*.
- Amazon S3 batch operations use the Amazon S3 PUT Object tagging operation to apply tags to each object in the manifest. Therefore, all restrictions and limitations that apply to the underlying PUT Object tagging operation also apply to Amazon S3 batch operations Put Object Tagging jobs. For more information, see the [Related Resources \(p. 485\)](#) section of this page.

Related Resources

- [Object Tagging \(p. 110\)](#)
- [GET Object tagging](#) in the *Amazon Simple Storage Service API Reference*
- [PUT Object tagging](#) in the *Amazon Simple Storage Service API Reference*

Managing Batch Operations Jobs

Amazon S3 provides a robust set of tools to help you manage your batch operations jobs after you have created them. This section describes the operations you can use to manage your jobs. You can perform all of the operations listed in this section using the AWS Management Console, AWS CLI, AWS SDKs, or REST APIs.

Topics

- [Listing Jobs \(p. 485\)](#)
- [Viewing Job Details \(p. 485\)](#)
- [Assigning Job Priority \(p. 486\)](#)
- [Job Status \(p. 486\)](#)
- [Tracking Job Failure \(p. 488\)](#)
- [Notifications and Logging \(p. 488\)](#)
- [Completion Reports \(p. 489\)](#)

Listing Jobs

You can retrieve a list of your batch operations jobs. The list includes jobs that haven't yet finished and jobs that finished within the last 90 days. The job list includes information for each job, such as its ID, description, priority, current status, and the number of tasks that have succeeded and failed. You can filter your job list by status. When you retrieve a job list through the console, you can also search your jobs by description or ID and filter them by AWS Region.

Viewing Job Details

If you want more information about a job than you can retrieve by listing jobs, you can view all of the details for a single job. In addition to the information returned in a job list, a single job's details include other items. This information includes the operation parameters, details about the manifest, information

about the completion report (if you configured one when you created the job), and the Amazon Resource Name (ARN) of the user role that you assigned to run the job. By viewing an individual job's details, you can access a job's entire configuration.

Assigning Job Priority

You can assign each job a numeric priority, which can be any positive integer. Amazon S3 batch operations prioritize jobs according to the assigned priority. Jobs with a higher priority (or a higher numeric value for the priority parameter) are evaluated first. Priority is determined in descending order. For example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1.

You can change a job's priority while it is running. If you submit a new job with a higher priority while a job is running, the lower-priority job can pause to allow the higher-priority job to run.

Note

Amazon S3 batch operations honor job priorities on a best-effort basis. Although jobs with higher priorities generally take precedence over jobs with lower priorities, Amazon S3 does not guarantee strict ordering of jobs.

Job Status

After you create a job, it progresses through a series of statuses. The following table describes the statuses that jobs can have and the possible transitions between job statuses.

Status	Description	Transitions
New	A job begins in the <code>New</code> state when you create it.	A job automatically moves to the <code>Preparing</code> state when Amazon S3 begins processing the manifest object.
Preparing	Amazon S3 is processing the manifest object and other job parameters to set up and run the job.	<p>A job automatically moves to the <code>Ready</code> state after Amazon S3 finishes processing the manifest and other parameters. It is then ready to begin running the specified operation on the objects listed in the manifest.</p> <p>If the job requires confirmation before running, such as when you create a job using the Amazon S3 console, then the job transitions from <code>Preparing</code> to <code>Suspended</code>. It remains in the <code>Suspended</code> state until you confirm that you want to run it.</p>
Suspended	The job requires confirmation, but you have not yet confirmed that you want to run it. Only jobs that you create using the Amazon S3 console require confirmation. A job that is created using the console enters the <code>Suspended</code> state	After you confirm that you want to run the job, its status changes to <code>Ready</code> .

Status	Description	Transitions
	immediately after <code>Preparing</code> . After you confirm that you want to run the job and the job becomes <code>Ready</code> , it never returns to the <code>Suspended</code> state.	
<code>Ready</code>	Amazon S3 is ready to begin running the requested object operations.	A job automatically moves to <code>Active</code> when Amazon S3 begins to run it. The amount of time that a job remains in the <code>Ready</code> state depends on whether you have higher-priority jobs running already and how long those jobs take to complete.
<code>Active</code>	Amazon S3 is executing the requested operation on the objects listed in the manifest. While a job is <code>Active</code> , you can monitor its progress using the Amazon S3 console or the <code>DescribeJob</code> operation through the REST API, AWS CLI, or AWS SDKs.	A job moves out of the <code>Active</code> state when it is no longer running operations on objects. This can happen automatically, such as when a job completes successfully or fails. Or it can occur as a result of user actions, such as canceling a job. The state that the job moves to depends on the reason for the transition.
<code>Pausing</code>	The job is transitioning to <code>Paused</code> from another state.	A job automatically moves to <code>Paused</code> when the <code>Pausing</code> stage is finished.
<code>Paused</code>	A job can become <code>Paused</code> if you submit another job with a higher priority while the current job is running.	A <code>Paused</code> job automatically returns to <code>Active</code> after any higher-priority jobs that are blocking the job's execution complete, fail, or are suspended.
<code>Complete</code>	The job has finished executing the requested operation on all objects in the manifest. The operation might have succeeded or failed for each object. If you configured the job to generate a completion report, the report is available as soon as the job is <code>Complete</code> .	<code>Complete</code> is a terminal state. Once a job reaches <code>Complete</code> , it does not transition to any other state.
<code>Cancelling</code>	The job is transitioning to the <code>Cancelled</code> state.	A job automatically moves to <code>Cancelled</code> when the <code>Cancelling</code> stage is finished.

Status	Description	Transitions
Cancelled	You requested that the job be cancelled, and Amazon S3 batch operations has successfully cancelled the job. The job will not submit any new requests to Amazon S3.	Cancelled is a terminal state. After a job reaches Cancelled, it will not transition to any other state.
Failing	The job is transitioning to the Failed state.	A job automatically moves to Failed once the Failing stage is finished.
Failed	The job has failed and is no longer running. For more information about job failures, see Tracking Job Failure (p. 488).	Failed is a terminal state. After a job reaches Failed, it will not transition to any other state.

Tracking Job Failure

If a batch operations job encounters a problem that prevents it from running successfully, such as not being able to read the specified manifest, the job fails. When a job fails, it generates one or more failure codes or failure reasons. Amazon S3 batch operations store the failure codes and reasons with the job so that you can view them by requesting the job's details. If you requested a completion report for the job, the failure codes and reasons also appear there.

To prevent jobs from running a large number of unsuccessful operations, Amazon S3 imposes a task-failure threshold on every batch operations job. When a job has executed at least 1,000 tasks, Amazon S3 monitors the task failure rate. If, at any point, the failure rate (the number of tasks that have failed as a proportion of the total number of tasks that have executed) exceeds 50 percent, then the job fails. If your job fails because it exceeded the task-failure threshold, you can identify the cause of the failures. For example, you might have accidentally included some objects in the manifest that don't exist in the specified bucket. After fixing the errors, you can resubmit the job.

Note

Amazon S3 batch operations operate asynchronously and the tasks don't necessarily execute in the order that the objects are listed in the manifest. Therefore, you can't use the manifest ordering to determine which objects' tasks succeeded and which ones failed. Instead, you can examine the job's completion report (if you requested one) or view your AWS CloudTrail event logs to help determine the source of the failures.

Notifications and Logging

In addition to requesting completion reports, you can also capture, review, and audit batch operations activity using Amazon S3 events. As a job progresses, it emits events that you can capture using AWS CloudTrail, Amazon Simple Notification Service (Amazon SNS), and Amazon Simple Queue Service (Amazon SQS). Because batch operations use existing Amazon S3 APIs to perform tasks, those tasks also emit the same events that they would if you called them directly. Thus, you can track and record the progress of your job and all of its tasks using the same notification, logging, and auditing tools and processes that you already use with Amazon S3. For more information about Amazon S3 events, see [Configuring Amazon S3 Event Notifications](#) (p. 530).

Completion Reports

When you create a job, you can request a completion report. Then as long as Amazon S3 batch operations successfully invoke at least one task, Amazon S3 generates a completion report after it finishes running tasks, fails, or is canceled. You can configure the completion report to include all tasks or only failed tasks.

The completion report includes the job configuration and status and information for each task, including the object key and version, status, error codes, and descriptions of any errors. If you don't configure a completion report, you can still monitor and audit your job and its tasks using CloudTrail, Amazon CloudWatch, Amazon SNS, and Amazon SQS. However, completion reports provide an easy way to view the results of your tasks in a consolidated format with no additional setup required. For an example of a completion report, see [Amazon S3 Batch Operations Completion Report Examples \(p. 489\)](#).

Amazon S3 Batch Operations Examples

Topics

- [Amazon S3 Batch Operations Completion Report Examples \(p. 489\)](#)
- [Cross-Account Copy Examples for Amazon S3 Batch Operations \(p. 491\)](#)
- [AWS CLI Examples for Amazon S3 Batch Operations \(p. 495\)](#)
- [Java Examples for Amazon S3 Batch Operations \(p. 498\)](#)

Amazon S3 Batch Operations Completion Report Examples

When you create an Amazon S3 batch operations job, you can request a completion report for all tasks or just for failed tasks. As long as at least one task has been invoked successfully, Amazon S3 Batch Operations generates a report for jobs that have completed, failed, or have been cancelled.

The completion report contains additional information for each task, including the object key name and version, status, error codes, and descriptions of any errors. The description of errors for each failed task can be used to diagnose issues during job creation such as permissions.

Example Top-Level Manifest Result File

The top-level `manifest.json` file contains the locations of each succeeded report and (if the job had any failures) the location of failed reports, as shown in the following example.

```
{
  "Format": "Report_CSV_20180820",
  "ReportCreationDate": "2019-04-05T17:48:39.725Z",
  "Results": [
    {
      "TaskExecutionStatus": "succeeded",
      "Bucket": "my-job-reports",
      "MD5Checksum": "83b1c4cbe93fc893f54053697e10fd6e",
      "Key": "job-f8fb9d89-a3aa-461d-bddc-ea6a1b131955/results/6217b0fab0de85c408b4be96aeaca9b195a7daa5.csv"
    },
    {
      "TaskExecutionStatus": "failed",
      "Bucket": "my-job-reports",
      "MD5Checksum": "22ee037f3515975f7719699e5c416eaa",

```

```

    "Key": "job-f8fb9d89-a3aa-461d-bddc-ea6a1b131955/results/
b2ddad417e94331e9f37b44f1faf8c7ed5873f2e.csv"
  },
  ],
  "ReportSchema": "Bucket, Key, VersionId, TaskStatus, ErrorCode, HTTPStatusCode,
ResultMessage"
}

```

Example Failed Tasks Reports

Failed tasks reports contain the Bucket, Key, VersionId, TaskStatus, ErrorCode, HTTPStatusCode, and ResultMessage for all failed tasks.

The following example report shows a case in which the AWS Lambda function timed out, causing failures to exceed the failure threshold. It was then marked as a PermanentFailure.

```

awsexamplebucket,image_14975,,failed,200,PermanentFailure,"Lambda returned function error:
{"errorMessage":"2019-04-05T17:35:21.155Z 2845ca0d-38d9-4c4b-abcfc379dc749c452 Task
timed out after 3.00 seconds"}"
awsexamplebucket,image_15897,,failed,200,PermanentFailure,"Lambda returned function error:
{"errorMessage":"2019-04-05T17:35:29.610Z 2d0a330b-de9b-425f-b511-29232fde5fe4 Task
timed out after 3.00 seconds"}"
awsexamplebucket,image_14819,,failed,200,PermanentFailure,"Lambda returned function error:
{"errorMessage":"2019-04-05T17:35:22.362Z fcf5efde-74d4-4e6d-b37a-c7f18827f551 Task
timed out after 3.00 seconds"}"
awsexamplebucket,image_15930,,failed,200,PermanentFailure,"Lambda returned function error:
{"errorMessage":"2019-04-05T17:35:29.809Z 3dd5b57c-4a4a-48aa-8a35-cbf027b7957e Task
timed out after 3.00 seconds"}"
awsexamplebucket,image_17644,,failed,200,PermanentFailure,"Lambda returned function error:
{"errorMessage":"2019-04-05T17:35:46.025Z 10a764e4-2b26-4d8c-9056-1e1072b4723f Task
timed out after 3.00 seconds"}"
awsexamplebucket,image_17398,,failed,200,PermanentFailure,"Lambda returned function error:
{"errorMessage":"2019-04-05T17:35:44.661Z 1e306352-4c54-4eba-ae8-4d02f8c0235c Task
timed out after 3.00 seconds"}"

```

Example Succeeded Tasks Report

Succeeded tasks reports contain the Bucket, Key, VersionId, TaskStatus, ErrorCode, HTTPStatusCode, and ResultMessage for the completed tasks.

In the following example, the Lambda function successfully copied the Amazon S3 object to another bucket. The returned Amazon S3 response is passed back to Amazon S3 batch operations and is then written into the final completion report.

```

awsexamplebucket,image_17775,,succeeded,200,,{"u'CopySourceVersionId':
'xVR78haVKlRnurYofbTfYr3ufYbktF8h', u'CopyObjectResult': {u'LastModified':
datetime.datetime(2019, 4, 5, 17, 35, 39, tzinfo=tzlocal()), u'ETag':
'"fe66f4390c50f29798f040d7aae72784"'}, 'ResponseMetadata': {'HTTPStatusCode':
200, 'RetryAttempts': 0, 'HostId': 'nXNaClIMxEJzWNmeMNQV2KpjbacJLn0OGoxWZpuVOFS/
iQYWxb3QtTvzX9SVfx2lA3oTKLwImKw=', 'RequestId': '3ED5852152014362', 'HTTPHeaders':
{'content-length': '234', 'x-amz-id-2': 'nXNaClIMxEJzWNmeMNQV2KpjbacJLn0OGoxWZpuVOFS/
iQYWxb3QtTvzX9SVfx2lA3oTKLwImKw=', 'x-amz-copy-source-version-id':
'xVR78haVKlRnurYofbTfYr3ufYbktF8h', 'server': 'AmazonS3', 'x-amz-request-id':
'3ED5852152014362', 'date': 'Fri, 05 Apr 2019 17:35:39 GMT', 'content-type': 'application/
xml'}}}
awsexamplebucket,image_17763,,succeeded,200,,{"u'CopySourceVersionId':
'6HjOUSim4Wj6BTcbxToXW44pSZ.40pwq', u'CopyObjectResult': {u'LastModified':
datetime.datetime(2019, 4, 5, 17, 35, 39, tzinfo=tzlocal()),
u'ETag': '"fe66f4390c50f29798f040d7aae72784"'}, 'ResponseMetadata':
{'HTTPStatusCode': 200, 'RetryAttempts': 0, 'HostId': 'GiCZNYr8Lhd/
Thyk6beTRP96IGZk2sYxujLe13TuuLpq6U2RD3we0YoluuIdm1PRvkMwnEW1aFc=', 'RequestId':
'1BC9F5B1B95D7000', 'HTTPHeaders': {'content-length': '234', 'x-amz-id-2': 'GiCZNYr8Lhd/

```

```
Thyk6beTRP96IGZk2sYxujLe13TuuLpq6U2RD3we0YoluuIdm1PRvkMwnEW1aFc=', 'x-amz-copy-source-  
version-id': '6HjOUsim4Wj6BTcbxToXW44pSZ.40pwq', 'server': 'AmazonS3', 'x-amz-request-id':  
'1BC9F5B1B95D7000', 'date': 'Fri, 05 Apr 2019 17:35:39 GMT', 'content-type': 'application/  
xml'}}}"  
awsexamplebucket,image_17860,,succeeded,200,,{"u'CopySourceVersionId':  
'm.MDD0g_QsUnYZ8TBzVFrP.TmjN8PJyX', u'CopyObjectResult': {u'LastModified':  
datetime.datetime(2019, 4, 5, 17, 35, 40, tzinfo=tzlocal()), u'ETag':  
'"fe66f4390c50f29798f040d7aae72784"'}, 'ResponseMetadata': {'HTTPStatusCode':  
200, 'RetryAttempts': 0, 'HostId': 'F9ooZOgpE5g9sNgBZxjdiPHqB4+0DNWgj3qbsir  
+sKai4fv7rQEcF2fBN1VeeFc2WH45a9ygb2g=', 'RequestId': '8D9CA56A56813DF3', 'HTTPHeaders':  
{'content-length': '234', 'x-amz-id-2': 'F9ooZOgpE5g9sNgBZxjdiPHqB4+0DNWgj3qbsir  
+sKai4fv7rQEcF2fBN1VeeFc2WH45a9ygb2g=', 'x-amz-copy-source-version-id':  
'm.MDD0g_QsUnYZ8TBzVFrP.TmjN8PJyX', 'server': 'AmazonS3', 'x-amz-request-id':  
'8D9CA56A56813DF3', 'date': 'Fri, 05 Apr 2019 17:35:40 GMT', 'content-type': 'application/  
xml'}}}"
```

Cross-Account Copy Examples for Amazon S3 Batch Operations

You can use Amazon S3 batch operations to create a PUT copy job to copy objects to a different AWS account (the *destination account*). When doing this, you can use Amazon S3 inventory to deliver the inventory report to the destination account for use during job creation or you can use a comma-separated values (CSV) manifest in the source or destination account. The following sections explain how to store and use a manifest that is in a different AWS account.

Topics

- [Using an Inventory Report Delivered to the Destination AWS Account \(p. 491\)](#)
- [Using a CSV Manifest Stored in the Source AWS Account \(p. 493\)](#)

Using an Inventory Report Delivered to the Destination AWS Account

The Amazon S3 inventory generates inventories of the objects in a bucket. The resulting list is published to an output file. The bucket that is inventoried is called the *source bucket*, and the bucket where the inventory report file is stored is called the *destination bucket*. The Amazon S3 inventory report can be configured to deliver the inventory report to another AWS account. This allows Amazon S3 batch operations to read the inventory report when the job is created in the destination AWS account. For more information about Amazon S3 inventory source and destination buckets, see [How Do I Set Up Amazon S3 Inventory? \(p. 423\)](#).

The easiest way to set up an inventory is by using the AWS Management Console, but you can also use the REST API, AWS CLI, or AWS SDKs.

In the following console procedure, you set up permissions for an Amazon S3 batch operations job to copy objects from a source account to a destination account, with the inventory report stored in the destination AWS account.

To set up Amazon S3 inventory for source and destination buckets that are owned by different AWS accounts

1. Choose a destination bucket to store the inventory report in.

Decide on a destination manifest bucket for storing the inventory report. In this procedure, the *destination account* is the account that owns both the destination manifest bucket and the bucket that the objects are copied to.

2. Configure an inventory to list the objects in a source bucket and publish the list to the destination manifest bucket.

Configure an inventory list for a source bucket. When you do this, you specify the destination bucket where you want the list to be stored. The inventory report for the source bucket is published to the destination bucket. In this procedure, the *source account* is the account that owns the source bucket.

For information about how to use the console to configure an inventory, see [How Do I Configure Amazon S3 Inventory?](#) in the *Amazon Simple Storage Service Console User Guide*.

Choose **CSV** for the output format.

When you enter information for the destination bucket, choose **Buckets in another account**. Then enter the name of the destination manifest bucket. Optionally, you can enter the account ID of the destination account.

After the inventory configuration is saved, the console displays a message similar to the following:

Amazon S3 could not create a bucket policy on the destination bucket. Ask the destination bucket owner to add the following bucket policy to allow Amazon S3 to place data in that bucket.

The console then displays a bucket policy that you can use for the destination bucket.

3. Copy the destination bucket policy that appears on the console.
4. In the destination account, add the copied bucket policy to the destination manifest bucket where the inventory report is stored.
5. Create a role in the destination account that is based on the Amazon S3 batch operations trust policy. For more information about the trust policy, see [Trust Policy \(p. 472\)](#). For more information about creating a role, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.

Enter a name for the role (the example role uses the name `BatchOperationsDestinationRoleCOPY`). Choose the **S3** service, and then choose the **S3 bucket Batch Operations** use case, which applies the trust policy to the role.

Then choose **Create policy** to attach the following policy to the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowBatchOperationsDestinationObjectCOPY",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectVersionAcl",
        "s3:PutObjectAcl",
        "s3:PutObjectVersionTagging",
        "s3:PutObjectTagging",
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetObjectAcl",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws:s3:::ObjectDestinationBucket/*",
        "arn:aws:s3:::ObjectSourceBucket/*",
        "arn:aws:s3:::ObjectDestinationManifestBucket/*"
      ]
    }
  ]
}
```

```
}
}
```

The role uses the policy to grant `batchoperations.s3.amazonaws.com` permission to read the manifest in the destination bucket. It also grants permissions to GET objects, access control lists (ACLs), tags, and versions in the source object bucket. And it grants permissions to PUT objects, ACLs, tags, and versions into the destination object bucket.

6. In the source account, create a bucket policy for the source bucket that grants the role that you created in the previous step to GET objects, ACLs, tags, and versions in the source bucket. This step allows Amazon S3 batch operations to get objects from the source bucket through the trusted role.

The following is an example of the bucket policy for the source account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowBatchOperationsSourceObjectCOPY",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::DestinationAccountNumber:role/BatchOperationsDestinationRoleCOPY"
      },
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetObjectAcl",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": "arn:aws:s3::ObjectSourceBucket/*"
    }
  ]
}
```

7. After the inventory report is available, create an Amazon S3 batch operations PUT object copy job in the destination account, choosing the inventory report from the destination manifest bucket. You need the ARN for the role that you created in the destination account.

For general information about creating a job, see [Creating an Amazon S3 Batch Operations Job \(p. 470\)](#). For information about creating a job using the console, see [Creating an Amazon S3 Batch Operations Job](#) in the *Amazon Simple Storage Service Console User Guide*.

Using a CSV Manifest Stored in the Source AWS Account

You can use a CSV file that is stored in a different AWS account as a manifest for an Amazon S3 batch operations job.

The following procedure shows how to set up permissions when using an Amazon S3 batch operations job to copy objects from a source account to a destination account with the CSV manifest file stored in the source account.

To set up a CSV manifest stored in a different AWS account

1. Create a role in the destination account that is based on the Amazon S3 batch operations trust policy. In this procedure, the *destination account* is the account that the objects are being copied to.

For more information about the trust policy, see [Trust Policy \(p. 472\)](#). For more information about creating a role, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.

If you create the role using the console, enter a name for the role (the example role uses the name `BatchOperationsDestinationRoleCOPY`). Choose the **S3** service, and then choose the **S3 bucket Batch Operations** use case, which applies the trust policy to the role. Then choose **Create policy** to attach the following policy to the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowBatchOperationsDestinationObjectCOPY",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectVersionAcl",
        "s3:PutObjectAcl",
        "s3:PutObjectVersionTagging",
        "s3:PutObjectTagging",
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetObjectAcl",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws:s3:::ObjectDestinationBucket/*",
        "arn:aws:s3:::ObjectSourceBucket/*",
        "arn:aws:s3:::ObjectSourceManifestBucket/*"
      ]
    }
  ]
}
```

Using the policy, the role grants `batchoperations.s3.amazonaws.com` permission to read the manifest in the source manifest bucket. It grants permissions to GET objects, ACLs, tags, and versions in the source object bucket. It also grants permissions to PUT objects, ACLs, tags, and versions into the destination object bucket.

2. In the source account, create a bucket policy for the bucket that contains the manifest to grant the role that you created in the previous step to GET objects and versions in the source manifest bucket. This step allows Amazon S3 batch operations to read the manifest using the trusted role. Apply the bucket policy to the bucket that contains the manifest.

The following is an example of the bucket policy to apply to the source manifest bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowBatchOperationsSourceManifestRead",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::DestinationAccountNumber:user/ConsoleUserCreatingJob",
          "arn:aws:iam::DestinationAccountNumber:role/BatchOperationsDestinationRoleCOPY"
        ]
      }
    }
  ]
}
```



```

        ],
        "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion"
        ],
        "Resource": "arn:aws:s3:::ObjectSourceManifestBucket/*"
    }
]
}

```

This policy also grants permissions to allow a console user who is creating a job in the destination account the same permissions in the source manifest bucket through the same bucket policy.

3. In the source account, create a bucket policy for the source bucket that grants the role you created to GET objects, ACLs, tags, and versions in the source object bucket. Amazon S3 batch operations can then get objects from the source bucket through the trusted role.

The following is an example of the bucket policy for the bucket that contains the source objects.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowBatchOperationsSourceObjectCOPY",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::DestinationAccountNumber:role/BatchOperationsDestinationRoleCOPY"
      },
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetObjectAcl",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": "arn:aws:s3:::ObjectSourceBucket/*"
    }
  ]
}

```

4. Create an Amazon S3 batch operations job in the destination account. You need the ARN for the role that you created in the destination account.

For general information about creating a job, see [Creating an Amazon S3 Batch Operations Job](#) (p. 470). For information about creating a job using the console, see [Creating an Amazon S3 Batch Operations Job](#) in the *Amazon Simple Storage Service Console User Guide*.

AWS CLI Examples for Amazon S3 Batch Operations

The following example creates an Amazon S3 batch operations S3PutObjectTagging job using the AWS Command Line Interface (AWS CLI).

1. Create an IAM role and assign permissions. This role grants Amazon S3 permission to add object tags, for which you create a job in the next step.
 - a. Create an IAM role as follows:

```
aws iam create-role \  
--role-name S3BatchJobRole \  
--assume-role-policy-document '{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "batchoperations.s3.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}'
```

Record the role's Amazon Resource Name (ARN). You need the ARN when you create a job.

- b. Create an IAM policy with permissions and attach it to the IAM role that you created in the previous step. For more information about permissions, see [Granting Permissions for Amazon S3 Batch Operations \(p. 471\)](#).

```
aws iam put-role-policy \  
--role-name S3BatchJobRole \  
--policy-name PutObjectTaggingBatchJobPolicy \  
--policy-document '{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:PutObjectTagging",  
        "s3:PutObjectVersionTagging"  
      ],  
      "Resource": "arn:aws:s3:::{{TargetResource}}/*"  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:GetObject",  
        "s3:GetObjectVersion",  
        "s3:GetBucketLocation"  
      ],  
      "Resource": [  
        "arn:aws:s3:::{{ManifestBucket}}/*"  
      ]  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:PutObject",  
        "s3:GetBucketLocation"  
      ],  
      "Resource": [  
        "arn:aws:s3:::{{ReportBucket}}/*"  
      ]  
    }  
  ]  
}'
```

2. Create an S3PutObjectTagging job. The manifest.csv file provides a list of bucket and object key values. The job applies the specified tags to objects identified in the manifest. The ETag is the ETag of the manifest.csv object, which you can get from the Amazon S3 console. The request

specifies the `no-confirmation-required` parameter. Therefore, Amazon S3 makes the job eligible for execution without you having to confirm it using the `update-job-status` command.

```
aws s3control create-job \
  --region us-west-2 \
  --account-id acct-id \
  --operation '{"S3PutObjectTagging": { "TagSet": [{"Key":"keyOne",
"Value":"ValueOne"}] }}' \
  --manifest '{"Spec":{"Format":"S3BatchOperations_CSV_20180820","Fields":
[ "Bucket", "Key" ]}, "Location":{"ObjectArn":"arn:aws:s3:::my_manifests/
manifest.csv", "ETag":"60e460c9d1046e73f7dde5043ac3ae85"}}' \
  --report '{"Bucket":"arn:aws:s3:::bucket-where-
completion-report-goes", "Prefix":"final-reports",
"Format":"Report_CSV_20180820", "Enabled":true, "ReportScope":"AllTasks"}' \
  --priority 42 \
  --role-arn IAM-role \
  --client-request-token $(uuidgen) \
  --description "job Description" \
  --no-confirmation-required
```

In response, Amazon S3 returns a job ID (for example, `00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c`). You need the ID in the next commands.

3. Get the job description.

```
aws s3control describe-job \
  --region us-west-2 \
  --account-id acct-id \
  --job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c
```

4. Get a list of Active and Complete jobs.

```
aws s3control list-jobs \
  --region us-west-2 \
  --account-id acct-id \
  --job-statuses ['Active','Complete'] \
  --max-results 20
```

5. Update the job priority (a higher number indicates a higher execution priority).

```
aws s3control update-job-priority \
  --region us-west-2 \
  --account-id acct-id \
  --priority 98 \
  --job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c
```

6. If you didn't specify the `--no-confirmation-required` parameter in the `create-job`, the job remains in a suspended state until you confirm the job by setting its status to Ready. Amazon S3 then makes the job eligible for execution.

```
aws s3control update-job-status \
  --region us-west-2 \
  --account-id 181572960644 \
  --job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c \
  --requested-job-status 'Ready'
```

7. Cancel the job by setting the job status to Cancelled.

```
aws s3control update-job-status \
  --region us-west-2 \
  --account-id 181572960644 \
```

```
--job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c \  
--status-update-reason "No longer needed" \  
--requested-job-status Cancelled
```

Java Examples for Amazon S3 Batch Operations

This section provides examples of how to create and manage Amazon S3 batch operations jobs using the AWS SDK for Java. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

Topics

- [Creating an Amazon S3 batch operations Job Using the AWS SDK for Java](#) (p. 498)
- [Canceling an Amazon S3 batch operations Job Using the AWS SDK for Java](#) (p. 499)
- [Updating the Status of a Amazon S3 batch operations Job Using the AWS SDK for Java](#) (p. 500)
- [Updating the Priority of a Amazon S3 batch operations Job Using the AWS SDK for Java](#) (p. 501)

Creating an Amazon S3 batch operations Job Using the AWS SDK for Java

The following example shows how to create an Amazon S3 batch operations job. For information about creating a job, see [Creating an Amazon S3 Batch Operations Job](#) (p. 470).

For information about setting up the permissions you need to create a job, see [Granting Permissions for Amazon S3 Batch Operations](#) (p. 471).

Example

```
package aws.example.s3control;  
  
import com.amazonaws.AmazonServiceException;  
import com.amazonaws.SdkClientException;  
import com.amazonaws.auth.profile.ProfileCredentialsProvider;  
import com.amazonaws.services.s3control.AWSS3Control;  
import com.amazonaws.services.s3control.AWSS3ControlClient;  
import com.amazonaws.services.s3control.model.*;  
  
import java.util.UUID;  
import java.util.ArrayList;  
  
import static com.amazonaws.regions.Regions.US_WEST_2;  
  
public class CreateJob {  
    public static void main(String[] args) {  
        String accountId = "Account ID";  
        String iamRoleArn = "IAM Role ARN";  
        String reportBucketName = "arn:aws:s3:::bucket-where-completion-report-goes";  
        String uuid = UUID.randomUUID().toString();  
  
        ArrayList tagSet = new ArrayList<S3Tag>();  
        tagSet.add(new S3Tag().withKey("keyOne").withValue("ValueOne"));  
  
        try {  
            JobOperation jobOperation = new JobOperation()
```

```

        .withS3PutObjectTagging(new S3SetObjectTaggingOperation()
            .withTagSet(tagSet)
        );

    JobManifest manifest = new JobManifest()
        .withSpec(new JobManifestSpec()
            .withFormat("S3BatchOperations_CSV_20180820")
            .withFields(new String[] {
                "Bucket", "Key"
            })
        .withLocation(new JobManifestLocation()
            .withObjectArn("arn:aws:s3:::my_manifests/manifest.csv")
            .withETag("60e460c9d1046e73f7dde5043ac3ae85"));
    JobReport jobReport = new JobReport()
        .withBucket(reportBucketName)
        .withPrefix("reports")
        .withFormat("Report_CSV_20180820")
        .withEnabled(true)
        .withReportScope("AllTasks");

    AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(US_WEST_2)
        .build();

    s3ControlClient.createJob(new CreateJobRequest()
        .withAccountId(accountId)
        .withOperation(jobOperation)
        .withManifest(manifest)
        .withReport(jobReport)
        .withPriority(42)
        .withRoleArn(iamRoleArn)
        .withClientRequestToken(uuid)
        .withDescription("job description")
        .withConfirmationRequired(false)
    );

} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it and returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
}

```

Canceling an Amazon S3 batch operations Job Using the AWS SDK for Java

The following example shows how to cancel an Amazon S3 batch operations job.

Example

```

package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;

```

```
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.UpdateJobStatusRequest;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class CancelJob {
    public static void main(String[] args) {
        String accountId = "Account ID";
        String jobId = "00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c";

        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            s3ControlClient.updateJobStatus(new UpdateJobStatusRequest()
                .withAccountId(accountId)
                .withJobId(jobId)
                .withStatusUpdateReason("No longer needed")
                .withRequestedJobStatus("Cancelled"));

        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Updating the Status of a Amazon S3 batch operations Job Using the AWS SDK for Java

The following example shows how to update the status of an Amazon S3 batch operations job. For more information about job status, see [Job Status \(p. 486\)](#).

Example

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.UpdateJobStatusRequest;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class UpdateJobStatus {
    public static void main(String[] args) {
        String accountId = "Account ID";
        String jobId = "00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c";

        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
```

```
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(US_WEST_2)
        .build();

    s3ControlClient.updateJobStatus(new UpdateJobStatusRequest()
        .withAccountId(accountId)
        .withJobId(jobId)
        .withRequestedJobStatus("Ready"));

    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it and returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Updating the Priority of a Amazon S3 batch operations Job Using the AWS SDK for Java

The following example shows how to update the priority of an Amazon S3 batch operations job. For more information about job priority, see [Assigning Job Priority \(p. 486\)](#).

Example

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.UpdateJobPriorityRequest;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class UpdateJobPriority {
    public static void main(String[] args) {
        String accountId = "Account ID";
        String jobId = "00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c";

        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            s3ControlClient.updateJobPriority(new UpdateJobPriorityRequest()
                .withAccountId(accountId)
                .withJobId(jobId)
                .withPriority(98));

        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        }
    }
}
```

```
    } catch (SdkClientException e) {  
        // Amazon S3 couldn't be contacted for a response, or the client  
        // couldn't parse the response from Amazon S3.  
        e.printStackTrace();  
    }  
}  
}
```


Hosting a Static Website on Amazon S3

You can host a static website on Amazon Simple Storage Service (Amazon S3). On a static website, individual webpages include static content. They might also contain client-side scripts. By contrast, a dynamic website relies on server-side processing, including server-side scripts such as PHP, JSP, or ASP.NET. Amazon S3 does not support server-side scripting. Amazon Web Services (AWS) also has resources for hosting dynamic websites. To learn more about website hosting on AWS, go to [Websites and Website Hosting](#).

Topics

- [Website Endpoints \(p. 504\)](#)
- [Configuring a Bucket for Website Hosting \(p. 505\)](#)
- [Example Walkthroughs - Hosting Websites on Amazon S3 \(p. 517\)](#)

To host a static website, you configure an Amazon S3 bucket for website hosting, and then upload your website content to the bucket. This bucket must have public read access. It is intentional that everyone in the world will have read access to this bucket. The website is then available at the AWS Region-specific website endpoint of the bucket, which is in one of the following formats:

```
<bucket-name>.s3-website-<AWS-region>.amazonaws.com
```

```
<bucket-name>.s3-website.<AWS-region>.amazonaws.com
```

For a list of AWS Region-specific website endpoints for Amazon S3, see [Website Endpoints \(p. 504\)](#). For example, suppose you create a bucket called `examplebucket` in the US West (Oregon) Region, and configure it as a website. The following example URLs provide access to your website content:

- This URL returns a default index document that you configured for the website.

```
http://examplebucket.s3-website-us-west-2.amazonaws.com/
```

- This URL requests the `photo.jpg` object, which is stored at the root level in the bucket.

```
http://examplebucket.s3-website-us-west-2.amazonaws.com/photo.jpg
```

- This URL requests the `docs/doc1.html` object in your bucket.

```
http://examplebucket.s3-website-us-west-2.amazonaws.com/docs/doc1.html
```

Using Your Own Domain

Instead of accessing the website by using an Amazon S3 website endpoint, you can use your own domain, such as `example.com` to serve your content. Amazon S3, along with Amazon Route 53, supports hosting a website at the root domain. For example, if you have the root domain `example.com`

and you host your website on Amazon S3, your website visitors can access the site from their browser by typing either `http://www.example.com` or `http://example.com`. For an example walkthrough, see [Example: Setting up a Static Website Using a Custom Domain \(p. 519\)](#).

To configure a bucket for website hosting, you add website configuration to the bucket. For more information, see [Configuring a Bucket for Website Hosting \(p. 505\)](#).

Note

The Amazon S3 website endpoints do not support HTTPS. For information about using HTTPS with an Amazon S3 bucket, see [How do I use CloudFront to serve HTTPS requests for my Amazon S3 bucket?](#) and [Requiring HTTPS for Communication Between CloudFront and Your Amazon S3 Origin](#).

Website Endpoints

When you configure a bucket for website hosting, the website is available via the region-specific website endpoint. Website endpoints are different from the endpoints where you send REST API requests. For more information about the differences between the endpoints, see [Key Differences Between the Amazon Website and the REST API Endpoint \(p. 505\)](#).

Note

The Amazon S3 website endpoints do not support HTTPS. For information about using HTTPS with an Amazon S3 bucket, see [How do I use CloudFront to serve HTTPS requests for my Amazon S3 bucket?](#) and [Requiring HTTPS for Communication Between CloudFront and Your Amazon S3 Origin](#).

The two general forms of an Amazon S3 website endpoint are as follows:

```
bucket-name.s3-website-region.amazonaws.com
```

```
bucket-name.s3-website.region.amazonaws.com
```

Which form is used for the endpoint depends on what Region the bucket is in. For example, if your bucket is named `example-bucket` and it resides in the US West (Oregon) region, the website is available at the following Amazon S3 website endpoint:

```
http://example-bucket.s3-website-us-west-2.amazonaws.com/
```

Or, if your bucket is named `example-bucket` and it resides in the EU (Frankfurt) region, the website is available at the following Amazon S3 website endpoint:

```
http://example-bucket.s3-website.eu-central-1.amazonaws.com/
```

For a list of the Amazon S3 website endpoints by Region, see [Amazon Simple Storage Service Website Endpoints](#) in the *AWS General Reference*.

In order for your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can use a bucket policy or an ACL on an object to grant the necessary permissions.

Note

Requester Pays buckets do not allow access through the website endpoint. Any request to such a bucket receives a 403 Access Denied response. For more information, see [Requester Pays Buckets \(p. 80\)](#).

If you have a registered domain, you can add a DNS CNAME entry to point to the Amazon S3 website endpoint. For example, if you have registered domain, `www.example-bucket.com`, you could create a bucket `www.example-bucket.com`, and add a DNS CNAME record that points to `www.example-bucket.com.s3-website-<region>.amazonaws.com`. All requests to `http://www.example-bucket.com` are routed to `www.example-bucket.com.s3-website-<region>.amazonaws.com`. For more information, see [Virtual Hosting of Buckets \(p. 45\)](#).

Key Differences Between the Amazon Website and the REST API Endpoint

The website endpoint is optimized for access from a web browser. The following table describes the key differences between the Amazon REST API endpoint and the website endpoint.

Key Difference	REST API Endpoint	Website Endpoint
Access control	Supports both public and private content.	Supports only publicly readable content.
Error message handling	Returns an XML-formatted error response.	Returns an HTML document.
Redirection support	Not applicable	Supports both object-level and bucket-level redirects.
Requests supported	Supports all bucket and object operations	Supports only GET and HEAD requests on objects.
Responses to GET and HEAD requests at the root of a bucket	Returns a list of the object keys in the bucket.	Returns the index document that is specified in the website configuration.
Secure Sockets Layer (SSL) support	Supports SSL connections.	Does not support SSL connections.

For a list of the Amazon S3 endpoints, see [Request Endpoints \(p. 11\)](#).

Configuring a Bucket for Website Hosting

You can host a static website in an Amazon Simple Storage Service (Amazon S3) bucket. However, to do so requires some configuration. Some optional configurations are also available, depending on your website requirements.

Required configurations:

- [Enabling Website Hosting \(p. 506\)](#)
- [Configuring Index Document Support \(p. 506\)](#)
- [Permissions Required for Website Access \(p. 508\)](#)

Optional configurations:

- [\(Optional\) Configuring Web Traffic Logging \(p. 508\)](#)

- (Optional) [Custom Error Document Support](#) (p. 509)
- (Optional) [Configuring a Webpage Redirect](#) (p. 510)

Enabling Website Hosting

Follow these steps to enable website hosting for your Amazon S3 buckets using the [Amazon S3 console](#):

To enable website hosting for an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the list, choose the bucket that you want to use for your hosted website.
3. Choose the **Properties** tab.
4. Choose **Static website hosting**, and then choose **Use this bucket to host a website**.
5. You are prompted to provide the index document and any optional error documents and redirection rules that are needed.

For information about what an index document is, see [Configuring Index Document Support](#) (p. 506).

Configuring Index Document Support

An *index document* is a webpage that Amazon S3 returns when a request is made to the root of a website or any subfolder. For example, if a user enters `http://www.example.com` in the browser, the user is not requesting any specific page. In that case, Amazon S3 serves up the index document, which is sometimes referred to as the default page.

When you configure your bucket as a website, provide the name of the index document. You then upload an object with this name and configure it to be publicly readable.

The trailing slash at the root-level URL is optional. For example, if you configure your website with `index.html` as the index document, either of the following two URLs return `index.html`.

```
http://example-bucket.s3-website-region.amazonaws.com/  
http://example-bucket.s3-website-region.amazonaws.com
```

For more information about Amazon S3 website endpoints, see [Website Endpoints](#) (p. 504).

Index Documents and Folders

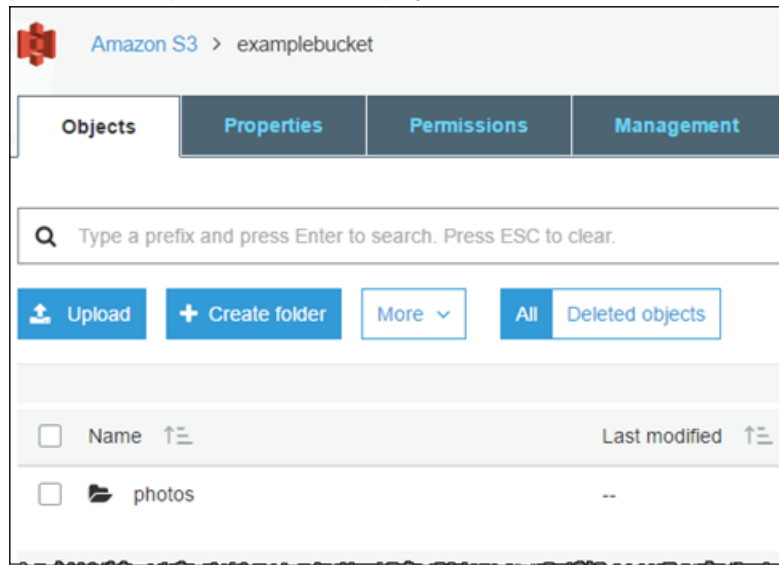
In Amazon S3, a bucket is a flat container of objects; it does not provide any hierarchical organization as the file system on your computer does. You can create a logical hierarchy by using object key names that imply a folder structure. For example, consider a bucket with three objects and the following key names.

- `sample1.jpg`
- `photos/2006/Jan/sample2.jpg`
- `photos/2006/Feb/sample3.jpg`

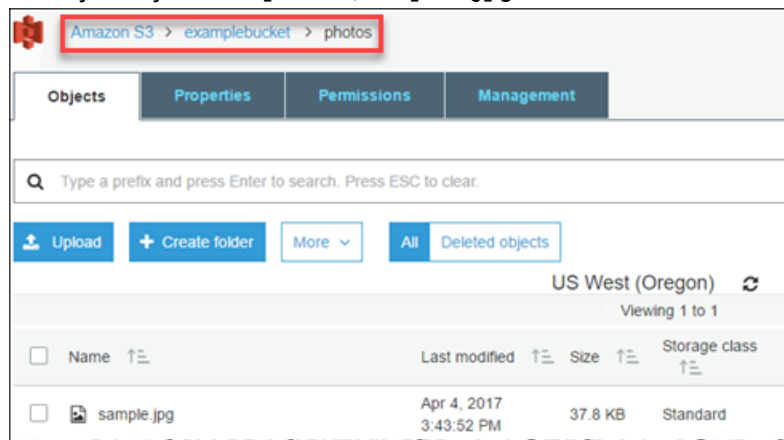
Although these are stored with no physical hierarchical organization, you can infer the following logical folder structure from the key names.

- `sample1.jpg` object is at the root of the bucket.
- `sample2.jpg` object is in the `photos/2006/Jan` subfolder.
- `sample3.jpg` object is in the `photos/2006/Feb` subfolder.

The folder concept that Amazon S3 console supports is based on object key names. To continue the previous example, the console displays the `examplebucket` with a `photos` folder.



You can upload objects to the bucket or to the `photos` folder within the bucket. If you add the object `sample.jpg` to the bucket, the key name is `sample.jpg`. If you upload the object to the `photos` folder, the object key name is `photos/sample.jpg`.



If you create such a folder structure in your bucket, you must have an index document at each level. When a user specifies a URL that resembles a folder lookup, the presence or absence of a trailing slash determines the behavior of the website. For example, the following URL, with a trailing slash, returns the `photos/index.html` index document.

```
http://example-bucket.s3-website-region.amazonaws.com/photos/
```

However, if you exclude the trailing slash from the preceding URL, Amazon S3 first looks for an object `photos` in the bucket. If the `photos` object is not found, then it searches for an index document,

photos/index.html. If that document is found, Amazon S3 returns a 302 Found message and points to the photos/ key. For subsequent requests to photos/, Amazon S3 returns photos/index.html. If the index document is not found, Amazon S3 returns an error.

Permissions Required for Website Access

When you configure a bucket as a website, you must make the objects that you want to serve publicly readable. To do this, you write a bucket policy that grants everyone s3:GetObject permission. On the website endpoint, if a user requests an object that doesn't exist, Amazon S3 returns HTTP response code 404 (Not Found). If the object exists but you haven't granted read permission on it, the website endpoint returns HTTP response code 403 (Access Denied). The user can use the response code to infer whether a specific object exists. If you don't want this behavior, you should not enable website support for your bucket.

The following sample bucket policy grants everyone access to the objects in the specified folder. For more information about bucket policies, see [Using Bucket Policies and User Policies \(p. 341\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "PublicReadGetObject",
    "Effect": "Allow",
    "Principal": "*",
    "Action": ["s3:GetObject"],
    "Resource": ["arn:aws:s3:::example-bucket/*"]
  }]
}
```

Note

Keep the following in mind:

- To host a website, your bucket must have public read access. It is intentional that everyone in the world will have read access to this bucket.
- The bucket policy applies only to objects that are owned by the bucket owner. If your bucket contains objects that aren't owned by the bucket owner, public READ permission on those objects should be granted using the object access control list (ACL).

You can grant public read permission to your objects by using either a bucket policy or an object ACL. To make an object publicly readable using an ACL, grant READ permission to the AllUsers group, as shown in the following grant element. Add this grant element to the object ACL. For information about managing ACLs, see [Managing Access with ACLs \(p. 403\)](#).

```
<Grant>
  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="Group">
    <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
  </Grantee>
  <Permission>READ</Permission>
</Grant>
```

(Optional) Configuring Web Traffic Logging

If you want to track the number of visitors who access your website, enable logging for the root domain bucket. Enabling logging is optional.

To enable logging for your root domain bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create a bucket for logging named `logs.example.com` in the same AWS Region that the `example.com` and `www.example.com` buckets were created in.
3. Create two folders in the `logs.example.com` bucket; one named `root`, and the other named `cdn`. If you configure Amazon CloudFront to speed up your website, you will use the `cdn` folder.
4. In the **Bucket name** list, choose your root domain bucket, choose **Properties**, and then choose **Server access logging**.
5. Choose **Enable logging**.
6. For **Target bucket**, choose the bucket that you created for the log files, `logs.example.com`.
7. For **Target prefix**, type `root/`. This setting groups the log data files in the bucket in a folder named `root` so that they are easy to locate.
8. Choose **Save**.

You can now review your logs in the `logs.example.com` bucket, in both the `root` and `cdn` folders.

(Optional) Custom Error Document Support

The following table lists the subset of HTTP response codes that Amazon S3 returns when an error occurs.

HTTP Error Code	Description
301 Moved Permanently	When a user sends a request directly to the Amazon S3 website endpoints (<code>http://s3-website-<region>.amazonaws.com/</code>), Amazon S3 returns a 301 Moved Permanently response and redirects those requests to <code>https://aws.amazon.com/s3/</code> .
302 Found	When Amazon S3 receives a request for a key <code>x</code> , <code>http://<bucket>.s3-website-<region>.amazonaws.com/x</code> , without a trailing slash, it first looks for the object with the key name <code>x</code> . If the object is not found, Amazon S3 determines that the request is for subfolder <code>x</code> and redirects the request by adding a slash at the end, and returns 302 Found .
304 Not Modified	Amazon S3 uses request headers <code>If-Modified-Since</code> , <code>If-Unmodified-Since</code> , <code>If-Match</code> and/or <code>If-None-Match</code> to determine whether the requested object is same as the cached copy held by the client. If the object is the same, the website endpoint returns a 304 Not Modified response.
400 Malformed Request	The website endpoint responds with a 400 Malformed Request when a user attempts to access a bucket through the incorrect regional endpoint.
403 Forbidden	The website endpoint responds with a 403 Forbidden when a user request translates to an object that is not publicly readable. The object owner must make the object publicly readable using a bucket policy or an ACL.
404 Not Found	The website endpoint responds with 404 Not Found for the following reasons: <ul style="list-style-type: none">• Amazon S3 determines that the URL of the website refers to an object key that does not exist.• Amazon infers that the request is for an index document that does not exist.• A bucket specified in the URL does not exist.

HTTP Error Code	Description
	<ul style="list-style-type: none">A bucket specified in the URL exists, but isn't configured as a website. <p>You can create a custom document that is returned for 404 Not Found. Make sure that the document is uploaded to the bucket configured as a website, and that the website hosting configuration is set to use the document.</p> <p>For information on how Amazon S3 interprets the URL as a request for an object or an index document, see Configuring Index Document Support (p. 506).</p>
500 Service Error	The website endpoint responds with a 500 Service Error when an internal server error occurs.
503 Service Unavailable	The website endpoint responds with a 503 Service Unavailable when Amazon S3 determines that you need to reduce your request rate.

For each of these errors, Amazon S3 returns a predefined HTML message. The following is an example HTML message that is returned for a **403 Forbidden** response.



Custom Error Document

You can optionally provide a custom error document that contains a user-friendly error message and additional help. You provide this custom error document as part of adding website configuration to your bucket. Amazon S3 returns your custom error document for only the HTTP 4XX class of error codes.

Error Documents and Browser Behavior

When an error occurs, Amazon S3 returns an HTML error document. If you configured your website with a custom error document, Amazon S3 returns that error document. However, some browsers display their own error message when an error occurs, ignoring the error document that Amazon S3 returns. For example, when an HTTP 404 Not Found error occurs, Google Chrome might ignore the error document that Amazon S3 returns and display its own error.

(Optional) Configuring a Webpage Redirect

If your Amazon S3 bucket is configured for website hosting, you can redirect requests for an object to another object in the same bucket or to an external URL.

Topics

- [Page Redirect Support in the Amazon S3 Console \(p. 511\)](#)
- [Setting a Page Redirect from the REST API \(p. 512\)](#)
- [Advanced Conditional Redirects \(p. 513\)](#)

You set the redirect by adding the `x-amz-website-redirect-location` property to the object metadata. The website then interprets the object as 301 redirect. To redirect a request to another object, you set the redirect location to the key of the target object. To redirect a request to an external URL, you set the redirect location to the URL that you want. For more information about object metadata, see [System-Defined Object Metadata \(p. 102\)](#).

A bucket configured for website hosting has both the website endpoint and the REST endpoint. A request for a page that is configured as a 301 redirect has the following possible outcomes, depending on the endpoint of the request:

- **Region-specific website endpoint** – Amazon S3 redirects the page request according to the value of the `x-amz-website-redirect-location` property.
- **REST endpoint** – Amazon S3 doesn't redirect the page request. It returns the requested object.

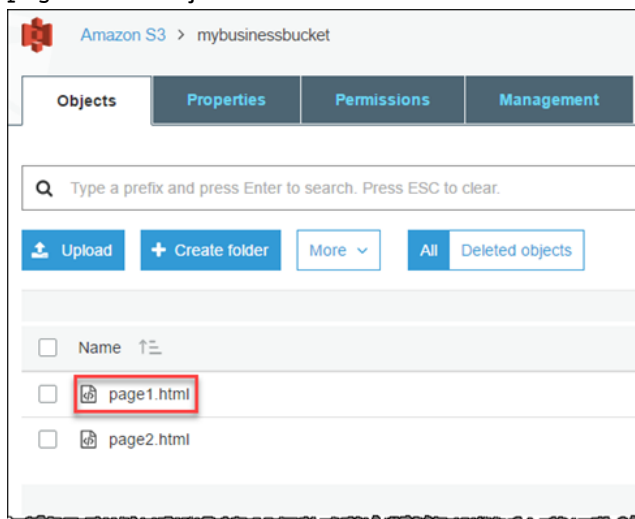
For more information about the endpoints, see [Key Differences Between the Amazon Website and the REST API Endpoint \(p. 505\)](#).

You can set a page redirect from the Amazon S3 console or by using the Amazon S3 REST API.

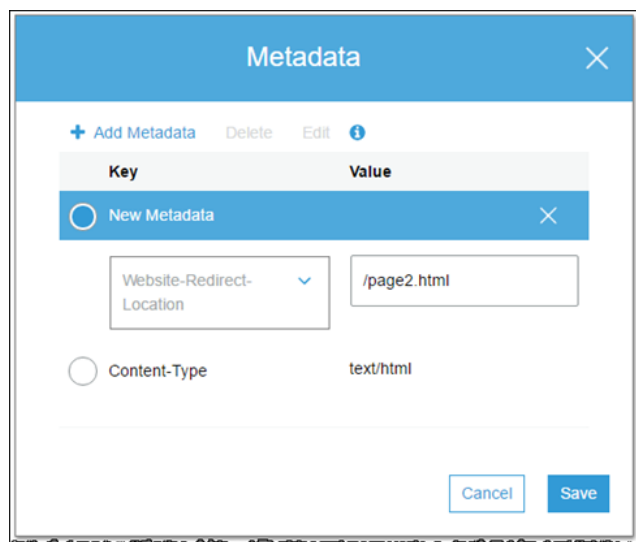
Page Redirect Support in the Amazon S3 Console

You can use the Amazon S3 console to set the website redirect location in the metadata of the object. When you set a page redirect, you can either keep or delete the source object content. For example, suppose that you have a `page1.html` object in your bucket. To redirect any requests for this page to another object, `page2.html`, you can do one of the following:

- To keep the content of the `page1.html` object and only redirect page requests, choose the `page1.html` object.



Choose the **Properties** tab for `page1.html`, and then choose the **Metadata** box. Add **Website Redirect Location** to the metadata, as shown in the following example, and set its value to `/page2.html`. The `/` prefix in the value is required.



You can also set the value to an external URL, such as `http://www.example.com`. For example, if your root domain is `example.com`, and you want to serve requests for both `http://example.com` and `http://www.example.com`, you can create two buckets named `example.com` and `www.example.com`. Then, maintain the content in one of the buckets (say `example.com`), and configure the other bucket to redirect all requests to the `example.com` bucket.

- To delete the content of the `page1.html` object and redirect requests, you can upload a new zero-byte object with the same key, `page1.html`, to replace the existing object. Then specify `Website Redirect Location` for `page1.html` in the upload process. For information about uploading an object, see [Uploading S3 Objects](#) in the *Amazon Simple Storage Service Console User Guide*.

Setting a Page Redirect from the REST API

The following Amazon S3 API actions support the `x-amz-website-redirect-location` header in the request. Amazon S3 stores the header value in the object metadata as `x-amz-website-redirect-location`.

- [PUT Object](#)
- [Initiate Multipart Upload](#)
- [POST Object](#)
- [PUT Object - Copy](#)

When setting a page redirect, you can either keep or delete the object content. For example, suppose you have a `page1.html` object in your bucket.

- To keep the content of `page1.html` and only redirect page requests, you can submit a [PUT Object - Copy](#) request to create a new `page1.html` object that uses the existing `page1.html` object as the source. In your request, you set the `x-amz-website-redirect-location` header. When the request is complete, you have the original page with its content unchanged, but Amazon S3 redirects any requests for the page to the redirect location that you specify.
- To delete the content of the `page1.html` object and redirect requests for the page, you can send a `PUT Object` request to upload a zero-byte object that has the same object key: `page1.html`. In the `PUT` request, you set `x-amz-website-redirect-location` for `page1.html` to the new object. When the request is complete, `page1.html` has no content, and requests are redirected to the location that is specified by `x-amz-website-redirect-location`.

When you retrieve the object using the [GET Object](#) action, along with other object metadata, Amazon S3 returns the `x-amz-website-redirect-location` header in the response.

Advanced Conditional Redirects

Using advanced redirection rules, you can route requests conditionally according to specific object key names, prefixes in the request, or response codes. For example, suppose that you delete or rename an object in your bucket. You can add a routing rule that redirects the request to another object. If you want to make a folder unavailable, you can add a routing rule to redirect the request to another webpage. You can also add a routing rule to handle error conditions by routing requests that return the error to another domain when the error is processed.

When configuring a bucket for website hosting, you have the option of specifying advanced redirection rules.

Static website hosting

Endpoint : <http://example.com.s3-website-us-west-2.amazonaws.com>

☒ Use this bucket to host a website [Learn more](#)

Index document ⓘ

Error document ⓘ

Redirection rules (optional) ⓘ

☐ Redirect requests ⓘ [Learn more](#)

☐ Disable website hosting

To redirect all requests to the bucket's website endpoint to another host, you only need to provide the host name.

You describe the rules using XML. The following section provides general syntax and examples of specifying redirection rules.

Syntax for Specifying Routing Rules

The following is general syntax for defining the routing rules in a website configuration:

```
<RoutingRules> =
  <RoutingRules>
    <RoutingRule>...</RoutingRule>
    [ <RoutingRule>...</RoutingRule>
      ... ]
  </RoutingRules>

<RoutingRule> =
  <RoutingRule>
    [ <Condition>...</Condition> ]
    <Redirect>...</Redirect>
  </RoutingRule>

<Condition> =
  <Condition>
    [ <KeyPrefixEquals>...</KeyPrefixEquals> ]
    [ <HttpErrorCodeReturnedEquals>...</HttpErrorCodeReturnedEquals> ]
  </Condition>
  Note: <Condition> must have at least one child element.

<Redirect> =
  <Redirect>
    [ <HostName>...</HostName> ]
    [ <Protocol>...</Protocol> ]
    [ <ReplaceKeyPrefixWith>...</ReplaceKeyPrefixWith> ]
    [ <ReplaceKeyWith>...</ReplaceKeyWith> ]
    [ <HttpRedirectCode>...</HttpRedirectCode> ]
  </Redirect>
```

*Note: <Redirect> must have at least one child element.
Also, you can have either ReplaceKeyPrefix with or ReplaceKeyWith,
but not both.*

The following table describes the elements in the routing rule.

Name	Description
RoutingRules	Container for a collection of RoutingRule elements.
RoutingRule	<p>A rule that identifies a condition and the redirect that is applied when the condition is met.</p> <p>Condition: A RoutingRules container must contain at least one routing rule.</p>
Condition	Container for describing a condition that must be met for the specified redirect to be applied. If the routing rule does not include a condition, the rule is applied to all requests.
KeyPrefixEquals	<p>The prefix of the object key name from which requests are redirected.</p> <p>KeyPrefixEquals is required if HttpStatusCodeReturnedEquals is not specified. If both KeyPrefixEquals and HttpStatusCodeReturnedEquals are specified, both must be true for the condition to be met.</p>
HttpStatusCodeReturnedEquals	<p>The HTTP error code that must match for the redirect to apply. If an error occurs, and if the error code meets this value, then the specified redirect applies.</p> <p>HttpStatusCodeReturnedEquals is required if KeyPrefixEquals is not specified. If both KeyPrefixEquals and HttpStatusCodeReturnedEquals are specified, both must be true for the condition to be met.</p>
Redirect	Container element that provides instructions for redirecting the request. You can redirect requests to another host or another page, or you can specify another protocol to use. A RoutingRule must have a Redirect element. A Redirect element must contain at least one of the following sibling elements: Protocol, HostName, ReplaceKeyPrefixWith, ReplaceKeyWith, or HttpRedirectCode.
Protocol	<p>The protocol, http or https, to be used in the Location header that is returned in the response.</p> <p>If one of its siblings is supplied, Protocol is not required.</p>
HostName	<p>The hostname to be used in the Location header that is returned in the response.</p> <p>If one of its siblings is supplied, HostName is not required.</p>
ReplaceKeyPrefixWith	<p>The prefix of the object key name that replaces the value of KeyPrefixEquals in the redirect request.</p> <p>If one of its siblings is supplied, ReplaceKeyPrefixWith is not required. It can be supplied only if ReplaceKeyWith is not supplied.</p>

Name	Description
<code>ReplaceKeyWith</code>	The object key to be used in the Location header that is returned in the response. If one of its siblings is supplied, <code>ReplaceKeyWith</code> is not required. It can be supplied only if <code>ReplaceKeyPrefixWith</code> is not supplied.
<code>HttpRedirectCode</code>	The HTTP redirect code to be used in the Location header that is returned in the response. If one of its siblings is supplied, <code>HttpRedirectCode</code> is not required.

The following examples explain common redirection tasks:

Example 1: Redirect after renaming a key prefix

Suppose that your bucket contains the following objects:

- `index.html`
- `docs/article1.html`
- `docs/article2.html`

You decide to rename the folder from `docs/` to `documents/`. After you make this change, you need to redirect requests for prefix `docs/` to `documents/`. For example, request for `docs/article1.html` will be redirected to `documents/article1.html`.

In this case, you add the following routing rule to the website configuration:

```
<RoutingRules>
  <RoutingRule>
    <Condition>
      <KeyPrefixEquals>docs/</KeyPrefixEquals>
    </Condition>
    <Redirect>
      <ReplaceKeyPrefixWith>documents/</ReplaceKeyPrefixWith>
    </Redirect>
  </RoutingRule>
</RoutingRules>
```

Example 2: Redirect requests for a deleted folder to a page

Suppose that you delete the `images/` folder (that is, you delete all objects with the key prefix `images/`). You can add a routing rule that redirects requests for any object with the key prefix `images/` to a page named `folderdeleted.html`.

```
<RoutingRules>
  <RoutingRule>
    <Condition>
      <KeyPrefixEquals>images/</KeyPrefixEquals>
    </Condition>
    <Redirect>
      <ReplaceKeyWith>folderdeleted.html</ReplaceKeyWith>
    </Redirect>
  </RoutingRule>
</RoutingRules>
```

Example 3: Redirect for an HTTP error

Suppose that when a requested object is not found, you want to redirect requests to an Amazon Elastic Compute Cloud (Amazon EC2) instance. Add a redirection rule so that when an HTTP status code 404 (Not Found) is returned, the site visitor is redirected to an Amazon EC2 instance that handles the request. The following example also inserts the object key prefix `report-404/` in the redirect. For example, if you request a page `ExamplePage.html` and it results in an HTTP 404 error, the request is redirected to a page `report-404/ExamplePage.html` on the specified Amazon EC2 instance. If there is no routing rule and the HTTP error 404 occurs, the error document that is specified in the configuration is returned.

```
<RoutingRules>
  <RoutingRule>
    <Condition>
      <HttpErrorCodeReturnedEquals>404</HttpErrorCodeReturnedEquals >
    </Condition>
    <Redirect>
      <HostName>ec2-11-22-333-44.compute-1.amazonaws.com</HostName>
      <ReplaceKeyPrefixWith>report-404/</ReplaceKeyPrefixWith>
    </Redirect>
  </RoutingRule>
</RoutingRules>
```

Example Walkthroughs - Hosting Websites on Amazon S3

Topics

- [Example: Setting up a Static Website \(p. 517\)](#)
- [Example: Setting up a Static Website Using a Custom Domain \(p. 519\)](#)
- [Example: Speed Up Your Website with Amazon CloudFront \(p. 525\)](#)
- [Clean Up Your Example Resources \(p. 528\)](#)

This section provides two examples. In the first example, you configure a bucket for website hosting, upload a sample index document, and test the website using the Amazon S3 website endpoint for the bucket. The second example shows how you can use your own domain, such as `example.com`, instead of the S3 bucket website endpoint, and serve content from an Amazon S3 bucket configured as a website. The example also shows how Amazon S3 offers the root domain support.

Example: Setting up a Static Website

You can configure an Amazon S3 bucket to function like a website. This example walks you through the steps of hosting a website on Amazon S3.

Topics

- [Step 1: Creating a Bucket and Configuring It as a Website \(p. 518\)](#)
- [Step 2: Adding a Bucket Policy That Makes Your Bucket Content Publicly Available \(p. 518\)](#)
- [Step 3: Uploading an Index Document \(p. 518\)](#)
- [Step 4: Testing Your Website \(p. 519\)](#)

Step 1: Creating a Bucket and Configuring It as a Website

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create a bucket.

For step-by-step instructions, see [How Do I Create an S3 Bucket?](#) in *Amazon Simple Storage Service Console User Guide*.

For bucket naming guidelines, see [Bucket Restrictions and Limitations](#) (p. 58). If you have a registered domain name, for additional information about bucket naming, see [Customizing Amazon S3 URLs with CNAMEs](#) (p. 48).

3. Open the bucket **Properties** pane, choose **Static Website Hosting**, and do the following:
 - a. Choose **Use this bucket to host a website**.
 - b. In the **Index Document** box, type the name of your index document. The name is typically `index.html`.
 - c. Choose **Save** to save the website configuration.
 - d. Write down the **Endpoint**.

This is the Amazon S3-provided website endpoint for your bucket. You use this endpoint in the following steps to test your website.

Step 2: Adding a Bucket Policy That Makes Your Bucket Content Publicly Available

1. In the **Properties** pane for the bucket, choose **Permissions**.
2. Choose **Add Bucket Policy**.
3. To host a website, your bucket must have public read access. It is intentional that everyone in the world will have read access to this bucket. Copy the following bucket policy, and then paste it in the Bucket Policy Editor.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "PublicReadForGetBucketObjects",
    "Effect": "Allow",
    "Principal": "*",
    "Action": ["s3:GetObject"],
    "Resource": ["arn:aws:s3:::example-bucket/*"]
  }]
}
```

4. In the policy, replace `example-bucket` with the name of your bucket.
5. Choose **Save**.

Step 3: Uploading an Index Document

1. Create a document. Give it the same name that you gave the index document earlier.
2. Using the console, upload the index document to your bucket.

For instructions, see [Uploading S3 Objects](#) in the *Amazon Simple Storage Service Console User Guide*.

Step 4: Testing Your Website

Type the following URL in the browser, replacing `example-bucket` with the name of your bucket and `website-region` with the name of the AWS Region where you deployed your bucket. For information about AWS Region names, see [Website Endpoints \(p. 504\)](#)).

The two general forms of an Amazon S3 website endpoint are as follows:

```
http://example-bucket.s3-website-region.amazonaws.com
```

```
http://example-bucket.s3-website.region.amazonaws.com
```

If your browser displays your `index.html` page, the website was successfully deployed.

Note

HTTPS access to the website is not supported.

You now have a website hosted on Amazon S3. This website is available at the Amazon S3 website endpoint. However, you might have a domain, such as `example.com`, that you want to use to serve the content from the website you created. You might also want to use Amazon S3 root domain support to serve requests for both `http://www.example.com` and `http://example.com`. This requires additional steps. For an example, see [Example: Setting up a Static Website Using a Custom Domain \(p. 519\)](#).

Example: Setting up a Static Website Using a Custom Domain

Suppose that you want to host your static website on Amazon S3. You registered a domain (for example, `example.com`), and you want requests for `http://www.example.com` and `http://example.com` to be served from your Amazon S3 content. Whether you have an existing static website that you want to host on Amazon S3, or you are starting from scratch, use this example to learn how to host websites on Amazon S3.

Topics

- [Before You Begin \(p. 519\)](#)
- [Step 1: Register a Domain \(p. 520\)](#)
- [Step 2: Create and Configure Buckets and Upload Data \(p. 520\)](#)
- [Step 3: Add Alias Records for `example.com` and `www.example.com` \(p. 523\)](#)
- [Step 4: Testing \(p. 525\)](#)

Before You Begin

As you follow the steps in this example, you work with the following services:

Amazon Route 53 – You use Route 53 to register domains and to define where you want to route internet traffic for your domain. We explain how to create Route 53 alias records that route traffic for your domain (`example.com`) and subdomain (`www.example.com`) to an Amazon S3 bucket that contains an HTML file.

Amazon S3 – You use Amazon S3 to create buckets, upload a sample website page, configure permissions so that everyone can see the content, and then configure the buckets for website hosting.

Step 1: Register a Domain

If you don't already have a registered domain name, such as `example.com`, register one with Route 53. For more information, see [Registering a New Domain](#) in the *Amazon Route 53 Developer Guide*. When you have a registered domain name, your next tasks are to create and configure Amazon S3 buckets for website hosting and to upload your website content.

Step 2: Create and Configure Buckets and Upload Data

To support requests from both the root domain such as `example.com` and subdomain such as `www.example.com`, you create two buckets. One bucket contains the content. You configure the other bucket to redirect requests.

Step 2.1: Create Two Buckets

The bucket names must match the names of the website that you are hosting. For example, to host your `example.com` website on Amazon S3, you would create a bucket named `example.com`. To host a website under `www.example.com`, you would name the bucket `www.example.com`. In this example, your website supports requests from both `example.com` and `www.example.com`.

In this step, you sign in to the Amazon S3 console with your AWS account credentials and create the following two buckets.

- `example.com`
- `www.example.com`

Note

Like domains, subdomains must have their own S3 buckets, and the buckets must share the exact names as the subdomains. In this example, we are creating the `www.example.com` subdomain, so we also need an S3 bucket named `www.example.com`.

To create your buckets and upload your website content for hosting

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create two buckets that match your domain name and subdomain. For instance, `example.com` and `www.example.com`.

For step-by-step instructions, see [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

3. Upload your website data to the `example.com` bucket.

You will host your content out of the root domain bucket (`example.com`), and you will redirect requests for `www.example.com` to the root domain bucket. You can store content in either bucket. For this example, you host content in the `example.com` bucket. The content can be text files, family photos, videos—whatever you want. If you have not yet created a website, then you only need one file for this example. You can upload any file. For example, you can create a file using the following HTML and upload it to the bucket. The file name of the home page of a website is typically `index.html`, but you can give it any name. In a later step, you provide this file name as the index document name for your website.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
  <title>My Website Home Page</title>
</head>
```

```
<body>
  <h1>Welcome to my website</h1>
  <p>Now hosted on Amazon S3!</p>
</body>
</html>
```

For step-by-step instructions, see [How Do I Upload an Object to an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

4. To host a website, your bucket must have public read access. It is intentional that everyone in the world will have read access to this bucket. To grant public read access, attach the following bucket policy to the *example.com* bucket, substituting the name of your bucket for *example.com*. For step-by-step instructions to attach a bucket policy, see [How Do I Add an S3 Bucket Policy?](#) in the *Amazon Simple Storage Service Console User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::example.com/*"
      ]
    }
  ]
}
```

You now have two buckets, *example.com* and *www.example.com*, and you have uploaded your website content to the *example.com* bucket. In the next step, you configure *www.example.com* to redirect requests to your *example.com* bucket. By redirecting requests, you can maintain only one copy of your website content. Visitors who type *www* in their browsers and those who specify only the root domain are routed to the same website content in your *example.com* bucket.

Step 2.2: Configure Buckets for Website Hosting

When you configure a bucket for website hosting, you can access the website using the Amazon S3 assigned bucket website endpoint.

In this step, you configure both buckets for website hosting. First, you configure *example.com* as a website and then you configure *www.example.com* to redirect all requests to the *example.com* bucket.

To configure your buckets for website hosting

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Bucket name** list, choose the name of the bucket that you want to enable static website hosting for.
3. Choose **Properties**.
4. Choose **Static website hosting**.
5. Configure the *example.com* bucket for website hosting. In the **Index Document** box, type the name that you gave your index page.

Static website hosting

Endpoint : `http://example.com.s3-website-us-west-2.amazonaws.com`

☒ Use this bucket to host a website [Learn more](#)

Index document ⓘ

`index.html`

Error document ⓘ

`error.html`

Redirection rules (optional) ⓘ

☒ Redirect requests [Learn more](#)

☐ Disable website hosting

Cancel Save

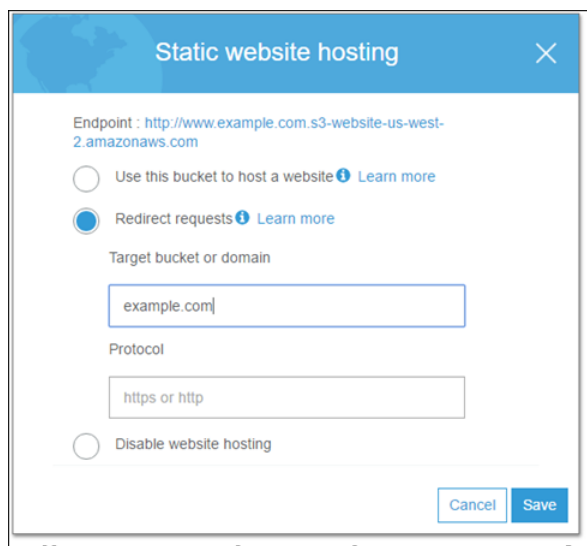
6. Choose **Save**.

Step 2.3: Configure Your Website Redirect

Now that you have configured your bucket for website hosting, configure the `www.example.com` bucket to redirect all requests for `www.example.com` to `example.com`.

To redirect requests from `www.example.com` to `example.com`

1. In the Amazon S3 console, in the **Buckets** list, choose your bucket (`www.example.com`, in this example).
2. Choose **Properties**.
3. Choose **Static website hosting**.
4. Choose **Redirect requests**. In the **Target bucket or domain** box, type `example.com`.
5. Choose **Save**.



Static website hosting

Endpoint : `http://www.example.com.s3-website-us-west-2.amazonaws.com`

☐ Use this bucket to host a website [Learn more](#)

☒ Redirect requests [Learn more](#)

Target bucket or domain

Protocol

☐ Disable website hosting

Step 2.4: Configure Logging for Website Traffic

Optionally, you can configure logging to track the number of visitors accessing your website. To do that, you enable logging for the root domain bucket. For more information, see [\(Optional\) Configuring Web Traffic Logging](#) (p. 508).

Step 2.5: Test Your Endpoint and Redirect

To test the website, type the URL of the endpoint in your browser. Your request is redirected, and the browser displays the index document for `example.com`.

In the next step, you use Amazon Route 53 to enable customers to use all of the URLs to navigate to your site.

Step 3: Add Alias Records for `example.com` and `www.example.com`

In this step, you create the alias records that you add to the hosted zone for your domain maps `example.com` and `www.example.com` to the corresponding S3 buckets. Instead of using IP addresses, the alias records use the Amazon S3 website endpoints. Amazon Route 53 maintains a mapping between the alias records and the IP addresses where the Amazon S3 buckets reside.

To route traffic to your website

1. Open the Route 53 console at <https://console.aws.amazon.com/route53/>.

Note

If you don't already use Amazon Route 53, you can get started [here](#). After you have completed your set up, you can resume the instructions below.

2. In the list of hosted zones, choose the name of your domain.
3. Choose **Create Record Set**.

Note

Each record contains information about how you want to route traffic for one domain (`example.com`) or subdomain (`www.example.com`). Records are stored in the hosted zone for your domain.

4. Specify the following values:

Name

For the first record that you'll create, accept the default value, which is the name of your hosted zone and your domain. This will route internet traffic to the bucket that has the same name as your domain.

Repeat this step to create a second record for your subdomain. For the second record, type **www**. This will route internet traffic to the `www.example.com` bucket.

Type

Choose **A – IPv4 address**.

Alias

Choose **Yes**.

Alias Target

Type the name of your Amazon S3 bucket endpoint, for example `example.com (s3-website-us-west-2)`.

Note

You specify the same value for **Alias Target** for both records. Route 53 figures out which bucket to route traffic to based on the name of the record.

Routing Policy

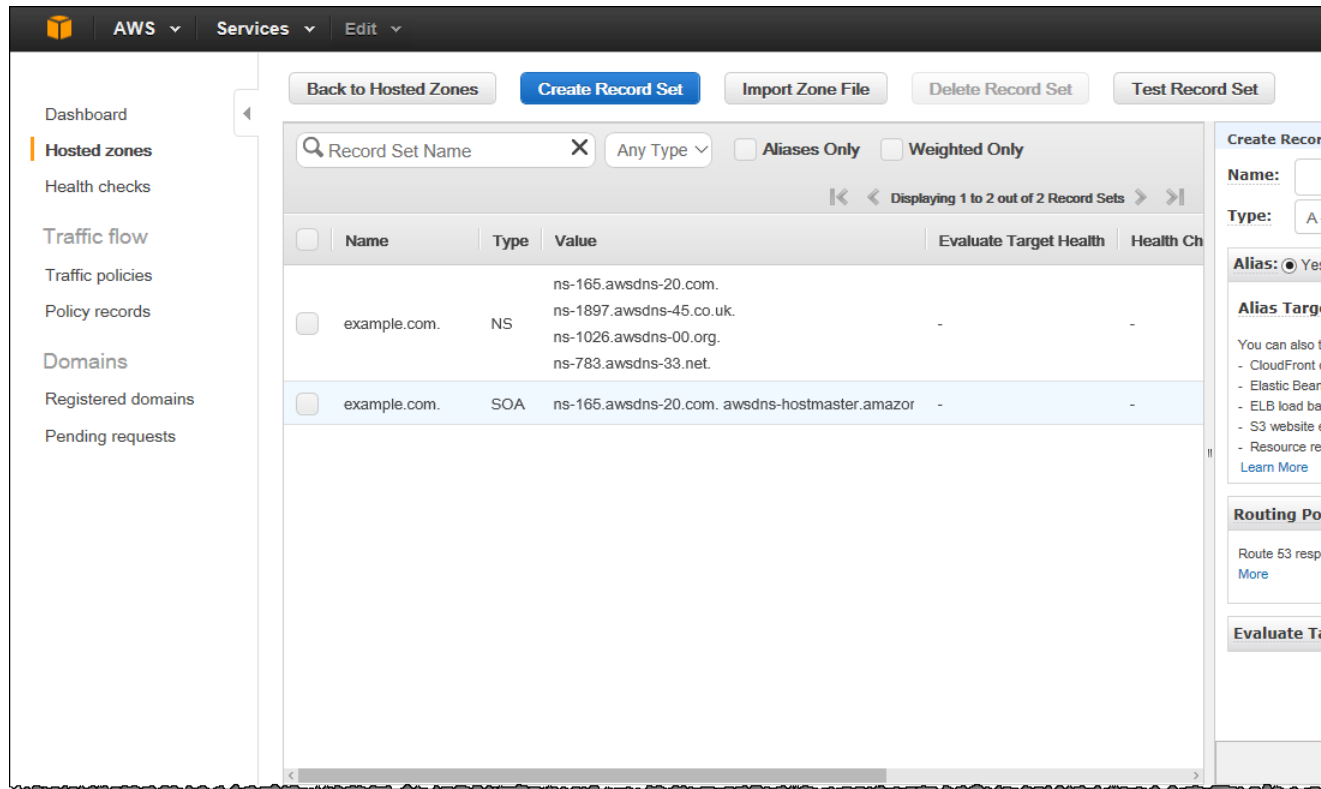
Accept the default value of **Simple**.

Evaluate Target Health

Accept the default value of **No**.

5. Choose **Create**.
6. For `www.example.com`, repeat steps 3 through 5 to create a record.

The following screenshot shows the alias record for `example.com` as an illustration. You also need to create an alias record for `www.example.com`.



Note

Creating, changing, and deleting resource record sets take time to propagate to the Route 53 DNS servers. Changes generally propagate to all Route 53 name servers in a couple of minutes. In rare circumstances, propagation can take up to 30 minutes.

Step 4: Testing

To verify that the website is working correctly, in your browser, try the following URLs:

- <http://example.com> – Displays the index document in the [example.com](#) bucket.
- <http://www.example.com> – Redirects your request to <http://example.com>.

In some cases, you might need to clear the cache of your web browser to see the expected behavior.

Example: Speed Up Your Website with Amazon CloudFront

You can use [Amazon CloudFront](#) to improve the performance of your website. CloudFront makes your website's files (such as HTML, images, and video) available from data centers around the world (called *edge locations*). When a visitor requests a file from your website, CloudFront automatically redirects the request to a copy of the file at the nearest edge location. This results in faster download times than if the visitor had requested the content from a data center that is located farther away.

CloudFront caches content at edge locations for a period of time that you specify. If a visitor requests content that has been cached for longer than the expiration date, CloudFront checks the origin server to see if a newer version of the content is available. If a newer version is available, CloudFront copies the

new version to the edge location. Changes that you make to the original content are replicated to edge locations as visitors request the content.

To speed up your website, use CloudFront to complete the following tasks.

Tasks

- [Create a CloudFront Distribution](#) (p. 526)
- [Update the Record Sets for Your Domain and Subdomain](#) (p. 527)
- [\(Optional\) Check the Log Files](#) (p. 527)

Create a CloudFront Distribution

First, you create a CloudFront distribution. This makes your website available from data centers around the world.

To create a distribution with an Amazon S3 origin

1. Open the CloudFront console at <https://console.aws.amazon.com/cloudfront/>.
2. Choose **Create Distribution**.
3. On the **Select a delivery method for your content** page, for **Web**, choose **Get Started**.
4. On the **Create Distribution** page, in the **Origin Settings** section, for **Origin Domain Name**, type the Amazon S3 static website hosting endpoint for your bucket. For example, `example.com.s3.amazonaws.com`.

CloudFront fills in the **Origin ID** for you.

5. For **Default Cache Behavior Settings**, leave the values set to the defaults. For more information about these configuration options, see [Values that You Specify When You Create or Update a Web Distribution](#) in the *Amazon CloudFront Developer Guide*.
6. For **Distribution Settings**, do the following:
 - a. Leave **Price Class** set to **Use All Edge Locations (Best Performance)**.
 - b. Set **Alternate Domain Names (CNAMEs)** to the root domain and `www` subdomain; in this tutorial, these are `example.com` and `www.example.com`. These values must be set before you create aliases for the A records that connect the specified domain names to the CloudFront distribution.

Important

Prior to performing this step, note the [requirements for using alternate domain names](#), in particular the need for a valid SSL/TLS certificate.

- c. Set **Default Root Object** to `index.html`. This is the default page that the CloudFront distribution returns if the URL used to access the distribution doesn't contain a file name. This value should match the index document value that you set in [Configuring a Bucket for Website Hosting](#) (p. 505).
 - d. Set **Logging** to **On**.
 - e. For **Bucket for Logs**, choose the logging bucket that you created.
 - f. To store the logs generated by traffic to the CloudFront distribution in a folder, named `cdn/` in the log bucket, type `cdn/` for **Log Prefix**.
 - g. Leave the other settings at their default values.
7. Choose **Create Distribution**.

To see the status of the distribution, find the distribution in the console and check the **Status** column. A status of `InProgress` indicates that the distribution is not yet fully deployed.

After your distribution is deployed, you can reference your content with the new CloudFront domain name. Record the value of **Domain Name** shown in the CloudFront console. You'll need it in the next step. In this example, the value is `dj4p1rv6mvubz.cloudfront.net`.

To verify that your CloudFront distribution is working, type the domain name of the distribution in a web browser. If it is working, your website is visible.

Update the Record Sets for Your Domain and Subdomain

Now that you have successfully created a CloudFront distribution, update the A records in Route 53 to point to the new CloudFront distribution.

To update A records to point to a CloudFront distribution

1. Open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. On the **Hosted Zones** page, choose the hosted zone that you created for your domain.
3. Choose **Go to Record Sets**.
4. Choose the A record that you created for the `www` subdomain.
5. For **Alias Target**, choose the CloudFront distribution.
6. Choose **Save Record Set**.
7. To redirect the A record for the root domain to the CloudFront distribution, repeat this procedure.

The update to the record sets takes effect within 2 to 48 hours. To see if the new A records have taken effect, in a web browser, type `http://www.example.com`. If the browser no longer redirects you to `http://example.com`, the new A records are in place.

This change in behavior occurs because traffic routed by the *old* A record to the `www` subdomain S3 bucket is redirected by the settings in Amazon S3 to the root domain. When the new A record has taken effect, traffic routed by the new A record to the CloudFront distribution is not redirected to the root domain.

Tip

Browsers can cache redirect settings. If you think the new A record settings should have taken effect, but your browser still redirects `http://www.example.com` to `http://example.com`, try clearing your browser history and cache, closing and reopening your browser application, or using a different web browser.

When the new A records are in effect, any visitors who reference the site by using `http://example.com` or `http://www.example.com` are redirected to the nearest CloudFront edge location, where they benefit from faster download times.

If you created your site as a learning exercise only, you can delete the resources that you allocated so that you no longer accrue charges. To do so, continue on to [Clean Up Your Example Resources \(p. 528\)](#). After you delete your AWS resources, your website is no longer available.

(Optional) Check the Log Files

The access logs tell you how many people are visiting the website. They also contain valuable business data that you can analyze with other services, such as [Amazon EMR](#).

In your bucket, older Amazon S3 log files are located in the `root` folder. All new log files, which should be CloudFront logs, are located in the `cdn` folder. Amazon S3 writes website access logs to your log bucket every two hours. CloudFront writes logs to your log bucket within 24 hours from when the corresponding requests are made.

To see the log files for your website

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose the logging bucket for your website.
3. To see the log files that are stored in the `cdn` or `root` folder, choose `cdn` or `root`.
4. Open Amazon S3 log files, which are text files, in a browser. Download the .gzip files written by CloudFront before opening them.

Clean Up Your Example Resources

If you created your static website as a learning exercise only, be sure to delete the AWS resources that you allocated so that you no longer accrue charges. After you delete your AWS resources, your website is no longer available.

Tasks

- [Delete the Amazon CloudFront Distribution \(p. 528\)](#)
- [Delete the Route 53 Hosted Zone \(p. 528\)](#)
- [Delete the S3 Bucket \(p. 529\)](#)

Delete the Amazon CloudFront Distribution

Before you delete an Amazon CloudFront distribution, you must disable it. A disabled distribution is no longer functional and does not accrue charges. You can enable a disabled distribution at any time. After you delete a disabled distribution, it is no longer available.

To disable and delete a CloudFront distribution

1. Open the CloudFront console at <https://console.aws.amazon.com/cloudfront/>.
2. Select the distribution that you want to disable, and then choose **Disable**.
3. When prompted for confirmation, choose **Yes, Disable**.
4. Select the disabled distribution, and then choose **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Delete the Route 53 Hosted Zone

Before you delete the hosted zone, you must delete the record sets that you created. You don't need to delete the NS and SOA records; these are automatically deleted when you delete the hosted zone.

To delete the record sets

1. Open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. In the list of domain names, select your domain name, and then choose **Go to Record Sets**.
3. In the list of record sets, select the A records that you created. The type of each record set is listed in the **Type** column.
4. Choose **Delete Record Set**.
5. When prompted for confirmation, choose **Confirm**.

To delete an Route 53 hosted zone

1. Continuing from the previous procedure, choose **Back to Hosted Zones**.

2. Select your domain name, and then choose **Delete Hosted Zone**.
3. When prompted for confirmation, choose **Confirm**.

Delete the S3 Bucket

Before you delete your S3 bucket, make sure that logging is disabled for the bucket. Otherwise, AWS continues to write logs to your bucket as you delete it.

To disable logging for a bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Select your bucket, and then choose **Properties**.
3. From **Properties**, choose **Logging**.
4. Clear the **Enabled** check box.
5. Choose **Save**.

Now, you can delete your bucket. For more information, see [How Do I Delete an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Configuring Amazon S3 Event Notifications

The Amazon S3 notification feature enables you to receive notifications when certain events happen in your bucket. To enable notifications, you must first add a notification configuration identifying the events you want Amazon S3 to publish, and the destinations where you want Amazon S3 to send the event notifications. You store this configuration in the *notification* subresource (see [Bucket Configuration Options \(p. 56\)](#)) associated with a bucket. Amazon S3 provides an API for you to manage this subresource.

Important

Amazon S3 event notifications typically deliver events in seconds but can sometimes take a minute or longer. On very rare occasions, events might be lost.

If your application requires particular semantics (for example, ensuring that no events are missed, or that operations run only once), we recommend that you account for missed and duplicate events when designing your application. You can audit for missed events by using the [LIST Objects](#) API or [Amazon S3 Inventory \(p. 422\)](#) reports. The LIST Objects API and Amazon S3 inventory reports are subject to [eventual consistency](#) and might not reflect recently added or deleted objects.

Topics

- [Overview \(p. 530\)](#)
- [How to Enable Event Notifications \(p. 532\)](#)
- [Event Notification Types and Destinations \(p. 533\)](#)
- [Configuring Notifications with Object Key Name Filtering \(p. 534\)](#)
- [Granting Permissions to Publish Event Notification Messages to a Destination \(p. 539\)](#)
- [Example Walkthrough 1: Configure a Bucket for Notifications \(Message Destination: SNS Topic and SQS Queue\) \(p. 541\)](#)
- [Example Walkthrough 2: Configure a Bucket for Notifications \(Message Destination: AWS Lambda\) \(p. 546\)](#)
- [Event Message Structure \(p. 546\)](#)

Overview

Currently, Amazon S3 can publish notifications for the following events:

- A new object created event—Amazon S3 supports multiple APIs to create objects. You can request notification when only a specific API is used (e.g., `s3:ObjectCreated:Put`) or you can use a wildcard (e.g., `s3:ObjectCreated:*`) to request notification when an object is created regardless of the API used.
- An object removal event—Amazon S3 supports deletes of versioned and unversioned objects. For information about object versioning, see [Object Versioning \(p. 108\)](#) and [Using Versioning \(p. 432\)](#).

You can request notification when an object is deleted or a versioned object is permanently deleted by using the `s3:ObjectRemoved:Delete` event type. Or you can request notification when a delete marker is created for a versioned object by using `s3:ObjectRemoved:DeleteMarkerCreated`. You

can also use a wildcard `s3:ObjectRemoved:*` to request notification anytime an object is deleted. For information about deleting versioned objects, see [Deleting Object Versions \(p. 444\)](#).

- Restore object events—Amazon S3 supports the restoration of objects archived to the GLACIER storage class. You request to be notified of object restoration completion by using `s3:ObjectRestore:Completed`. You use `s3:ObjectRestore:Post` to request notification of the initiation of a restore.
- A Reduced Redundancy Storage (RRS) object lost event—Amazon S3 sends a notification message when it detects that an object of the RRS storage class has been lost.

For a list of supported event types, see [Supported Event Types \(p. 533\)](#).

Amazon S3 supports the following destinations where it can publish events:

- Amazon Simple Notification Service (Amazon SNS) topic

Amazon SNS is a flexible, fully managed push messaging service. Using this service, you can push messages to mobile devices or distributed services. With SNS you can publish a message once, and deliver it one or more times. An SNS topic is an access point that recipients can dynamically subscribe to in order to receive event notifications. For more information about SNS, see the [Amazon SNS](#) product detail page.

- Amazon Simple Queue Service (Amazon SQS) queue

Amazon SQS is a scalable and fully managed message queuing service. You can use SQS to transmit any volume of data without requiring other services to be always available. In your notification configuration you can request that Amazon S3 publish events to an SQS queue. Currently, Standard SQS queue is only allowed as an Amazon S3 event notification destination, whereas FIFO SQS queue is not allowed. For more information about SQS, see [Amazon SQS](#) product detail page.

- AWS Lambda

AWS Lambda is a compute service that makes it easy for you to build applications that respond quickly to new information. AWS Lambda runs your code in response to events such as image uploads, in-app activity, website clicks, or outputs from connected devices. You can use AWS Lambda to extend other AWS services with custom logic, or create your own back-end that operates at AWS scale, performance, and security. With AWS Lambda, you can easily create discrete, event-driven applications that execute only when needed and scale automatically from a few requests per day to thousands per second.

AWS Lambda can run custom code in response to Amazon S3 bucket events. You upload your custom code to AWS Lambda and create what is called a Lambda function. When Amazon S3 detects an event of a specific type (for example, an object created event), it can publish the event to AWS Lambda and invoke your function in Lambda. In response, AWS Lambda executes your function. For more information, see [AWS Lambda](#) product detail page.

The following sections offer more detail about how to enable event notifications on a bucket. The subtopics also provide example walkthroughs to help you explore the notification feature.

- [Example Walkthrough 1: Configure a Bucket for Notifications \(Message Destination: SNS Topic and SQS Queue\) \(p. 541\)](#)

- [Example Walkthrough 2: Configure a Bucket for Notifications \(Message Destination: AWS Lambda\)](#) (p. 546)

How to Enable Event Notifications

Enabling notifications is a bucket-level operation; that is, you store notification configuration information in the *notification* subresource associated with a bucket. You can use any of the following methods to manage notification configuration:

- Using the Amazon S3 console

The console UI enables you to set a notification configuration on a bucket without having to write any code. For instruction, see [How Do I Enable and Configure Event Notifications for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

- Programmatically using the AWS SDKs

Note

If you need to, you can also make the Amazon S3 REST API calls directly from your code. However, this can be cumbersome because it requires you to write code to authenticate your requests.

Internally, both the console and the SDKs call the Amazon S3 REST API to manage *notification* subresources associated with the bucket. For notification configuration using AWS SDK examples, see the walkthrough link provided in the preceding section.

Regardless of the method you use, Amazon S3 stores the notification configuration as XML in the *notification* subresource associated with a bucket. For information about bucket subresources, see [Bucket Configuration Options](#) (p. 56)). By default, notifications are not enabled for any type of event. Therefore, initially the *notification* subresource stores an empty configuration.

```
<NotificationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
</NotificationConfiguration>
```

To enable notifications for events of specific types, you replace the XML with the appropriate configuration that identifies the event types you want Amazon S3 to publish and the destination where you want the events published. For each destination, you add a corresponding XML configuration. For example:

- Publish event messages to an SQS queue—To set an SQS queue as the notification destination for one or more event types, you add the `QueueConfiguration`.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Id>optional-id-string</Id>
    <Queue>sqs-queue-arn</Queue>
    <Event>event-type</Event>
    <Event>event-type</Event>
    ...
  </QueueConfiguration>
  ...
</NotificationConfiguration>
```

- Publish event messages to an SNS topic—To set an SNS topic as the notification destination for specific event types, you add the `TopicConfiguration`.

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Id>optional-id-string</Id>
```

```
<Topic>sns-topic-arn</Topic>
<Event>event-type</Event>
<Event>event-type</Event>
...
</TopicConfiguration>
...
</NotificationConfiguration>
```

- Invoke the AWS Lambda function and provide an event message as an argument—To set a Lambda function as the notification destination for specific event types, you add the `CloudFunctionConfiguration`.

```
<NotificationConfiguration>
  <CloudFunctionConfiguration>
    <Id>optional-id-string</Id>
    <Cloudcode>cloud-function-arn</Cloudcode>
    <Event>event-type</Event>
    <Event>event-type</Event>
    ...
  </CloudFunctionConfiguration>
  ...
</NotificationConfiguration>
```

To remove all notifications configured on a bucket, you save an empty `<NotificationConfiguration/>` element in the *notification* subresource.

When Amazon S3 detects an event of the specific type, it publishes a message with the event information. For more information, see [Event Message Structure \(p. 546\)](#).

Event Notification Types and Destinations

This section describes the event notification types that are supported by Amazon S3 and the type of destinations where the notifications can be published.

Supported Event Types

Amazon S3 can publish events of the following types. You specify these event types in the notification configuration.

Event types	Description
<i>s3:ObjectCreated:*</i>	Amazon S3 APIs such as PUT, POST, and COPY can create an object. Using these event types, you can enable notification when an object is created using a specific API, or you can use the <i>s3:ObjectCreated:*</i> event type to request notification regardless of the API that was used to create an object.
<i>s3:ObjectCreated:Put</i>	
<i>s3:ObjectCreated:Post</i>	
<i>s3:ObjectCreated:Copy</i>	
<i>s3:ObjectCreated:CompleteMultipartUpload</i>	
<i>s3:ObjectRemoved:*</i>	By using the <i>ObjectRemoved</i> event types, you can enable notification when an object or a batch of objects is removed from a bucket.
<i>s3:ObjectRemoved>Delete</i>	
<i>s3:ObjectRemoved>DeleteMarkerCreated</i>	
	You can request notification when an object is deleted or a versioned object is permanently deleted by using the

Event types	Description
	<p><i>s3:ObjectRemoved:Delete</i> event type. Or you can request notification when a delete marker is created for a versioned object by using <i>s3:ObjectRemoved:DeleteMarkerCreated</i>. For information about deleting versioned objects, see Deleting Object Versions (p. 444). You can also use a wildcard <i>s3:ObjectRemoved:*</i> to request notification anytime an object is deleted.</p> <p>You will not receive event notifications from automatic deletes from lifecycle policies or from failed operations.</p>
<p><i>s3:ObjectRestore:Post</i></p> <p><i>s3:ObjectRestore:Completed</i></p>	<p>Using restore object event types you can receive notifications for initiation and completion when restoring objects from the GLACIER storage class.</p> <p>You use <i>s3:ObjectRestore:Post</i> to request notification of object restoration initiation. You use <i>s3:ObjectRestore:Completed</i> to request notification of restoration completion.</p>
<i>s3:ReducedRedundancyLostObject</i>	<p>You can use this event type to request Amazon S3 to send a notification message when Amazon S3 detects that an object of the RRS storage class is lost.</p>

Supported Destinations

Amazon S3 can send event notification messages to the following destinations. You specify the ARN value of these destinations in the notification configuration.

- Publish event messages to an Amazon Simple Notification Service (Amazon SNS) topic
- Publish event messages to an Amazon Simple Queue Service (Amazon SQS) queue

Note

If the destination queue or topic is SSE enabled, Amazon S3 will need access to the associated KMS key to enable message encryption.

- Publish event messages to AWS Lambda by invoking a Lambda function and providing the event message as an argument

You must grant Amazon S3 permissions to post messages to an Amazon SNS topic or an Amazon SQS queue. You must also grant Amazon S3 permission to invoke an AWS Lambda function on your behalf. For information about granting these permissions, see [Granting Permissions to Publish Event Notification Messages to a Destination \(p. 539\)](#).

Configuring Notifications with Object Key Name Filtering

You can configure notifications to be filtered by the prefix and suffix of the key name of objects. For example, you can set up a configuration so that you are sent a notification only when image files with a ".jpg" extension are added to a bucket. Or you can have a configuration that delivers a notification to an Amazon SNS topic when an object with the prefix "images/" is added to the bucket, while having notifications for objects with a "logs/" prefix in the same bucket delivered to an AWS Lambda function.

You can set up notification configurations that use object key name filtering in the Amazon S3 console and by using Amazon S3 APIs through the AWS SDKs or the REST APIs directly. For information about using the console UI to set a notification configuration on a bucket, see [How Do I Enable and Configure Event Notifications for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Amazon S3 stores the notification configuration as XML in the *notification* subresource associated with a bucket as described in [How to Enable Event Notifications](#) (p. 532). You use the `Filter` XML structure to define the rules for notifications to be filtered by the prefix and/or suffix of an object key name. For information about the details of the `Filter` XML structure, see [PUT Bucket notification](#) in the *Amazon Simple Storage Service API Reference*.

Notification configurations that use `Filter` cannot define filtering rules with overlapping prefixes, overlapping suffixes, or prefix and suffix overlapping. The following sections have examples of valid notification configurations with object key name filtering and examples of notification configurations that are invalid because of prefix/suffix overlapping.

Examples of Valid Notification Configurations with Object Key Name Filtering

The following notification configuration contains a queue configuration identifying an Amazon SQS queue for Amazon S3 to publish events to of the `s3:ObjectCreated:Put` type. The events will be published whenever an object that has a prefix of `images/` and a `jpg` suffix is PUT to a bucket.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images</Value>
        </FilterRule>
        <FilterRule>
          <Name>suffix</Name>
          <Value>jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Queue>arn:aws:sqs:us-west-2:444455556666:s3notificationqueue</Queue>
    <Event>s3:ObjectCreated:Put</Event>
  </QueueConfiguration>
</NotificationConfiguration>
```

The following notification configuration has multiple non-overlapping prefixes. The configuration defines that notifications for PUT requests in the `images/` folder will go to queue-A while notifications for PUT requests in the `logs/` folder will go to queue-B.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Queue>arn:aws:sqs:us-west-2:444455556666:sqs-queue-A</Queue>
    <Event>s3:ObjectCreated:Put</Event>
```

```
</QueueConfiguration>
<QueueConfiguration>
  <Id>2</Id>
  <Filter>
    <S3Key>
      <FilterRule>
        <Name>prefix</Name>
        <Value>logs</Value>
      </FilterRule>
    </S3Key>
  </Filter>
  <Queue>arn:aws:sqs:us-west-2:444455556666:sqs-queue-B</Queue>
  <Event>s3:ObjectCreated:Put</Event>
</QueueConfiguration>
</NotificationConfiguration>
```

The following notification configuration has multiple non-overlapping suffixes. The configuration defines that all .jpg images newly added to the bucket will be processed by Lambda cloud-function-A and all newly added .png images will be processed by cloud-function-B. The suffixes .png and .jpg are not overlapping even though they have the same last letter. Two suffixes are considered overlapping if a given string can end with both suffixes. A string cannot end with both .png and .jpg so the suffixes in the example configuration are not overlapping suffixes.

```
<NotificationConfiguration>
  <CloudFunctionConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>.jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Cloudcode>arn:aws:lambda:us-west-2:444455556666:cloud-function-A</Cloudcode>
    <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
  <CloudFunctionConfiguration>
    <Id>2</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>.png</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Cloudcode>arn:aws:lambda:us-west-2:444455556666:cloud-function-B</Cloudcode>
    <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
</NotificationConfiguration>
```

Your notification configurations that use `Filter` cannot define filtering rules with overlapping prefixes for the same event types, unless the overlapping prefixes are used with suffixes that do not overlap. The following example configuration shows how objects created with a common prefix but non-overlapping suffixes can be delivered to different destinations.

```
<NotificationConfiguration>
  <CloudFunctionConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
```

```
<FilterRule>
  <Name>prefix</Name>
  <Value>images</Value>
</FilterRule>
<FilterRule>
  <Name>suffix</Name>
  <Value>.jpg</Value>
</FilterRule>
</S3Key>
</Filter>
<Cloudcode>arn:aws:lambda:us-west-2:444455556666:cloud-function-A</Cloudcode>
<Event>s3:ObjectCreated:Put</Event>
</CloudFunctionConfiguration>
<CloudFunctionConfiguration>
  <Id>2</Id>
  <Filter>
    <S3Key>
      <FilterRule>
        <Name>prefix</Name>
        <Value>images</Value>
      </FilterRule>
      <FilterRule>
        <Name>suffix</Name>
        <Value>.png</Value>
      </FilterRule>
    </S3Key>
  </Filter>
  <Cloudcode>arn:aws:lambda:us-west-2:444455556666:cloud-function-B</Cloudcode>
  <Event>s3:ObjectCreated:Put</Event>
</CloudFunctionConfiguration>
</NotificationConfiguration>
```

Examples of Notification Configurations with Invalid Prefix/Suffix Overlapping

Your notification configurations that use `Filter`, for the most part, cannot define filtering rules with overlapping prefixes, overlapping suffixes, or overlapping combinations of prefixes and suffixes for the same event types. (You can have overlapping prefixes as long as the suffixes do not overlap. For an example, see [Configuring Notifications with Object Key Name Filtering \(p. 534\)](#).)

You can use overlapping object key name filters with different event types. For example, you could create a notification configuration that uses the prefix `image/` for the `ObjectCreated:Put` event type and the prefix `image/` for the `ObjectDeleted:*` event type.

You will get an error if you try to save a notification configuration that has invalid overlapping name filters for the same event types, when using the AWS Amazon S3 console or when using the Amazon S3 API. This section shows examples of notification configurations that are invalid because of overlapping name filters.

Any existing notification configuration rule is assumed to have a default prefix and suffix that match any other prefix and suffix respectively. The following notification configuration is invalid because it has overlapping prefixes, where the root prefix overlaps with any other prefix. (The same thing would be true if we were using suffix instead of prefix in this example. The root suffix overlaps with any other suffix.)

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-notification-one</Topic>
    <Event>s3:ObjectCreated:*</Event>
  </TopicConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-notification-two</Topic>
```

```
<Event>s3:ObjectCreated:*/Event>
<Filter>
  <S3Key>
    <FilterRule>
      <Name>prefix</Name>
      <Value>images</Value>
    </FilterRule>
  </S3Key>
</Filter>
</TopicConfiguration>
</NotificationConfiguration>
```

The following notification configuration is invalid because it has overlapping suffixes. Two suffixes are considered overlapping if a given string can end with both suffixes. A string can end with `jpg` and `pg` so the suffixes are overlapping. (The same is true for prefixes, two prefixes are considered overlapping if a given string can begin with both prefixes.)

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-one</Topic>
    <Event>s3:ObjectCreated:*/Event>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
  </TopicConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-two</Topic>
    <Event>s3:ObjectCreated:Put</Event>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>pg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
  </TopicConfiguration>
</NotificationConfiguration>
```

The following notification configuration is invalid because it has overlapping prefixes and suffixes.

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-one</Topic>
    <Event>s3:ObjectCreated:*/Event>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images</Value>
        </FilterRule>
        <FilterRule>
          <Name>suffix</Name>
          <Value>jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
  </TopicConfiguration>
```

```
<TopicConfiguration>
  <Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-two</Topic>
  <Event>s3:ObjectCreated:Put</Event>
  <Filter>
    <S3Key>
      <FilterRule>
        <Name>suffix</Name>
        <Value>jpg</Value>
      </FilterRule>
    </S3Key>
  </Filter>
</TopicConfiguration>
</NotificationConfiguration>
```

Granting Permissions to Publish Event Notification Messages to a Destination

Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function.

Granting Permissions to Invoke an AWS Lambda Function

Amazon S3 publishes event messages to AWS Lambda by invoking a Lambda function and providing the event message as an argument.

When you use the Amazon S3 console to configure event notifications on an Amazon S3 bucket for a Lambda function, the Amazon S3 console will set up the necessary permissions on the Lambda function so that Amazon S3 has permissions to invoke the function from the bucket. For more information, see [How Do I Enable and Configure Event Notifications for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

You can also grant Amazon S3 permissions from AWS Lambda to invoke your Lambda function. For more information, see [Tutorial: Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

Granting Permissions to Publish Messages to an SNS Topic or an SQS Queue

You attach an IAM policy to the destination SNS topic or SQS queue to grant Amazon S3 permissions to publish messages to the SNS topic or SQS queue.

Example of an IAM policy that you attach to the destination SNS topic.

```
{
  "Version": "2008-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
    },
  ],
}
```

```
"Action": [
  "SNS:Publish"
],
"Resource": "arn:aws:sns:REGION:ACCOUNT-ID:TOPICNAME",
"Condition": {
  "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket-name" }
}
}
```

Example of an IAM policy that you attach to the destination SQS queue.

```
{
  "Version": "2008-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SQS:SendMessage"
      ],
      "Resource": "arn:aws:sqs:REGION:ACCOUNT-ID:QUEUENAMEHERE",
      "Condition": {
        "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket-name" }
      }
    }
  ]
}
```

Note that for both the Amazon SNS and Amazon SQS IAM policies, you can specify the `StringLike` condition in the policy, instead of the `ArnLike` condition.

```
"Condition": {
  "StringLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket-name" }
}
```

Example of a key policy that you attach to the associated KMS key if the SQS queue is SSE enabled.

```
{
  "Version": "2012-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

The policy grants Amazon S3 service principal permission for specific KMS actions that are necessary to encrypt messages added to the queue.

For an example of how to attach a policy to a SNS topic or an SQS queue, see [Example Walkthrough 1: Configure a Bucket for Notifications \(Message Destination: SNS Topic and SQS Queue\)](#) (p. 541).

For more information about permissions, see the following topics:

- [Example Cases for Amazon SNS Access Control](#) in the *Amazon Simple Notification Service Developer Guide*
- [Access Control Using AWS Identity and Access Management \(IAM\)](#) in the *Amazon Simple Queue Service Developer Guide*

Example Walkthrough 1: Configure a Bucket for Notifications (Message Destination: SNS Topic and SQS Queue)

Topics

- [Walkthrough Summary](#) (p. 541)
- [Step 1: Create an Amazon SNS Topic](#) (p. 542)
- [Step 2: Create an Amazon SQS Queue](#) (p. 542)
- [Step 3: Add a Notification Configuration to Your Bucket](#) (p. 544)
- [Step 4: Test the Setup](#) (p. 546)

Walkthrough Summary

In this walkthrough you add notification configuration on a bucket requesting Amazon S3 to:

- Publish events of the `s3:ObjectCreated:*` type to an Amazon SQS queue.
- Publish events of the `s3:ReducedRedundancyLostObject` type to an Amazon SNS topic.

For information about notification configuration, see [Configuring Amazon S3 Event Notifications](#) (p. 530).

You can do all these steps using the console, without writing any code. In addition, code examples, using AWS SDKs for Java and .NET are also provided so you can add notification configuration programmatically.

You will do the following in this walkthrough:

1. Create an Amazon SNS topic.

Using the Amazon SNS console, you create an SNS topic and subscribe to the topic so that any events posted to it are delivered to you. You will specify email as the communications protocol. After you create a topic, Amazon SNS will send an email. You must click a link in the email to confirm the topic subscription.

You will attach an access policy to the topic to grant Amazon S3 permission to post messages.

2. Create an Amazon SQS queue.

Using the Amazon SQS console, you create an SQS queue. You can access any messages Amazon S3 sends to the queue programmatically. But for this walkthrough, you will verify notification messages in the console.

You will attach an access policy to the topic to grant Amazon S3 permission to post messages.

3. Add notification configuration to a bucket.

Step 1: Create an Amazon SNS Topic

Follow the steps to create and subscribe to an Amazon Simple Notification Service (Amazon SNS) topic.

1. Using Amazon SNS console create a topic. For instructions, see [Create a Topic](#) in the *Amazon Simple Notification Service Developer Guide*.
2. Subscribe to the topic. For this exercise, use email as the communications protocol. For instructions, see [Subscribe to a Topic](#) in the *Amazon Simple Notification Service Developer Guide*.

You will get email requesting you to confirm your subscription to the topic. Confirm the subscription.

3. Replace the access policy attached to the topic with the following policy. You must update the policy by providing your SNS topic Amazon Resource Name (ARN) and bucket name:

```
{
  "Version": "2008-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:Publish"
      ],
      "Resource": "SNS-topic-ARN",
      "Condition": {
        "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket-name" }
      }
    }
  ]
}
```

4. Note the topic ARN.

The SNS topic you created is another resource in your AWS account, and it has a unique Amazon Resource Name (ARN). You will need this ARN in the next step. The ARN will be of the following format:

```
arn:aws:sns:aws-region:account-id:topic-name
```

Step 2: Create an Amazon SQS Queue

Follow the steps to create and subscribe to an Amazon Simple Queue Service (Amazon SQS) queue.

1. Using the Amazon SQS console, create a queue. For instructions, see [Getting Started with Amazon SQS](#) in the *Amazon Simple Queue Service Developer Guide*.

2. Replace the access policy attached to the queue with the following policy (in the SQS console, you select the queue, and in the **Permissions** tab, click **Edit Policy Document (Advanced)**).

```
{
  "Version": "2012-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SQS:SendMessage"
      ],
      "Resource": "SQS-queue-ARN",
      "Condition": {
        "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket-name" }
      }
    }
  ]
}
```

3. (Optional) If the Amazon SQS queue is server-side encryption (SSE) enabled, add the following policy to the associated custom AWS Key Management Service (AWS KMS) customer master key (CMK). You must add the policy to a custom CMK because the default AWS managed CMK for Amazon SQS cannot be modified. For more information about using SSE for Amazon SQS with AWS KMS, see [Protecting Data Using Server-Side Encryption \(SSE\) and AWS KMS](#).

```
{
  "Version": "2012-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

4. Note the queue ARN.

The SQS queue you created is another resource in your AWS account, and it has a unique Amazon Resource Name (ARN). You will need this ARN in the next step. The ARN will be of the following format:

```
arn:aws:sqs:aws-region:account-id:queue-name
```

Step 3: Add a Notification Configuration to Your Bucket

You can enable bucket notifications either by using the Amazon S3 console or programmatically by using AWS SDKs. Choose any one of the options to configure notifications on your bucket. This section provides code examples using the AWS SDKs for Java and .NET.

Step 3 (option a): Enable Notifications on a Bucket Using the Console

Using the Amazon S3 console, add a notification configuration requesting Amazon S3 to:

- Publish events of the `s3:ObjectCreated:*` type to your Amazon SQS queue.
- Publish events of the `s3:ReducedRedundancyLostObject` type to your Amazon SNS topic.

After you save the notification configuration, Amazon S3 will post a test message, which you will get via email.

For instructions, see [How Do I Enable and Configure Event Notifications for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Step 3 (option b): Enable Notifications on a Bucket Using the AWS SDK for .NET

The following C# code example provides a complete code listing that adds a notification configuration to a bucket. You will need to update the code and provide your bucket name and SNS topic ARN. For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Example

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class EnableNotificationsTest
    {
        private const string bucketName = "**** bucket name ****";
        private const string snsTopic = "**** SNS topic ARN ****";
        private const string sqsQueue = "**** SQS topic ARN ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USSouth2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            EnableNotificationAsync().Wait();
        }

        static async Task EnableNotificationAsync()
        {

```

```
try
{
    PutBucketNotificationRequest request = new PutBucketNotificationRequest
    {
        BucketName = bucketName
    };

    TopicConfiguration c = new TopicConfiguration
    {
        Events = new List<EventType> { EventType.ObjectCreatedCopy },
        Topic = snsTopic
    };
    request.TopicConfigurations = new List<TopicConfiguration>();
    request.TopicConfigurations.Add(c);
    request.QueueConfigurations = new List<QueueConfiguration>();
    request.QueueConfigurations.Add(new QueueConfiguration()
    {
        Events = new List<EventType> { EventType.ObjectCreatedPut },
        Queue = sqsQueue
    });

    PutBucketNotificationResponse response = await
client.PutBucketNotificationAsync(request);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:'{0}' ",
e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown error encountered on server. Message:'{0}' ",
e.Message);
    }
}
}
```

Step 3 (option c): Enable Notifications on a Bucket Using the AWS SDK for Java

The following example shows how to add a notification configuration to a bucket. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 677\)](#).

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.IOException;
import java.util.EnumSet;

public class EnableNotificationOnABucket {

    public static void main(String[] args) throws IOException {
        String bucketName = "**** Bucket name ****";
        Regions clientRegion = Regions.DEFAULT_REGION;
```

```
String snsTopicARN = "*** SNS Topic ARN ***";
String sqsQueueARN = "*** SQS Queue ARN ***";

try {
    AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(clientRegion)
        .build();

    BucketNotificationConfiguration notificationConfiguration = new
    BucketNotificationConfiguration();

    // Add an SNS topic notification.
    notificationConfiguration.addConfiguration("snsTopicConfig",
        new TopicConfiguration(snsTopicARN,
    EnumSet.of(S3Event.ObjectCreated)));

    // Add an SQS queue notification.
    notificationConfiguration.addConfiguration("sqsQueueConfig",
        new QueueConfiguration(sqsQueueARN,
    EnumSet.of(S3Event.ObjectCreated)));

    // Create the notification configuration request and set the bucket
    notification configuration.
    SetBucketNotificationConfigurationRequest request = new
    SetBucketNotificationConfigurationRequest(
        bucketName, notificationConfiguration);
    s3Client.setBucketNotificationConfiguration(request);
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Step 4: Test the Setup

Now you can test the setup by uploading an object to your bucket and verify the event notification in the Amazon SQS console. For instructions, see [Receiving a Message](#) in the *Amazon Simple Queue Service Developer Guide* "Getting Started" section.

Example Walkthrough 2: Configure a Bucket for Notifications (Message Destination: AWS Lambda)

For an example of using Amazon S3 notifications with AWS Lambda, see [Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

Event Message Structure

The notification message that Amazon S3 sends to publish an event is in the JSON format. The following example shows the structure of the JSON message.

Note the following about the example:

- The `eventVersion` key value contains a major and minor version in the form `<major>.<minor>`.

The major version is incremented if Amazon S3 makes a change to the event structure that is not backward compatible. This includes removing a JSON field that is already present or changing how the contents of a field are represented (for example, a date format).

The minor version is incremented if Amazon S3 adds new fields to the event structure. This might occur if new information is provided for some or all existing events, or if new information is provided on only newly introduced event types. Applications should ignore new fields to stay forward compatible with new minor versions of the event structure.

If new event types are introduced but the structure of the event is otherwise unmodified, the event version does not change.

To ensure that your applications can parse the event structure correctly, we recommend that you do an equal-to comparison on the major version number. To ensure that the fields expected by your application are present, we also recommend doing a greater-than-or-equal-to comparison on the minor version.

- The `responseElements` key value is useful if you want to trace a request by following up with AWS Support. Both `x-amz-request-id` and `x-amz-id-2` help Amazon S3 trace an individual request. These values are the same as those that Amazon S3 returns in the response to the request that initiates the events, so they can be used to match the event to the request.
- The `s3` key provides information about the bucket and object involved in the event. The object key name value is URL encoded. For example, "red flower.jpg" becomes "red+flower.jpg" (Amazon S3 returns "application/x-www-form-urlencoded" as the content type in the response).
- The `sequencer` key provides a way to determine the sequence of events. Event notifications are not guaranteed to arrive in the order that the events occurred. However, notifications from events that create objects (PUTs) and delete objects contain a `sequencer`, which can be used to determine the order of events for a given object key.

If you compare the `sequencer` strings from two event notifications on the same object key, the event notification with the greater `sequencer` hexadecimal value is the event that occurred later. If you are using event notifications to maintain a separate database or index of your Amazon S3 objects, you will probably want to compare and store the `sequencer` values as you process each event notification.

Note the following:

- You cannot use `sequencer` to determine order for events on different object keys.
- The `sequencers` can be of different lengths. So to compare these values, you first left pad the shorter value with zeros, and then do a lexicographical comparison.
- The `glacierEventData` key is only visible for `s3:ObjectRestore:Completed` events.
- The `restoreEventData` key contains attributes related to your restore request.

The following example shows version 2.1 of the event message JSON structure, which is the version currently being used by Amazon S3.

```
{
  "Records":[
    {
      "eventVersion":"2.1",
      "eventSource":"aws:s3",
      "awsRegion":"us-west-2",
      "eventTime":The time, in ISO-8601 format, for example, 1970-01-01T00:00:00.000Z,
when Amazon S3 finished processing the request,
      "eventName":"event-type",
      "userIdentity":{
        "principalId":Amazon-customer-ID-of-the-user-who-caused-the-event
      },
      "requestParameters":{
        "sourceIPAddress":ip-address-where-request-came-from
      },
      "responseElements":{
        "x-amz-request-id":Amazon S3 generated request ID,
        "x-amz-id-2":Amazon S3 host that processed the request
      },
      "s3":{
        "s3SchemaVersion":"1.0",
        "configurationId":ID found in the bucket notification configuration,
        "bucket":{
          "name":bucket-name,
          "ownerIdentity":{
            "principalId":Amazon-customer-ID-of-the-bucket-owner
          },
          "arn":bucket-ARN
        },
        "object":{
          "key":object-key,
          "size":object-size,
          "eTag":object eTag,
          "versionId":object version if bucket is versioning-enabled, otherwise
null,
          "sequencer": a string representation of a hexadecimal value used to
determine event sequence,
only used with PUTs and DELETES
        }
      },
      "glacierEventData": {
        "restoreEventData": {
          "lifecycleRestorationExpiryTime": The time, in ISO-8601 format, for
example, 1970-01-01T00:00:00.000Z, of Restore Expiry,
          "lifecycleRestoreStorageClass": Source storage class for restore
        }
      }
    }
  ]
}
```

The following example shows version 2.0 of the event message structure, which is no longer used by Amazon S3.

```
{
  "Records":[
    {
      "eventVersion":"2.0",
      "eventSource":"aws:s3",
      "awsRegion":"us-west-2",
      "eventTime":The time, in ISO-8601 format, for example, 1970-01-01T00:00:00.000Z,
when S3 finished processing the request,
      "eventName":"event-type",
      "userIdentity":{
```

```

    "principalId": "Amazon-customer-ID-of-the-user-who-caused-the-event"
  },
  "requestParameters": {
    "sourceIPAddress": "ip-address-where-request-came-from"
  },
  "responseElements": {
    "x-amz-request-id": "Amazon S3 generated request ID",
    "x-amz-id-2": "Amazon S3 host that processed the request"
  },
  "s3": {
    "s3SchemaVersion": "1.0",
    "configurationId": "ID found in the bucket notification configuration",
    "bucket": {
      "name": "bucket-name",
      "ownerIdentity": {
        "principalId": "Amazon-customer-ID-of-the-bucket-owner"
      },
      "arn": "bucket-ARN"
    },
    "object": {
      "key": "object-key",
      "size": "object-size",
      "eTag": "object eTag",
      "versionId": "object version if bucket is versioning-enabled, otherwise
null",
      "sequencer": "a string representation of a hexadecimal value used to
determine event sequence,
only used with PUTs and DELETES"
    }
  }
}
]
}

```

The following are example messages:

- Test message—When you configure an event notification on a bucket, Amazon S3 sends the following test message:

```

{
  "Service": "Amazon S3",
  "Event": "s3:TestEvent",
  "Time": "2014-10-13T15:57:02.089Z",
  "Bucket": "bucketname",
  "RequestId": "5582815E1AEA5ADF",
  "HostId": "8cLeGAmw098X5cv4Zkwcmo8vvZa3eH3eKxsPzbB9wrR+YstdA6Knx4Ip8EXAMPLE"
}

```

- Example message when an object is created using the PUT request—The following message is an example of a message Amazon S3 sends to publish an s3:ObjectCreated:Put event:

```

{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-west-2",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AIDAJDPLRKLG7UEXAMPLE"
      },
      "requestParameters": {

```

```
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "C3D13FE58DE4C810",
        "x-amz-id-2": "FMMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/
JRWeUWerMUE5JgHvANOjpd"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "mybucket",
          "ownerIdentity": {
            "principalId": "A3NL1KOZZKExample"
          },
          "arn": "arn:aws:s3:::mybucket"
        },
        "object": {
          "key": "HappyFace.jpg",
          "size": 1024,
          "eTag": "d41d8cd98f00b204e9800998ecf8427e",
          "versionId": "096fKKXTRTtl3on89fVO.nfljtsv6qko",
          "sequencer": "0055AED6DCD90281E5"
        }
      }
    }
  ]
}
```


Replication

Replication enables automatic, asynchronous copying of objects across Amazon S3 buckets. Buckets that are configured for object replication can be owned by the same AWS account or by different accounts. You can copy objects between different AWS Regions or within the same Region.

To enable object replication, you use a bucket-level configuration. You add the replication configuration to your source bucket. The minimum configuration must provide the following:

- The destination bucket where you want Amazon S3 to replicate objects
- An AWS Identity and Access Management (IAM) role that Amazon S3 can assume to replicate objects on your behalf

Additional configuration options are available. For more information, see [Additional Replication Configurations](#) (p. 567).

Types of Object Replication

You can replicate objects between different AWS Regions or within the same AWS Region.

- **Cross-Region replication (CRR)** is used to copy objects across Amazon S3 buckets in different AWS Regions.
- **Same-Region replication (SRR)** is used to copy objects across Amazon S3 buckets in the same AWS Region.

When to Use Replication

Replication can help you do the following:

- **Replicate objects while retaining metadata**—Replicating objects via AWS Lambda functions can be useful. However, they don't retain object metadata such as the original object creation time and version IDs. Replication offers a simpler and more automated way to replicate objects that retains this metadata.
- **Replicate objects into different storage classes**—You can use replication to directly put objects into Glacier, DEEP ARCHIVE, or another storage class in the destination bucket. You can also replicate your data to the same storage class and use lifecycle policies on the destination bucket to move your objects to a colder storage class as it ages.
- **Maintain object copies under different ownership**—Regardless of who owns the source object, you can tell Amazon S3 to change replica ownership to the AWS account that owns the destination bucket. This is referred to as the *owner override* option. You can use this option to restrict access to object replicas.

When to Use CRR

Cross-Region replication can help you do the following:

- **Meet compliance requirements**—Although Amazon S3 stores your data across multiple geographically distant Availability Zones by default, compliance requirements might dictate that you store data at even greater distances. Cross-Region replication allows you to replicate data between distant AWS Regions to satisfy these requirements.
- **Minimize latency**—If your customers are in two geographic locations, you can minimize latency in accessing objects by maintaining object copies in AWS Regions that are geographically closer to your users.
- **Increase operational efficiency**—If you have compute clusters in two different AWS Regions that analyze the same set of objects, you might choose to maintain object copies in those Regions.

When to Use SRR

Same-Region replication can help you do the following:

- **Aggregate logs into a single bucket**—If you store logs in multiple buckets or across multiple accounts, you can easily replicate logs into a single, in-Region bucket. This allows for simpler processing of logs by a single account.
- **Configure live replication between developer and test accounts**—If you or your customers have developer and test accounts that use the same data, you can replicate objects between multiple accounts, while maintaining object metadata, by implementing SRR rules.
- **Abide by data sovereignty laws**—Often customers are required to store data in separate AWS accounts while being barred from letting the data leave a certain Region. Same-Region replication can help you back up critical data when compliance regulations don't allow the data to leave your country.

Requirements for Replication

Replication requires the following:

- The source bucket owner must have the source and destination AWS Regions enabled for their account. The destination bucket owner must have the destination Region enabled for their account. For more information about enabling or disabling an AWS Region, see [AWS Regions and Endpoints](#) in the *AWS General Reference*.
- Both source and destination buckets must have versioning enabled.
- Amazon S3 must have permissions to replicate objects from the source bucket to the destination bucket on your behalf.
- If the owner of the source bucket doesn't own the object in the bucket, the object owner must grant the bucket owner `READ` and `READ_ACP` permissions with the object access control list (ACL). For more information, see [Managing Access with ACLs](#) (p. 403).
- If the source bucket has Amazon S3 object lock enabled, the destination bucket must also have object lock enabled. For more information, see [Locking Objects Using Amazon S3 Object Lock](#) (p. 453).

To enable replication on a bucket that has object lock enabled, contact [AWS Support](#).

For more information, see [Overview of Setting Up Replication](#) (p. 555).

If you are setting the replication configuration in a *cross-account scenario*, where source and destination buckets are owned by different AWS accounts, the following additional requirement applies:

- The owner of the destination bucket must grant the owner of the source bucket permissions to replicate objects with a bucket policy. For more information, see [Granting Permissions When Source and Destination Buckets Are Owned by Different AWS Accounts](#) (p. 566).

What Does Amazon S3 Replicate?

Amazon S3 replicates only specific items in buckets that are configured for replication.

What Is Replicated?

Amazon S3 replicates the following:

- Objects created after you add a replication configuration, with exceptions described in the next section.
- Both unencrypted objects and objects encrypted using Amazon S3 managed keys (SSE-S3) or AWS KMS managed keys (SSE-KMS), although you must explicitly enable the option to replicate objects encrypted using KMS keys. The replicated copy of the object is encrypted using the same type of server-side encryption that was used for the source object. For more information about server-side encryption, see [Protecting Data Using Server-Side Encryption](#) (p. 265).
- Object metadata.
- Only objects in the source bucket for which the bucket owner has permissions to read objects and access control lists (ACLs). For more information about resource ownership, see [Amazon S3 Bucket and Object Ownership](#) (p. 302).
- Object ACL updates, unless you direct Amazon S3 to change the replica ownership when source and destination buckets aren't owned by the same accounts. For more information, see [Additional Replication Configuration: Changing the Replica Owner](#) (p. 568)).

It can take a while until Amazon S3 can bring the two ACLs in sync. This applies only to objects created after you add a replication configuration to the bucket.

- Object tags, if there are any.

- Amazon S3 object lock retention information, if there is any. When Amazon S3 replicates objects that have retention information applied, it applies those same retention controls to your replicas, overriding the default retention period configured on your destination bucket. If you don't have retention controls applied to the objects in your source bucket, and you replicate into a destination bucket that has a default retention period set, the destination bucket's default retention period is applied to your object replicas. For more information, see [Locking Objects Using Amazon S3 Object Lock \(p. 453\)](#).

How Delete Operations Affect Replication

If you delete an object from the source bucket, the following occurs:

- If you make a DELETE request without specifying an object version ID, Amazon S3 adds a delete marker. Amazon S3 deals with the delete marker as follows:
 - If you are using the latest version of the replication configuration (that is, you specify the `Filter` element in a replication configuration rule), Amazon S3 does not replicate the delete marker.
 - If you don't specify the `Filter` element, Amazon S3 assumes that the replication configuration is an earlier version V1. In the earlier version, Amazon S3 handled replication of delete markers differently. For more information, see [Backward Compatibility \(p. 563\)](#).
- If you specify an object version ID to delete in a DELETE request, Amazon S3 deletes that object version in the source bucket. But it doesn't replicate the deletion in the destination bucket. In other words, it doesn't delete the same object version from the destination bucket. This protects data from malicious deletions.

What Isn't Replicated?

Amazon S3 doesn't replicate the following:

- Objects that existed before you added the replication configuration to the bucket. In other words, Amazon S3 doesn't replicate objects retroactively.
- The following encrypted objects:
 - Objects created with server-side encryption using customer-provided (SSE-C) encryption keys.
 - Objects created with server-side encryption using AWS KMS managed encryption (SSE-KMS) keys. By default, Amazon S3 does not replicate objects encrypted using KMS keys. However, you can explicitly enable replication of these objects in the replication configuration, and provide relevant information so that Amazon S3 can replicate these objects.

For more information about server-side encryption, see [Protecting Data Using Server-Side Encryption \(p. 265\)](#).

- Objects that are stored in GLACIER or DEEP_ARCHIVE storage class. To learn more about the Amazon S3 Glacier service, see the [Amazon S3 Glacier Developer Guide](#).
- Objects in the source bucket that the bucket owner doesn't have permissions for (when the bucket owner is not the owner of the object). For information about how an object owner can grant permissions to a bucket owner, see [Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control \(p. 377\)](#).

- Updates to bucket-level subresources. For example, if you change the lifecycle configuration or add a notification configuration to your source bucket, these changes are not applied to the destination bucket. This makes it possible to have different configurations on source and destination buckets.
- Actions performed by lifecycle configuration.

For example, if lifecycle configuration is enabled only on your source bucket, Amazon S3 creates delete markers for expired objects but doesn't replicate those markers. If you want the same lifecycle configuration applied to both source and destination buckets, enable the same lifecycle configuration on both.

For more information about lifecycle configuration, see [Object Lifecycle Management \(p. 119\)](#).

Note

If using the latest version of the replication configuration (the XML specifies `Filter` as the child of `Rule`), delete markers created either by a user action or by Amazon S3 as part of the lifecycle action are not replicated. However, if you are using an earlier version of the replication configuration (the XML specifies `Prefix` as the child of `Rule`), delete markers resulting from user actions are replicated. For more information, see [Backward Compatibility \(p. 563\)](#).

- Objects in the source bucket that are replicas that were created by another replication rule.

You can replicate objects from a source bucket to *only one* destination bucket. After Amazon S3 replicates an object, the object can't be replicated again. For example, if you change the destination bucket in an existing replication configuration, Amazon S3 won't replicate the object again.

Another example: Suppose that you configure replication where bucket A is the source and bucket B is the destination. Now suppose that you add another replication configuration where bucket B is the source and bucket C is the destination. In this case, objects in bucket B that are replicas of objects in bucket A are not replicated to bucket C.

Related Topics

[Replication \(p. 551\)](#)

[Overview of Setting Up Replication \(p. 555\)](#)

[Replication Status Information \(p. 594\)](#)

Overview of Setting Up Replication

To enable replication, you simply add a replication configuration to your source bucket. The configuration tells Amazon S3 to replicate objects as specified. In the replication configuration, you must provide the following:

- The destination bucket—The bucket where you want Amazon S3 to replicate the objects.
- The objects that you want to replicate—You can replicate all of the objects in the source bucket or a subset. You identify subset by providing a key name prefix, one or more object tags, or both in the configuration. For example, if you configure a replication rule to replicate only objects with the key name prefix `Tax/`, Amazon S3 replicates objects with keys such as `Tax/doc1` or `Tax/doc2`. But it doesn't replicate an object with the key `Legal/doc3`. If you specify both prefix and one or more tags, Amazon S3 replicates only objects having the specific key prefix and tags.

A replica has the same key names and metadata (for example, creation time, user-defined metadata, and version ID) as the original object. Amazon S3 encrypts all data in transit using Secure Sockets Layer (SSL).

In addition to these minimum requirements, you can choose the following options:

- By default, Amazon S3 stores object replicas using the same storage class as the source object. You can specify a different storage class for the replicas.
- Amazon S3 assumes that an object replica continues to be owned by the owner of the source object. So when it replicates objects, it also replicates the corresponding object access control list (ACL). If the source and destination buckets are owned by different AWS accounts, you can configure replication to change the owner of a replica to the AWS account that owns the destination bucket.

Additional configuration options are available. For more information, see [Additional Replication Configurations](#) (p. 567).

Important

If you have an object expiration lifecycle policy in your non-versioned bucket and you want to maintain the same permanent delete behavior when you enable versioning, you must add a noncurrent expiration policy. The noncurrent expiration lifecycle policy will manage the deletes of the noncurrent object versions in the version-enabled bucket. (A version-enabled bucket maintains one current and zero or more noncurrent object versions.) For more information, see [How Do I Create a Lifecycle Policy for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

Amazon S3 also provides APIs to support setting up replication rules. For more information, see the following topics in the *Amazon Simple Storage Service API Reference*:

- [PUT Bucket replication](#)
- [GET Bucket replication](#)
- [DELETE Bucket replication](#)

Instead of making these API calls directly from your code, you can add a replication configuration to a bucket with the AWS SDK, AWS CLI, or the Amazon S3 console. It's easiest to use the console. For examples with step-by-step instructions, see [Replication Walkthroughs](#) (p. 575).

If you are new to replication configurations, we recommend that you read the following sections before exploring the examples and optional configurations. For examples that provide step-by-step instructions for setting up basic replication configurations, see [Replication Configuration Overview](#) (p. 556).

Topics

- [Replication Configuration Overview](#) (p. 556)
- [Setting Up Permissions for Replication](#) (p. 564)

Replication Configuration Overview

Amazon S3 stores a replication configuration as XML. In the replication configuration XML file, you specify an AWS Identity and Access Management (IAM) role and one or more rules.

```
<ReplicationConfiguration>
  <Role>IAM-role-ARN</Role>
  <Rule>
    ...
  </Rule>
  <Rule>
```

```
...
</Rule>
...
</ReplicationConfiguration>
```

Amazon S3 can't replicate objects without your permission. You grant permissions with the IAM role that you specify in the replication configuration. Amazon S3 assumes the IAM role to replicate objects on your behalf. You must grant the required permissions to the IAM role first. For more information about managing permissions, see [Setting Up Permissions for Replication \(p. 564\)](#).

You add one rule in replication configuration in the following scenarios:

- You want to replicate all objects.
- You want to replicate a subset of objects. You identify the object subset by adding a filter in the rule. In the filter, you specify an object key prefix, tags, or a combination of both, to identify the subset of objects that the rule applies to.

You add multiple rules in a replication configuration if you want to select a different subset of objects. In each rule, you specify a filter that selects a different subset of objects. For example, you might choose to replicate objects that have either `tax/` or `document/` key prefixes. You would add two rules and specify the `tax/` key prefix filter in one rule and the `document/` key prefix in the other.

The following sections provide additional information.

Topics

- [Basic Rule Configuration \(p. 557\)](#)
- [Optional: Specifying a Filter \(p. 558\)](#)
- [Additional Destination Configurations \(p. 559\)](#)
- [Example Replication Configurations \(p. 560\)](#)
- [Backward Compatibility \(p. 563\)](#)

Basic Rule Configuration

Each rule must include the rule's status and priority, and indicate whether to replicate delete markers.

- **Status** indicates whether the rule is enabled or disabled. If a rule is disabled, Amazon S3 doesn't perform the actions specified in the rule.
- **Priority** indicates which rule has priority when multiple rules apply to an object.
- Currently, delete markers aren't replicated, so you must set `DeleteMarkerReplication` to `Disabled`.

In the destination configuration, you must provide the name of the bucket where you want Amazon S3 to replicate objects.

The following code shows the minimum requirements for a rule.

```
...
<Rule>
  <ID>Rule-1</ID>
  <Status>rule-Enabled-or-Disabled</Status>
  <Priority>integer</Priority>
  <DeleteMarkerReplication>
    <Status>Disabled</Status>
  </DeleteMarkerReplication>
  <Destination>
```

```
        <Bucket>arn:aws:s3:::bucket-name</Bucket>
      </Destination>
    </Rule>
    <Rule>
      ...
    </Rule>
    ...
  ...
```

You can also specify other configuration options. For example, you might choose to use a storage class for object replicas that differs from the class for the source object.

Optional: Specifying a Filter

To choose a subset of objects that the rule applies to, add an optional filter. You can filter by object key prefix, object tags, or combination of both. If you filter on both a key prefix and object tags, Amazon S3 combines the filters using a logical AND operator. In other words, the rule applies to a subset of objects with a specific key prefix and specific tags.

To specify a rule with a filter based on an object key prefix, use the following code. You can specify only one prefix.

```
<Rule>
  ...
  <Filter>
    <Prefix>key-prefix</Prefix>
  </Filter>
  ...
</Rule>
...
```

To specify a rule with a filter based on object tags, use the following code. You can specify one or more object tags.

```
<Rule>
  ...
  <Filter>
    <And>
      <Tag>
        <Key>key1</Key>
        <Value>value1</Value>
      </Tag>
      <Tag>
        <Key>key2</Key>
        <Value>value2</Value>
      </Tag>
      ...
    </And>
  </Filter>
  ...
</Rule>
...
```

To specify a rule filter with a combination of a key prefix and object tags, use this code. You wrap these filters in an AND parent element. Amazon S3 performs a logical AND operation to combine these filters. In other words, the rule applies to a subset of objects with a specific key prefix and specific tags.

```
<Rule>
  ...
  <Filter>
    <And>
```



```
        <Prefix>key-prefix</Prefix>
        <Tag>
            <Key>key1</Key>
            <Value>value1</Value>
        </Tag>
        <Tag>
            <Key>key2</Key>
            <Value>value2</Value>
        </Tag>
        ...
    </Filter>
    ...
</Rule>
...
```

Additional Destination Configurations

In the destination configuration, you specify the bucket where you want Amazon S3 to replicate objects. You can set configurations to replicate objects from one source bucket to one destination bucket. If you add multiple rules in a replication configuration, all of the rules must identify the same destination bucket.

```
...
<Destination>
    <Bucket>arn:aws:s3:::destination-bucket</Bucket>
</Destination>
...
```

You can add the following options in the `<Destination>` element:

- You can specify the storage class for the object replicas. By default, Amazon S3 uses the storage class of the source object to create object replicas, as in the following example.

```
...
<Destination>
    <Bucket>arn:aws:s3:::destinationbucket</Bucket>
    <StorageClass>storage-class</StorageClass>
</Destination>
...
```

- When source and destination buckets aren't owned by the same accounts, you can change the ownership of the replica to the AWS account that owns the destination bucket by adding the `AccessControlTranslation` element.

```
...
<Destination>
    <Bucket>arn:aws:s3:::destinationbucket</Bucket>
    <Account>destination-bucket-owner-account-id</Account>
    <AccessControlTranslation>
        <Owner>Destination</Owner>
    </AccessControlTranslation>
</Destination>
...
```

If you don't add this element to the replication configuration, the replicas are owned by same AWS account that owns the source object. For more information, see [Additional Replication Configuration: Changing the Replica Owner \(p. 568\)](#).

- Your source bucket might contain objects that were created with server-side encryption using keys stored in AWS KMS. By default, Amazon S3 doesn't replicate these objects. You can optionally

direct Amazon S3 to replicate these objects by first explicitly opting into this feature by adding the `SourceSelectionCriteria` element and then providing the AWS KMS key (for the AWS Region of the destination bucket) to use for encrypting object replicas.

```
...
<SourceSelectionCriteria>
  <SseKmsEncryptedObjects>
    <Status>Enabled</Status>
  </SseKmsEncryptedObjects>
</SourceSelectionCriteria>
<Destination>
  <Bucket>arn:aws:s3:::dest-bucket-name</Bucket>
  <EncryptionConfiguration>
    <ReplicaKmsKeyID>AWS KMS key IDs to use for encrypting object replicas</
ReplicaKmsKeyID>
  </EncryptionConfiguration>
</Destination>
...
```

For more information, see [Additional Replication Configuration: Replicating Objects Created with Server-Side Encryption \(SSE\) Using Encryption Keys stored in AWS KMS \(p. 570\)](#).

Example Replication Configurations

To get started, you can add the following example replication configurations to your bucket, as appropriate.

Important

To add a replication configuration to a bucket, you must have the `iam:PassRole` permission. This permission allows you to pass the IAM role that grants Amazon S3 replication permissions. You specify the IAM role by providing the Amazon Resource Name (ARN) that is used in the `Role` element in the replication configuration XML. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*.

Example 1: Replication Configuration with One Rule

The following basic replication configuration specifies one rule. The rule specifies an IAM role that Amazon S3 can assume and a destination bucket for object replicas. The rule `Status` indicates that the rule is in effect.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::AcctID:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>

    <Destination><Bucket>arn:aws:s3:::destinationbucket</Bucket></Destination>

  </Rule>
</ReplicationConfiguration>
```

To choose a subset of objects to replicate, you can add a filter. In the following configuration, the filter specifies an object key prefix. This rule applies to objects that have the prefix `Tax/` in their key names.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::AcctID:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
```

```
<Priority>1</Priority>
<DeleteMarkerReplication>
  <Status>string</Status>
</DeleteMarkerReplication>

<Filter>
  <Prefix>Tax</Prefix>
</Filter>

<Destination><Bucket>arn:aws:s3:::destinationbucket</Bucket></Destination>

</Rule>
</ReplicationConfiguration>
```

If you specify the `Filter` element, you must also include the `Priority` and `DeleteMarkerReplication` elements. In this example, priority is irrelevant because there is only one rule.

In the following configuration, the filter specifies one prefix and two tags. The rule applies to the subset of objects that have the specified key prefix and tags. Specifically, it applies to object that have the `Tax` prefix in their key names and the two specified object tags. Priority doesn't apply because there is only one rule.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::AcctID:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>

    <Filter>
      <And>
        <Prefix>Tax</Prefix>
        <Tag>
          <Key>tagA</Key>
          <Value>valueA</Value>
        </Tag>
        <Tag>
          <Key>tagB</Key>
          <Value>valueB</Value>
        </Tag>
      </And>
    </Filter>

    <Destination><Bucket>arn:aws:s3:::destinationbucket</Bucket></Destination>

  </Rule>
</ReplicationConfiguration>
```

You can specify a storage class for the object replicas as follows.

```
<?xml version="1.0" encoding="UTF-8"?>

<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
```

```
<Rule>
  <Status>Enabled</Status>
  <Destination>
    <Bucket>arn:aws:s3:::destinationbucket</Bucket>
    <StorageClass>storage-class</StorageClass>
  </Destination>
</Rule>
</ReplicationConfiguration>
```

You can specify any storage class that Amazon S3 supports.

Example 2: Replication Configuration with Two Rules

Example

In the following replication configuration:

- Each rule filters on a different key prefix so that each rule applies to a distinct subset of objects. Amazon S3 replicates objects with key names `Tax/doc1.pdf` and `Project/project1.txt`, but it doesn't replicate objects with the key name `PersonalDoc/documentA`.
- Rule priority is irrelevant because the rules apply to two distinct sets of objects. The next example shows what happens when rule priority is applied.
- The second rule specifies a storage class for object replicas. Amazon S3 uses the specified storage class for those object replicas.
- Both rules specify the same destination bucket. You can specify only one destination bucket, regardless of how many rules you specify.

```
<?xml version="1.0" encoding="UTF-8"?>

<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>Tax</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Destination>
      <Bucket>arn:aws:s3:::destinationbucket</Bucket>
    </Destination>
    ...
  </Rule>
  <Rule>
    <Status>Enabled</Status>
    <Priority>2</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>Project</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Destination>
      <Bucket>arn:aws:s3:::destinationbucket</Bucket>
      <StorageClass>STANDARD_IA</StorageClass>
    </Destination>
    ...
  </Rule>
</ReplicationConfiguration>
```

```
</Rule>

</ReplicationConfiguration>
```

Example 3: Replication Configuration with Two Rules with Overlapping Prefixes

In this configuration, the two rules specify filters with overlapping key prefixes, `star/` and `starship`. Both rules apply to objects with the key name `starship-x`. In this case, Amazon S3 uses the rule priority to determine which rule to apply.

```
<ReplicationConfiguration>

  <Role>arn:aws:iam::AcctID:role/role-name</Role>

  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>star</Prefix>
    </Filter>
    <Destination>
      <Bucket>arn:aws:s3::destinationbucket</Bucket>
    </Destination>
  </Rule>
  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>starship</Prefix>
    </Filter>
    <Destination>
      <Bucket>arn:aws:s3::destinationbucket</Bucket>
    </Destination>
  </Rule>
</ReplicationConfiguration>
```

Example 4: Example Walkthroughs

For example walkthroughs, see [Replication Walkthroughs \(p. 575\)](#).

For more information about the XML structure of replication configuration, see [PutBucketReplication](#) in the *Amazon Simple Storage Service API Reference*.

Backward Compatibility

The latest version of the replication configuration XML is V2. For backward compatibility, Amazon S3 continues to support the V1 configuration. If you have used replication configuration XML V1, consider the following issues that affect backward compatibility:

- Replication configuration XML V2 includes the `Filter` element for rules. With the `Filter` element, you can specify object filters based on the object key prefix, tags, or both to scope the objects that the rule applies to. Replication configuration XML V1 supported filtering based only on the key prefix. In that case, you add the `Prefix` directly as a child element of the `Rule` element, as in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::AcctID:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
    <Prefix>key-prefix</Prefix>
    <Destination><Bucket>arn:aws:s3::destinationbucket</Bucket></Destination>

  </Rule>
</ReplicationConfiguration>
```

For backward compatibility, Amazon S3 continues to support the V1 configuration.

- When you delete an object from your source bucket without specifying an object version ID, Amazon S3 adds a delete marker. If you use V1 of the replication configuration XML, Amazon S3 replicates delete markers that resulted from user actions. In other words, if the user deleted the object, and not if Amazon S3 deleted it because the object expired as part of lifecycle action. In V2, Amazon S3 doesn't replicate delete markers. Therefore, you must set the `DeleteMarkerReplication` element to Disabled.

```
...
  <Rule>
    <ID>Rule-1</ID>
    <Status>rule-Enabled-or-Disabled</Priority>
    <Priority>integer</Status>
    <DeleteMarkerReplication>
      <Status>Disabled</Status>
    </DeleteMarkerReplication>
    <Destination>
      <Bucket>arn:aws:s3:::bucket-name</Bucket>
    </Destination>
  </Rule>
...
```

Setting Up Permissions for Replication

When setting up replication, you must acquire necessary permissions as follows:

- Create an IAM role—Amazon S3 needs permissions to replicate objects on your behalf. You grant these permissions by creating an IAM role and specify the role in your replication configuration.
- When source and destination buckets aren't owned by the same accounts, the owner of the destination bucket must grant the source bucket owner permissions to store the replicas.

Topics

- [Creating an IAM Role \(p. 564\)](#)
- [Granting Permissions When Source and Destination Buckets Are Owned by Different AWS Accounts \(p. 566\)](#)

Creating an IAM Role

By default, all Amazon S3 resources—buckets, objects, and related subresources—are private: Only the resource owner can access the resource. To read objects from the source bucket and replicate them to the destination bucket, Amazon S3 needs permissions to perform these tasks. You grant these permissions by creating an IAM role and specifying the role in your replication configuration.

This section explains the trust policy and minimum required permissions policy. The example walkthroughs provide step-by-step instructions to create an IAM role. For more information, see [Replication Walkthroughs \(p. 575\)](#).

- The following shows a *trust policy*, where you identify Amazon S3 as the service principal who can assume the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For more information about IAM roles, see [IAM Roles](#) in the *IAM User Guide*.

- The following shows an *access policy*, where you grant the role permissions to perform replication tasks on your behalf. When Amazon S3 assumes the role, it has the permissions that you specify in this policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetReplicationConfiguration",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::source-bucket"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObjectVersion",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws:s3:::source-bucket/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ReplicateObject",
        "s3:ReplicateDelete",
        "s3:ReplicateTags"
      ],
      "Resource": "arn:aws:s3:::destination-bucket/*"
    }
  ]
}
```

```
}  
  ]  
}
```

The access policy grants permissions for the following actions:

- `s3:GetReplicationConfiguration` and `s3:ListBucket`—Permissions for these actions on the *source* bucket allow Amazon S3 to retrieve the replication configuration and list bucket content (the current permissions model requires the `s3:ListBucket` permission for accessing delete markers).
- `s3:GetObjectVersion` and `s3:GetObjectVersionAcl`—Permissions for these actions granted on all objects allow Amazon S3 to get a specific object version and access control list (ACL) associated with objects.
- `s3:ReplicateObject` and `s3:ReplicateDelete`—Permissions for these actions on objects in the *destination* bucket allow Amazon S3 to replicate objects or delete markers to the destination bucket. For information about delete markers, see [How Delete Operations Affect Replication](#) (p. 554).

Note

Permissions for the `s3:ReplicateObject` action on the *destination* bucket also allow replication of object tags, so you don't need to explicitly grant permission for the `s3:ReplicateTags` action.

- `s3:GetObjectVersionTagging`—Permissions for this action on objects in the *source* bucket allow Amazon S3 to read object tags for replication (see [Object Tagging](#) (p. 110)). If Amazon S3 doesn't have these permissions, it replicates the objects, but not the object tags.

For a list of Amazon S3 actions, see [Specifying Permissions in a Policy](#) (p. 345).

Important

The AWS account that owns the IAM role must have permissions for the actions that it grants to the IAM role.

For example, suppose that the source bucket contains objects owned by another AWS account. The owner of the objects must explicitly grant the AWS account that owns the IAM role the required permissions through the object ACL. Otherwise, Amazon S3 can't access the objects, and replication of the objects fails. For information about ACL permissions, see [Access Control List \(ACL\) Overview](#) (p. 403).

The permissions described here are related to minimum replication configuration. If you choose to add optional replication configurations, you must grant additional permissions to Amazon S3. For more information, see [Additional Replication Configurations](#) (p. 567).

Granting Permissions When Source and Destination Buckets Are Owned by Different AWS Accounts

When source and destination buckets aren't owned by the same accounts, the owner of the destination bucket must also add a bucket policy to grant the owner of the source bucket permissions to perform replication actions, as follows.

```
{  
  "Version": "2008-10-17",  
  "Id": "PolicyForDestinationBucket",  
  "Statement": [  
    {  
      "Sid": "1",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "SourceBucket-AcctID"  
      },  
      "Action": [  
        "s3:ReplicateDelete",  
        "s3:ReplicateObject",  
        "s3:ReplicateDelete"  
      ]  
    }  
  ]  
}
```



```
        "s3:ReplicateObject"
      ],
      "Resource": "arn:aws:s3:::destinationbucket/*"
    },
    {
      "Sid": "2",
      "Effect": "Allow",
      "Principal": {
        "AWS": "SourceBucket-AcctID"
      },
      "Action": "s3:List*",
      "Resource": "arn:aws:s3:::destinationbucket"
    }
  ]
}
```

For an example, see [Example 2: Configuring Replication When the Source and Destination Buckets Are Owned by Different Accounts](#) (p. 584).

If objects in the source bucket are tagged, note the following:

- If the source bucket owner grants Amazon S3 permission for the `s3:GetObjectVersionTagging` and `s3:ReplicateTags` actions to replicate object tags (through the IAM role), Amazon S3 replicates the tags along with the objects. For information about the IAM role, see [Creating an IAM Role](#) (p. 564).
- If the owner of the destination bucket doesn't want to replicate the tags, they can add the following statement to the destination bucket policy to explicitly deny permission for the `s3:ReplicateTags` action.

```
...
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::SourceBucket-AcctID:root"
      },
      "Action": ["s3:ReplicateTags"],
      "Resource": "arn:aws:s3:::destinationbucket/*"
    }
  ]
...

```

Changing Replica Ownership

When different AWS accounts own the source and destination buckets, you can tell Amazon S3 to change the ownership of the replica to the AWS account that owns the destination bucket. This is called the *owner override* option. For more information, see [Additional Replication Configuration: Changing the Replica Owner](#) (p. 568).

Additional Replication Configurations

This section describes additional replication configuration options that are available in Amazon S3. For information about core replication configuration, see [Overview of Setting Up Replication](#) (p. 555).

Topics

- [Additional Replication Configuration: Changing the Replica Owner](#) (p. 568)

- [Additional Replication Configuration: Replicating Objects Created with Server-Side Encryption \(SSE\) Using Encryption Keys stored in AWS KMS \(p. 570\)](#)

Additional Replication Configuration: Changing the Replica Owner

In replication, the owner of the source object also owns the replica by default. When source and destination buckets are owned by different AWS accounts, you can add optional configuration settings to change replica ownership to the AWS account that owns the destination bucket. You might do this, for example, to restrict access to object replicas. This is referred to as the *owner override* option of the replication configuration. This section explains only the relevant additional configuration settings. For information about setting the replication configuration, see [Replication \(p. 551\)](#).

To configure the owner override, you do the following:

- Add the owner override option to the replication configuration to tell Amazon S3 to change replica ownership.
- Grant Amazon S3 permissions to change replica ownership.
- Add permission in the destination bucket policy to allow changing replica ownership. This allows the owner of the destination bucket to accept the ownership of object replicas.

The following sections describe how to perform these tasks. For a working example with step-by-step instructions, see [Example 3: Changing the Replica Owner When the Source and Destination Buckets Are Owned by Different Accounts \(p. 585\)](#).

Adding the Owner Override Option to the Replication Configuration

Warning

Add the owner override option only when the source and destination buckets are owned by different AWS accounts. Amazon S3 doesn't check if the buckets are owned by same or different accounts. If you add the owner override when both buckets are owned by same AWS account, Amazon S3 applies the owner override. It grants full permissions to the owner of the destination bucket and doesn't replicate subsequent updates to the source object access control list (ACL). The replica owner can directly change the ACL associated with a replica with a `PUT ACL` request, but not through replication.

To specify the owner override option, add the following to the `Destination` element:

- The `AccessControlTranslation` element, which tells Amazon S3 to change replica ownership
- The `Account` element, which specifies the AWS account of the destination bucket owner

```
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  ...
  <Destination>
    ...
    <AccessControlTranslation>
      <Owner>Destination</Owner>
    </AccessControlTranslation>
    <Account>destination-bucket-owner-account-id</Account>
  </Destination>
</Rule>
</ReplicationConfiguration>
```

The following example replication configuration tells Amazon S3 to replicate objects that have the `Tax` key prefix to the destination bucket and change ownership of the replicas.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <ID>Rule-1</ID>
    <Priority>1</Priority>
    <Status>Enabled</Status>
    <Status>Enabled</Status>
    <DeleteMarkerReplication>
      <Status>Disabled</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>Tax</Prefix>
    </Filter>
    <Destination>
      <Bucket>arn:aws:s3:::destination-bucket</Bucket>
      <Account>destination-bucket-owner-account-id</Account>
      <AccessControlTranslation>
        <Owner>Destination</Owner>
      </AccessControlTranslation>
    </Destination>
  </Rule>
</ReplicationConfiguration>
```

Granting Amazon S3 Permission to Change Replica Ownership

Grant Amazon S3 permissions to change replica ownership by adding permission for the `s3:ObjectOwnerOverrideToBucketOwner` action in the permissions policy associated with the IAM role. This is the IAM role that you specified in the replication configuration that allows Amazon S3 to assume and replicate objects on your behalf.

```
...
{
  "Effect": "Allow",
  "Action": [
    "s3:ObjectOwnerOverrideToBucketOwner"
  ],
  "Resource": "arn:aws:s3:::destination-bucket/*"
}
...
```

Adding Permission in the Destination Bucket Policy to Allow Changing Replica Ownership

The owner of the destination bucket must grant the owner of the source bucket permission to change replica ownership. The owner of the destination bucket grants the owner of the source bucket permission for the `s3:ObjectOwnerOverrideToBucketOwner` action. This allows the destination bucket owner to accept ownership of the object replicas. The following example bucket policy statement shows how to do this.

```
...
{
  "Sid": "1",
  "Effect": "Allow",
  "Principal": { "AWS": "source-bucket-account-id" },
  "Action": [ "s3:ObjectOwnerOverrideToBucketOwner" ],
  "Resource": "arn:aws:s3:::destination-bucket/*"
}
```

```
}  
...
```

Additional Considerations

When you configure the ownership override option, the following considerations apply:

- By default, the owner of the source object also owns the replica. Amazon S3 replicates the object version and the ACL associated with it.

If you add the owner override, Amazon S3 replicates only the object version, not the ACL. In addition, Amazon S3 doesn't replicate subsequent changes to the source object ACL. Amazon S3 sets the ACL on the replica that grants full control to the destination bucket owner.

- When you update a replication configuration to enable, or disable, the owner override, the following occurs.

- If you add the owner override option to the replication configuration:

When Amazon S3 replicates an object version, it discards the ACL that is associated with the source object. Instead, it sets the ACL on the replica, giving full control to the owner of the destination bucket. It doesn't replicate subsequent changes to the source object ACL. However, this ACL change doesn't apply to object versions that were replicated before you set the owner override option. ACL updates on source objects that were replicated before the owner override was set continue to be replicated (because the object and its replicas continue to have the same owner).

- If you remove the owner override option from the replication configuration:

Amazon S3 replicates new objects that appear in the source bucket and the associated ACLs to the destination bucket. For objects that were replicated before you removed the owner override, Amazon S3 doesn't replicate the ACLs because the object ownership change that Amazon S3 made remains in effect. That is, ACLs put on the object version that were replicated when the owner override was set continue to be not replicated.

Additional Replication Configuration: Replicating Objects Created with Server-Side Encryption (SSE) Using Encryption Keys stored in AWS KMS

By default, Amazon S3 doesn't replicate objects that are stored at rest using server-side encryption with keys stored in AWS KMS. This section explains additional configuration you add that you add to direct Amazon S3 to replicate these objects.

For an example with step-by-step instructions, see [Example 4: Replicating Encrypted Objects \(p. 589\)](#). For information about creating a replication configuration, see [Replication \(p. 551\)](#).

Topics

- [Specifying Additional Information in the Replication Configuration](#) (p. 571)
- [Granting Additional Permissions for the IAM Role](#) (p. 572)
- [Granting Additional Permissions for Cross-Account Scenarios](#) (p. 574)
- [AWS KMS Transaction Limit Considerations](#) (p. 575)

Specifying Additional Information in the Replication Configuration

In the replication configuration, you do the following:

- In the `Destination` configuration, add the AWS KMS key that you want Amazon S3 to use to encrypt object replicas.
- Explicitly opt in by enabling replication of objects encrypted using the AWS KMS keys by adding the `SourceSelectionCriteria` element.

```
<ReplicationConfiguration>
  <Rule>
    ...
    <SourceSelectionCriteria>
      <SseKmsEncryptedObjects>
        <Status>Enabled</Status>
      </SseKmsEncryptedObjects>
    </SourceSelectionCriteria>

    <Destination>
      ...
      <EncryptionConfiguration>
        <ReplicaKmsKeyID>AWS KMS key ID for the AWS region of the destination
        bucket.</ReplicaKmsKeyID>
      </EncryptionConfiguration>
    </Destination>
    ...
  </Rule>
</ReplicationConfiguration>
```

Important

The AWS KMS key must have been created in the same AWS Region as the destination bucket. The AWS KMS key *must* be valid. The PUT Bucket replication API doesn't check the validity of AWS KMS keys. If you use an invalid key, you receive the 200 OK status code in response, but replication fails.

The following example shows a replication configuration, which includes optional configuration elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration>
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <ID>Rule-1</ID>
    <Priority>1</Priority>
    <Status>Enabled</Status>
    <DeleteMarkerReplication>
      <Status>Disabled</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>Tax</Prefix>
    </Filter>
```

```
<Destination>
  <Bucket>arn:aws:s3:::destination-bucket</Bucket>
  <EncryptionConfiguration>
    <ReplicaKmsKeyID>The AWS KMS key ID for the AWS region of the destination
    bucket (S3 uses it to encrypt object replicas).</ReplicaKmsKeyID>
  </EncryptionConfiguration>
</Destination>
<SourceSelectionCriteria>
  <SseKmsEncryptedObjects>
    <Status>Enabled</Status>
  </SseKmsEncryptedObjects>
</SourceSelectionCriteria>
</Rule>
</ReplicationConfiguration>
```

This replication configuration has one rule. The rule applies to objects with the `Tax` key prefix. Amazon S3 uses the AWS KMS key ID to encrypt these object replicas.

Granting Additional Permissions for the IAM Role

To replicate objects created using server-side encryption with keys stored in AWS KMS, grant the following additional permissions to the IAM role you specify in the replication configuration. You grant these permissions by updating the permission policy associated with the IAM role:

- Permission for the `s3:GetObjectVersionForReplication` action for source objects. Permission for this action allows Amazon S3 to replicate both unencrypted objects and objects created with server-side encryption using Amazon S3 managed encryption (SSE-S3) keys or AWS KMS managed encryption (SSE-KMS) keys.

Note

We recommend that you use the `s3:GetObjectVersionForReplication` action instead of the `s3:GetObjectVersion` action because it provides Amazon S3 with only the minimum permissions necessary for replication. In addition, permission for the `s3:GetObjectVersion` action allows replication of unencrypted and SSE-S3-encrypted objects, but not of objects created using an AWS KMS managed encryption key.

- Permissions for the following AWS KMS actions:
 - `kms:Decrypt` permissions for the AWS KMS key that was used to encrypt the source object
 - `kms:Encrypt` permissions for the AWS KMS key used to encrypt the object replica

We recommend that you restrict these permissions to specific buckets and objects using AWS KMS condition keys, as shown in the following example policy statements.

```
{
  "Action": ["kms:Decrypt"],
  "Effect": "Allow",
  "Condition": {
    "StringLike": {
      "kms:ViaService": "s3.source-bucket-region.amazonaws.com",
      "kms:EncryptionContext:aws:s3:arn": [
        "arn:aws:s3:::source-bucket-name/key-prefix1*",
      ]
    }
  },
  "Resource": [
    "List of AWS KMS key IDs used to encrypt source objects.",
  ]
},
{
  "Action": ["kms:Encrypt"],
  "Effect": "Allow",
  "Condition": {
```

```

    "StringLike": {
      "kms:ViaService": "s3.destination-bucket-region.amazonaws.com",
      "kms:EncryptionContext:aws:s3:arn": [
        "arn:aws:s3:::destination-bucket-name/key-prefix1*",
      ]
    },
  },
  "Resource": [
    "AWS KMS key IDs (for the AWS region of the destination bucket). S3 uses it to encrypt object replicas",
  ]
}

```

The AWS account that owns the IAM role must have permissions for these AWS KMS actions (`kms:Encrypt` and `kms:Decrypt`) for AWS KMS keys listed in the policy. If the AWS KMS keys are owned by another AWS account, the key owner must grant these permissions to the AWS account that owns the IAM role. For more information about managing access to these keys, see [Using IAM Policies with AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

The following is a complete IAM policy that grants the necessary permissions to replicate unencrypted objects, objects created with server-side encryption using Amazon S3 managed encryption keys, and AWS KMS managed encryption keys.

Note

Objects created with server-side encryption using customer-provided (SSE-C) encryption keys are not replicated.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetReplicationConfiguration",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::source-bucket"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObjectVersionForReplication",
        "s3:GetObjectVersionAcl"
      ],
      "Resource": [
        "arn:aws:s3:::source-bucket/key-prefix1*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ReplicateObject",
        "s3:ReplicateDelete"
      ],
      "Resource": "arn:aws:s3:::destination-bucket/key-prefix1*"
    },
    {
      "Action": [
        "kms:Decrypt"
      ],
    }
  ]
}

```

```

    "Effect": "Allow",
    "Condition": {
      "StringLike": {
        "kms:ViaService": "s3.source-bucket-region.amazonaws.com",
        "kms:EncryptionContext:aws:s3:arn": [
          "arn:aws:s3:::source-bucket-name/key-prefix1*"
        ]
      }
    },
    "Resource": [
      "List of AWS KMS key IDs used to encrypt source objects."
    ]
  },
  {
    "Action": [
      "kms:Encrypt"
    ],
    "Effect": "Allow",
    "Condition": {
      "StringLike": {
        "kms:ViaService": "s3.destination-bucket-region.amazonaws.com",
        "kms:EncryptionContext:aws:s3:arn": [
          "arn:aws:s3:::destination-bucket-name/prefix1*"
        ]
      }
    },
    "Resource": [
      "AWS KMS key IDs (for the AWS region of the destination bucket) to use for encrypting object replicas"
    ]
  }
]
}

```

Granting Additional Permissions for Cross-Account Scenarios

In a cross-account scenario, where *source* and *destination* buckets are owned by different AWS accounts, the AWS KMS key to encrypt object replicas must be a customer master key (CMK). The key owner must grant the source bucket owner permission to use the key.

To grant the source bucket owner permission to use the key (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Encryption keys**.
3. Choose the AWS KMS key.
4. In **Key Policy**, **Key Users**, **External Accounts**, choose **Add External Account**.
5. For the **arn:aws:iam::**, enter the source bucket account ID.
6. Choose **Save Changes**.

To grant the source bucket owner permission to use the key (AWS CLI)

- For information, see [put-key-policy](#) in the *AWS CLI Command Reference*. For information about the underlying API, see [PutKeyPolicy](#) in the *AWS Key Management Service API Reference*.

AWS KMS Transaction Limit Considerations

When you add many new objects with AWS KMS encryption after enabling replication, you might experience throttling (HTTP 503 Slow Down errors). Throttling occurs when the number of AWS KMS transactions per second exceeds the current limit. For more information, see [Limits](#) in the *AWS Key Management Service Developer Guide*.

To request an increase in your AWS KMS API rate limit, contact [AWS Support](#).

Replication Walkthroughs

The following examples show how to configure replication for common use cases. The examples demonstrate replication configuration using the Amazon S3 console, AWS Command Line Interface (AWS CLI), and AWS SDKs (Java and .NET SDK examples are shown). For information about installing and configuring the AWS CLI, see the following topics in the *AWS Command Line Interface User Guide*.

- [Installing the AWS Command Line Interface](#)
- [Configuring the AWS CLI](#) - You must set up at least one profile. If you are exploring cross-account scenarios, set up two profiles.

For information about AWS SDKs, see [AWS SDK for Java](#) and [AWS SDK for .NET](#).

Topics

- [Example 1: Configuring Replication When the Source and Destination Buckets Are Owned by the Same Account \(p. 575\)](#)
- [Example 2: Configuring Replication When the Source and Destination Buckets Are Owned by Different Accounts \(p. 584\)](#)
- [Example 3: Changing the Replica Owner When the Source and Destination Buckets Are Owned by Different Accounts \(p. 585\)](#)
- [Example 4: Replicating Encrypted Objects \(p. 589\)](#)

Example 1: Configuring Replication When the Source and Destination Buckets Are Owned by the Same Account

In this example, you set up replication for source and destination buckets that are owned by the same AWS account. Examples are provided for using the Amazon S3 console, the AWS Command Line Interface (AWS CLI), and the AWS SDK for Java and AWS SDK for .NET.

Configure Replication When Buckets Are Owned by the Same Account (Console)

For step-by-step instructions, see [How Do I Add a Replication Rule to an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*. This topic provides instructions for setting replication configuration when buckets are owned by same and different AWS accounts.

Configure Replication When Buckets Are Owned by the Same Account (AWS CLI)

To use the AWS CLI to set up replication when the source and destination buckets are owned by the same AWS account, you create source and destination buckets, enable versioning on the buckets, create an IAM

role that gives Amazon S3 permission to replicate objects, and add the replication configuration to the source bucket. To verify your setup, you test it.

To set up replication when source and destination buckets are owned by the same AWS account

1. Set a credentials profile for the AWS CLI. In this example, we use the profile name `acctA`. For information about setting credential profiles, see [Named Profiles](#) in the *AWS Command Line Interface User Guide*.

Important

The profile you use for this exercise must have the necessary permissions. For example, in the replication configuration, you specify the IAM role that Amazon S3 can assume. You can do this only if the profile you use has the `iam:PassRole` permission. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*. If you use administrator user credentials to create a named profile, you can perform all the tasks.

2. Create a **source** bucket and enable versioning on it. The following code creates a **source** bucket in the US East (N. Virginia) (us-east-1) Region.

```
aws s3api create-bucket \  
--bucket source \  
--region us-east-1 \  
--profile acctA
```

```
aws s3api put-bucket-versioning \  
--bucket source \  
--versioning-configuration Status=Enabled \  
--profile acctA
```

3. Create a **destination** bucket and enable versioning on it. The following code creates a **destination** bucket in the US West (Oregon) (us-west-2) Region.

Note

To set up replication configuration when both source and destination buckets are in the same AWS account, you use the same profile. This example uses `acctA`. To test replication configuration when the buckets are owned by different AWS accounts, you specify different profiles for each. This example uses the `acctB` profile for the destination bucket.

```
aws s3api create-bucket \  
--bucket destination \  
--region us-west-2 \  
--create-bucket-configuration LocationConstraint=us-west-2 \  
--profile acctA
```

```
aws s3api put-bucket-versioning \  
--bucket destination \  
--versioning-configuration Status=Enabled \  
--profile acctA
```

4. Create an IAM role. You specify this role in the replication configuration that you add to the **source** bucket later. Amazon S3 assumes this role to replicate objects on your behalf. You create an IAM role in two steps:
 - Create a role.
 - Attach a permissions policy to the role.

a. Create the IAM role.

- i. Copy the following trust policy and save it to a file named `S3-role-trust-policy.json` in the current directory on your local computer. This policy grants Amazon S3 service principal permissions to assume the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- ii. Run the following command to create a role.

```
$ aws iam create-role \
--role-name replicationRole \
--assume-role-policy-document file://s3-role-trust-policy.json \
--profile acctA
```

b. Attach a permissions policy to the role.

- i. Copy the following permissions policy and save it to a file named `S3-role-permissions-policy.json` in the current directory on your local computer. This policy grants permissions for various Amazon S3 bucket and object actions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObjectVersionForReplication",
        "s3:GetObjectVersionAcl"
      ],
      "Resource": [
        "arn:aws:s3:::source-bucket/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetReplicationConfiguration"
      ],
      "Resource": [
        "arn:aws:s3:::source-bucket"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ReplicateObject",
        "s3:ReplicateDelete",
        "s3:ReplicateTags"
      ]
    }
  ]
}
```

```
        "s3:GetObjectVersionTagging"
      ],
      "Resource": "arn:aws:s3:::destination-bucket/*"
    }
  ]
}
```

- ii. Run the following command to create a policy and attach it to the role.

```
$ aws iam put-role-policy \
--role-name replicationRole \
--policy-document file://s3-role-permissions-policy.json \
--policy-name replicationRolePolicy \
--profile acctA
```

5. Add replication configuration to the *source* bucket.

- a. Although the Amazon S3 API requires replication configuration as XML, the AWS CLI requires that you specify the replication configuration as JSON. Save the following JSON in a file called `replication.json` to the local directory on your computer.

```
{
  "Role": "IAM-role-ARN",
  "Rules": [
    {
      "Status": "Enabled",
      "Priority": 1,
      "DeleteMarkerReplication": { "Status": "Disabled" },
      "Filter" : { "Prefix": "Tax"},
      "Destination": {
        "Bucket": "arn:aws:s3:::destination-bucket"
      }
    }
  ]
}
```

- b. Update the JSON by providing values for the *destination-bucket* and *IAM-role-ARN*. Save the changes.
- c. Run the following command to add the replication configuration to your source bucket. Be sure to provide the *source* bucket name.

```
$ aws s3api put-bucket-replication \
--replication-configuration file://replication.json \
--bucket source \
--profile acctA
```

To retrieve the replication configuration, use the `get-bucket-replication` command.

```
$ aws s3api get-bucket-replication \
--bucket source \
--profile acctA
```

6. Test the setup in the Amazon S3 console:

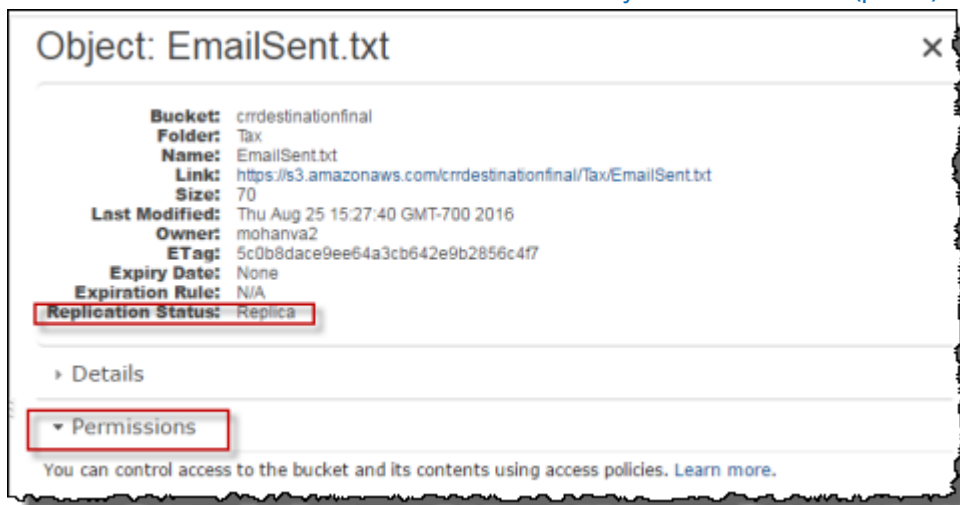
- a. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- b. In the *source* bucket, create a folder named `Tax`.
- c. Add sample objects to the `Tax` folder in the *source* bucket.

Note

The amount of time it takes for Amazon S3 to replicate an object depends on the size of the object. For information about how to see the status of replication, see [Replication Status Information](#) (p. 594).

In the *destination* bucket, verify the following:

- That Amazon S3 replicated the objects.
- In object **properties**, that the **Replication Status** is set to *Replica* (identifying this as a replica object).
- In object **properties**, that the permission section shows no permissions. This means that the replica is still owned by the *source* bucket owner, and the *destination* bucket owner has no permission on the object replica. You can add optional configuration to tell Amazon S3 to change the replica ownership. For an example, see [Example 3: Changing the Replica Owner When the Source and Destination Buckets Are Owned by Different Accounts](#) (p. 585).



- d. Update an object's ACL in the *source* bucket and verify that changes appear in the *destination* bucket.

For instructions, see [How Do I Set Permissions on an Object?](#) in the *Amazon Simple Storage Service Console User Guide*.

Configure Replication When Buckets Are Owned by the Same Account (AWS SDK)

Use the following code examples to add a replication configuration to a bucket with the AWS SDK for Java and AWS SDK for .NET, respectively.

Java

The following example adds a replication configuration to a bucket and then retrieves and verifies the configuration. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples](#) (p. 677).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
```

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateRoleRequest;
import com.amazonaws.services.identitymanagement.model.PutRolePolicyRequest;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketReplicationConfiguration;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.DeleteMarkerReplication;
import com.amazonaws.services.s3.model.DeleteMarkerReplicationStatus;
import com.amazonaws.services.s3.model.ReplicationDestinationConfig;
import com.amazonaws.services.s3.model.ReplicationRule;
import com.amazonaws.services.s3.model.ReplicationRuleStatus;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;
import com.amazonaws.services.s3.model.StorageClass;
import com.amazonaws.services.s3.model.replication.ReplicationFilter;
import com.amazonaws.services.s3.model.replication.ReplicationFilterPredicate;
import com.amazonaws.services.s3.model.replication.ReplicationPrefixPredicate;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CrossRegionReplication {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String accountId = "*** Account ID ***";
        String roleName = "*** Role name ***";
        String sourceBucketName = "*** Source bucket name ***";
        String destBucketName = "*** Destination bucket name ***";
        String prefix = "Tax/";

        String roleARN = String.format("arn:aws:iam::%s:role/%s", accountId, roleName);
        String destinationBucketARN = "arn:aws:s3::" + destBucketName;

        AmazonS3 s3Client = AmazonS3Client.builder()
            .withCredentials(new ProfileCredentialsProvider())
            .withRegion(clientRegion)
            .build();

        createBucket(s3Client, clientRegion, sourceBucketName);
        createBucket(s3Client, clientRegion, destBucketName);
        assignRole(roleName, clientRegion, sourceBucketName, destBucketName);

        try {

            // Create the replication rule.
            List<ReplicationFilterPredicate> andOperands = new
            ArrayList<ReplicationFilterPredicate>();
            andOperands.add(new ReplicationPrefixPredicate(prefix));

            Map<String, ReplicationRule> replicationRules = new HashMap<String,
            ReplicationRule>();
            replicationRules.put("ReplicationRule1",
                new ReplicationRule()
                    .withPriority(0)
                    .withStatus(ReplicationRuleStatus.Enabled)
                    .withDeleteMarkerReplication(new
            DeleteMarkerReplication().withStatus(DeleteMarkerReplicationStatus.DISABLED))
```

```
        .withFilter(new ReplicationFilter().withPredicate(new
ReplicationPrefixPredicate(prefix)))
        .withDestinationConfig(new ReplicationDestinationConfig()
            .withBucketARN(destinationBucketARN)
            .withStorageClass(StorageClass.Standard)));

    // Save the replication rule to the source bucket.
    s3Client.setBucketReplicationConfiguration(sourceBucketName,
        new BucketReplicationConfiguration()
            .withRoleARN(roleARN)
            .withRules(replicationRules));

    // Retrieve the replication configuration and verify that the configuration
    // matches the rule we just set.
    BucketReplicationConfiguration replicationConfig =
s3Client.getBucketReplicationConfiguration(sourceBucketName);
    ReplicationRule rule = replicationConfig.getRule("ReplicationRule1");
    System.out.println("Retrieved destination bucket ARN: " +
rule.getDestinationConfig().getBucketARN());
    System.out.println("Retrieved priority: " + rule.getPriority());
    System.out.println("Retrieved source-bucket replication rule status: " +
rule.getStatus());
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}

private static void createBucket(AmazonS3 s3Client, Regions region, String
bucketName) {
    CreateBucketRequest request = new CreateBucketRequest(bucketName,
region.getName());
    s3Client.createBucket(request);
    BucketVersioningConfiguration configuration = new
BucketVersioningConfiguration().withStatus(BucketVersioningConfiguration.ENABLED);

    SetBucketVersioningConfigurationRequest enableVersioningRequest = new
SetBucketVersioningConfigurationRequest(bucketName, configuration);
    s3Client.setBucketVersioningConfiguration(enableVersioningRequest);

}

private static void assignRole(String roleName, Regions region, String
sourceBucket, String destinationBucket) {
    AmazonIdentityManagement iamClient =
AmazonIdentityManagementClientBuilder.standard()
        .withRegion(region)
        .withCredentials(new ProfileCredentialsProvider())
        .build();
    StringBuilder trustPolicy = new StringBuilder();
    trustPolicy.append("{\r\n    ");
    trustPolicy.append("\t\t\"Version\": \"2012-10-17\", \r\n    ");
    trustPolicy.append("\t\t\"Statement\": [\r\n        {\r\n            ");
    trustPolicy.append("\t\t\t\"Effect\": \"Allow\", \r\n            ");
    trustPolicy.append("\t\t\t\"Principal\": {\r\n                ");
    trustPolicy.append("\t\t\t\t\"Service\": \"s3.amazonaws.com\", \r\n                ");
    trustPolicy.append("\t\t\t\t\"Action\": \"sts:AssumeRole\" \r\n            ");
    trustPolicy.append("\t\t\t\t] \r\n        } \r\n    ");
}
```

```

CreateRoleRequest createRoleRequest = new CreateRoleRequest()
    .withRoleName(roleName)
    .withAssumeRolePolicyDocument(trustPolicy.toString());

iamClient.createRole(createRoleRequest);

StringBuilder permissionPolicy = new StringBuilder();
permissionPolicy.append("{\r\n    \"Version\":\"2012-10-17\",\r\n    \"Statement\":[\r\n        {\r\n            \"Effect\":\"Allow\",\r\n            \"Action\":\r\n                \"s3:GetObjectVersionForReplication\",\r\n                \"s3:GetObjectVersionAcl\",\r\n                \"Resource\":\r\n                    \"arn:aws:s3:::\r\n                        /*\r\n                            ]\r\n                        },\r\n                        {\r\n\r\n            \"Effect\":\"Allow\",\r\n            \"Action\":\r\n                \"s3:ListBucket\",\r\n                \"s3:GetReplicationConfiguration\",\r\n                \"Resource\":\r\n                    \"arn:aws:s3:::\r\n                        sourceBucket\",\r\n                        \"s3:ReplicateObject\",\r\n                        \"s3:ReplicateDelete\",\r\n                        \"s3:ReplicateTags\",\r\n                        \"s3:GetObjectVersionTagging\",\r\n                    \"Resource\":\"arn:aws:s3:::\r\n                        destinationBucket\",\r\n                    /*\r\n                        }\r\n                    }\r\n            }\r\n        }\r\n    ]\r\n}");

PutRolePolicyRequest putRolePolicyRequest = new PutRolePolicyRequest()
    .withRoleName(roleName)
    .withPolicyDocument(permissionPolicy.toString())
    .withPolicyName("crrRolePolicy");

iamClient.putRolePolicy(putRolePolicyRequest);
}
}

```

C#

The following AWS SDK for .NET code example adds a replication configuration to a bucket and then retrieves it. To use this code, provide the names for your buckets and the Amazon Resource Name (ARN) for your IAM role. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

```



```
namespace Amazon.DocSamples.S3
{
    class CrossRegionReplicationTest
    {
        private const string sourceBucket = "*** source bucket ***";
        // Bucket ARN example - arn:aws:s3:::destinationbucket
        private const string destinationBucketArn = "*** destination bucket ARN ***";
        private const string roleArn = "*** IAM Role ARN ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint sourceBucketRegion =
        RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        public static void Main()
        {
            s3Client = new AmazonS3Client(sourceBucketRegion);
            EnableReplicationAsync().Wait();
        }
        static async Task EnableReplicationAsync()
        {
            try
            {
                ReplicationConfiguration replConfig = new ReplicationConfiguration
                {
                    Role = roleArn,
                    Rules =
                    {
                        new ReplicationRule
                        {
                            Prefix = "Tax",
                            Status = ReplicationRuleStatus.Enabled,
                            Destination = new ReplicationDestination
                            {
                                BucketArn = destinationBucketArn
                            }
                        }
                    }
                };

                PutBucketReplicationRequest putRequest = new
                PutBucketReplicationRequest
                {
                    BucketName = sourceBucket,
                    Configuration = replConfig
                };

                PutBucketReplicationResponse putResponse = await
                s3Client.PutBucketReplicationAsync(putRequest);

                // Verify configuration by retrieving it.
                await RetrieveReplicationConfigurationAsync(s3Client);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when
                writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
                writing an object", e.Message);
            }
        }
        private static async Task RetrieveReplicationConfigurationAsync(IAmazonS3
        client)
        {
            // Retrieve the configuration.
        }
    }
}
```

```
GetBucketReplicationRequest getRequest = new GetBucketReplicationRequest
{
    BucketName = sourceBucket
};
GetBucketReplicationResponse getResponse = await
client.GetBucketReplicationAsync(getRequest);
// Print.
Console.WriteLine("Printing replication configuration information...");
Console.WriteLine("Role ARN: {0}", getResponse.Configuration.Role);
foreach (var rule in getResponse.Configuration.Rules)
{
    Console.WriteLine("ID: {0}", rule.Id);
    Console.WriteLine("Prefix: {0}", rule.Prefix);
    Console.WriteLine("Status: {0}", rule.Status);
}
}
}
```

Example 2: Configuring Replication When the Source and Destination Buckets Are Owned by Different Accounts

Setting up replication when *source* and *destination* buckets are owned by different AWS accounts is similar to setting replication when both buckets are owned by the same account. The only difference is that the *destination* bucket owner must grant the *source* bucket owner permission to replicate objects by adding a bucket policy.

To configure replication when the source and destination buckets are owned by different AWS accounts

1. In this example, you create *source* and *destination* buckets in two different AWS accounts. You need to have two credential profiles set for the AWS CLI (in this example, we use *acctA* and *acctB* for profile names). For more information about setting credential profiles, see [Named Profiles](#) in the *AWS Command Line Interface User Guide*.
2. Follow the step-by-step instructions in [Example 1: Configuring for Buckets in the Same Account \(p. 575\)](#) with the following changes:
 - For all AWS CLI commands related to *source* bucket activities (for creating the *source* bucket, enabling versioning, and creating the IAM role), use the *acctA* profile. Use the *acctB* profile to create the *destination* bucket.
 - Make sure that the permissions policy specifies the *source* and *destination* buckets that you created for this example.
3. In the console, add the following bucket policy on the *destination* bucket to allow the owner of the *source* bucket to replicate objects. Be sure to edit the policy by providing the AWS account ID of the *source* bucket owner and the *destination* bucket name.

```
{
  "Version": "2008-10-17",
  "Id": "",
  "Statement": [
    {
      "Sid": "Stmt123",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::source-bucket-owner-AWS-acct-ID:root"
      }
    }
  ]
}
```

```
    },  
    "Action":["s3:ReplicateObject", "s3:ReplicateDelete"],  
    "Resource":["arn:aws:s3:::destination/*"]  
  }  
]  
}
```

Choose the bucket and add the bucket policy. For instructions, see [How Do I Add an S3 Bucket Policy?](#) in the *Amazon Simple Storage Service Console User Guide*.

Example 3: Changing the Replica Owner When the Source and Destination Buckets Are Owned by Different Accounts

When the *source* and *destination* buckets in a replication configuration are owned by different AWS accounts, you can tell Amazon S3 to change replica ownership to the AWS account that owns the *destination* bucket. This example explains how to use the Amazon S3 console and the AWS CLI to change replica ownership. For more information, see [Additional Replication Configuration: Changing the Replica Owner](#) (p. 568).

Change the Replica Owner When Buckets Are Owned by Different Accounts (Console)

For step-by-step instructions, see [Configuring a Replication Rule When the Destination Bucket is in a Different AWS Account](#) in the *Amazon Simple Storage Service Console User Guide*.

Change the Replica Owner When Buckets Are Owned by Different Accounts (AWS CLI)

To change replica ownership using the AWS CLI, you create buckets, enable versioning on the buckets, create an IAM role that gives Amazon S3 permission to replicate objects, and add the replication configuration to the source bucket. In the replication configuration you direct Amazon S3 to change replica owner. You also test the setup.

To change replica ownership when source and destination buckets are owned by different AWS accounts (AWS CLI)

1. In this example, you create the *source* and *destination* buckets in two different AWS accounts. Configure the AWS CLI with two named profiles. This example uses profiles named *acctA* and *acctB*, respectively. For more information about setting credential profiles, see [Named Profiles](#) in the *AWS Command Line Interface User Guide*.

Important

The profiles you use for this exercise must have the necessary permissions. For example, in the replication configuration, you specify the IAM role that Amazon S3 can assume. You can do this only if the profile you use has the `iam:PassRole` permission. If you use administrator user credentials to create a named profile then you can perform all the tasks. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*.

You will need to make sure these profiles have necessary permissions. For example, the replication configuration includes an IAM role that Amazon S3 can assume. The named profile you use to attach such configuration to a bucket can do so only if it has the `iam:PassRole` permission. If you specify administrator user credentials when creating these named profiles, they have all the permissions.

For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*.

2. Create the *source* bucket and enable versioning. This example creates the *source* bucket in the US East (N. Virginia) (us-east-1) Region.

```
aws s3api create-bucket \  
--bucket source \  
--region us-east-1 \  
--profile acctA
```

```
aws s3api put-bucket-versioning \  
--bucket source \  
--versioning-configuration Status=Enabled \  
--profile acctA
```

3. Create a *destination* bucket and enable versioning. This example creates the *destination* bucket in the US West (Oregon) (us-west-2) Region. Use an AWS account profile different from the one you used for the *source* bucket.

```
aws s3api create-bucket \  
--bucket destination \  
--region us-west-2 \  
--create-bucket-configuration LocationConstraint=us-west-2 \  
--profile acctB
```

```
aws s3api put-bucket-versioning \  
--bucket destination \  
--versioning-configuration Status=Enabled \  
--profile acctB
```

4. You must add permissions to your *destination* bucket policy to allow changing the replica ownership.
 - a. Save the following policy to *destination-bucket-policy.json*

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "<destination_bucket_policy_sid>",  
      "Principal": {  
        "AWS": "<src_account>"  
      },  
      "Action": [  
        "s3:ReplicateObject",  
        "s3:ReplicateDelete"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:partition:s3:::<destination_bucket_name>/*"  
      ]  
    }  
  ]  
}
```

- b. Put the above policy to *destination* bucket:

```
aws s3api put-bucket-policy --region ${destination_region} --bucket  
${destination_bucket_name} --policy file://destination_bucket_policy.json
```

5. Create an IAM role. You specify this role in the replication configuration that you add to the *source* bucket later. Amazon S3 assumes this role to replicate objects on your behalf. You create an IAM role in two steps:

- Create a role.
- Attach a permissions policy to the role.

- a. Create an IAM role.

- i. Copy the following trust policy and save it to a file named `s3-role-trust-policy.json` in the current directory on your local computer. This policy grants Amazon S3 permissions to assume the role.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "s3.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```

- ii. Run the following AWS CLI command to create a role.

```
$ aws iam create-role \  
--role-name replicationRole \  
--assume-role-policy-document file://s3-role-trust-policy.json \  
--profile acctA
```

- b. Attach a permissions policy to the role.

- i. Copy the following permissions policy and save it to a file named `s3-role-perm-pol-changeowner.json` in the current directory on your local computer. This policy grants permissions for various Amazon S3 bucket and object actions. In the following steps, you create an IAM role and attach this policy to the role.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:GetObjectVersionForReplication",  
        "s3:GetObjectVersionAcl"  
      ],  
      "Resource": [  
        "arn:aws:s3:::source/*"  
      ]  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  

```

```
        "s3:ListBucket",
        "s3:GetReplicationConfiguration"
    ],
    "Resource": [
        "arn:aws:s3:::source"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:ReplicateObject",
        "s3:ReplicateDelete",
        "s3:ObjectOwnerOverrideToBucketOwner",
        "s3:ReplicateTags",
        "s3:GetObjectVersionTagging"
    ],
    "Resource": "arn:aws:s3:::destination/*"
}
]
```

- ii. To create a policy and attach it to the role, run the following command.

```
$ aws iam put-role-policy \
--role-name replicationRole \
--policy-document file://s3-role-perm-pol-changeowner.json \
--policy-name replicationRolechangeownerPolicy \
--profile acctA
```

6. Add a replication configuration to your source bucket.

- a. The AWS CLI requires specifying the replication configuration as JSON. Save the following JSON in a file named `replication.json` in the current directory on your local computer. In the configuration, the addition of `AccessControlTranslation` to indicate change in replica ownership.

```
{
  "Role": "IAM-role-ARN",
  "Rules": [
    {
      "Status": "Enabled",
      "Priority": "1",
      "DeleteMarkerReplication": {
        "Status": "Disabled"
      },
      "Filter": {
        "Prefix": "Tax"
      },
      "Status": "Enabled",
      "Destination": {
        "Bucket": "arn:aws:s3:::destination",
        "Account": "destination-bucket-owner-account-id",
        "AccessControlTranslation": {
          "Owner": "Destination"
        }
      }
    }
  ]
}
```

- b. Edit the JSON by providing values for the *destination* bucket owner account ID and *IAM-role-ARN*. Save the changes.

- c. To add the replication configuration to the source bucket, run the following command. Provide the **source** bucket name.

```
$ aws s3api put-bucket-replication \
--replication-configuration file://replication-changeowner.json \
--bucket source \
--profile acctA
```

7. Check replica ownership in the Amazon S3 console.
 - a. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
 - b. In the **source** bucket, create a folder named Tax.
 - c. Add objects to the folder in the **source** bucket. Verify that the **destination** bucket contains the object replicas and that the ownership of the replicas has changed to the AWS account that owns the **destination** bucket.

Change the Replica Owner When Buckets Are Owned by Different Accounts (AWS SDK)

For a code example to add replication configuration, see [Configure Replication When Buckets Are Owned by the Same Account \(AWS SDK\)](#) (p. 579). You need to modify the replication configuration appropriately. For conceptual information, see [Additional Replication Configuration: Changing the Replica Owner](#) (p. 568).

Example 4: Replicating Encrypted Objects

By default, Amazon S3 doesn't replicate objects that are stored at rest using server-side encryption with AWS KMS-managed keys. To replicate encrypted objects, you modify the bucket replication configuration to tell Amazon S3 to replicate these objects. This example explains how to use the Amazon S3 console and the AWS Command Line Interface (AWS CLI) to change the bucket replication configuration to enable replicating encrypted objects. For more information, see [Additional Replication Configuration: Replicating Objects Created with Server-Side Encryption \(SSE\) Using Encryption Keys stored in AWS KMS](#) (p. 570).

Replicate Encrypted Objects (Console)

For step-by-step instructions, see [How Do I Add a Replication Rule to an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*. This topic provides instructions for setting replication configuration when buckets are owned by same and different AWS accounts.

Replicate Encrypted Objects (AWS CLI)

To replicate encrypted objects with the AWS CLI, you create buckets, enable versioning on the buckets, create an IAM role that gives Amazon S3 permission to replicate objects, and add the replication configuration to the source bucket. The replication configuration provides information related to replicating objects encrypted using KMS keys. The IAM role permissions include necessary permissions to replicate the encrypted objects. You also test the setup.

To replicate encrypted objects (AWS CLI)

1. In this example, we create both the **source** and **destination** buckets in the same AWS account. Set a credentials profile for the AWS CLI. In this example, we use the profile name acctA. For more information about setting credential profiles, see [Named Profiles](#) in the AWS Command Line Interface User Guide.

2. Create the **source** bucket and enable versioning on it. In this example, we create the **source** bucket in the US East (N. Virginia) (us-east-1) Region.

```
aws s3api create-bucket \
--bucket source \
--region us-east-1 \
--profile acctA
```

```
aws s3api put-bucket-versioning \
--bucket source \
--versioning-configuration Status=Enabled \
--profile acctA
```

3. Create the **destination** bucket and enable versioning on it. In this example, we create the **destination** bucket in the US West (Oregon) (us-west-2) Region.

Note

To set up replication configuration when both **source** and **destination** buckets are in the same AWS account, you use the same profile. In this example, we use acctA. To test replication configuration when the buckets are owned by different AWS accounts, you specify different profiles for each.

```
aws s3api create-bucket \
--bucket destination \
--region us-west-2 \
--create-bucket-configuration LocationConstraint=us-west-2 \
--profile acctA
```

```
aws s3api put-bucket-versioning \
--bucket destination \
--versioning-configuration Status=Enabled \
--profile acctA
```

4. Create an IAM role. You specify this role in the replication configuration that you add to the **source** bucket later. Amazon S3 assumes this role to replicate objects on your behalf. You create an IAM role in two steps:

- Create a role
- Attach a permissions policy to the role

a. Create an IAM role.

- i. Copy the following trust policy and save it to a file called `s3-role-trust-policy-kmsobj.json` in the current directory on your local computer. This policy grants Amazon S3 service principal permissions to assume the role so Amazon S3 can perform tasks on your behalf.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```



```
}
```

- ii. Create a role:

```
$ aws iam create-role \  
--role-name replicationRolekmsobj \  
--assume-role-policy-document file://s3-role-trust-policy-kmsobj.json \  
--profile acctA
```

- b. Attach a permissions policy to the role. This policy grants permissions for various Amazon S3 bucket and object actions.
- i. Copy the following permissions policy and save it to a file named `s3-role-permissions-policykmsobj.json` in the current directory on your local computer. You create an IAM role and attach the policy to it later.

Important

In the permissions policy, you specify the AWS KMS key IDs that will be used for encryption of *source* and *destination* buckets. You must create two separate KMS keys for the *source* and *destination* buckets. KMS keys are never shared outside the AWS Region in which they were created.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "s3:ListBucket",  
        "s3:GetReplicationConfiguration",  
        "s3:GetObjectVersionForReplication",  
        "s3:GetObjectVersionAcl"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:aws:s3:::source",  
        "arn:aws:s3:::source/*"  
      ]  
    },  
    {  
      "Action": [  
        "s3:ReplicateObject",  
        "s3:ReplicateDelete",  
        "s3:ReplicateTags",  
        "s3:GetObjectVersionTagging"  
      ],  
      "Effect": "Allow",  
      "Condition": {  
        "StringLikeIfExists": {  
          "s3:x-amz-server-side-encryption": [  
            "aws:kms",  
            "AES256"  
          ],  
          "s3:x-amz-server-side-encryption-aws-kms-key-id": [  
            "AWS KMS key IDs(in ARN format) to use for encrypting object  
replicas"  
          ]  
        }  
      },  
      "Resource": "arn:aws:s3:::destination/*"  
    },  
    {  
      "Action": [  
        "kms:Decrypt"  
      ]  
    }  
  ]  
}
```

```
    ],
    "Effect": "Allow",
    "Condition": {
      "StringLike": {
        "kms:ViaService": "s3.us-east-1.amazonaws.com",
        "kms:EncryptionContext:aws:s3:arn": [
          "arn:aws:s3:::source/*"
        ]
      }
    },
    "Resource": [
      "AWS KMS key IDs(in ARN format) used to encrypt source objects."
    ]
  },
  {
    "Action": [
      "kms:Encrypt"
    ],
    "Effect": "Allow",
    "Condition": {
      "StringLike": {
        "kms:ViaService": "s3.us-west-2.amazonaws.com",
        "kms:EncryptionContext:aws:s3:arn": [
          "arn:aws:s3:::destination/*"
        ]
      }
    },
    "Resource": [
      "AWS KMS key IDs(in ARN format) to use for encrypting object  
replicas"
    ]
  }
]
```

- ii. Create a policy and attach it to the role:

```
$ aws iam put-role-policy \
--role-name replicationRolekmsobj \
--policy-document file://s3-role-permissions-policykmsobj.json \
--policy-name replicationRolechangeownerPolicy \
--profile acctA
```

5. Add the following replication configuration to the *source* bucket. It tells Amazon S3 to replicate objects with the *Tax/* prefix to the *destination* bucket.

Important

In the replication configuration you specify the IAM role that Amazon S3 can assume. You can do this only if you have the `iam:PassRole` permission. The profile you specify in the CLI command must have the permission. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*.

```
<ReplicationConfiguration>
<Role>IAM-Role-ARN</Role>
<Rule>
  <Status>Enabled</Status>
  <Priority>1</Priority>
  <DeleteMarkerReplication>
    <Status>Disabled</Status>
  </DeleteMarkerReplication>
  <Filter>
    <Prefix>Tax</Prefix>
  </Filter>
  <Status>Enabled</Status>
```

```
<SourceSelectionCriteria>
  <SseKmsEncryptedObjects>
    <Status>Enabled</Status>
  </SseKmsEncryptedObjects>
</SourceSelectionCriteria>
<Destination>
  <Bucket>arn:aws:s3:::dest-bucket-name</Bucket>
  <EncryptionConfiguration>
    <ReplicaKmsKeyID>AWS KMS key IDs to use for encrypting object replicas</
ReplicaKmsKeyID>
  </EncryptionConfiguration>
</Destination>
</Rule>
</ReplicationConfiguration>
```

To add replication configuration to the *source* bucket, do the following:

- a. The AWS CLI requires you to specify the replication configuration as JSON. Save the following JSON in a file (`replication.json`) in the current directory on your local computer.

```
{
  "Role": "IAM-Role-ARN",
  "Rules": [
    {
      "Status": "Enabled",
      "Priority": 1,
      "DeleteMarkerReplication": {
        "Status": "Disabled"
      },
      "Filter": {
        "Prefix": "Tax"
      },
      "Destination": {
        "Bucket": "arn:aws:s3:::destination",
        "EncryptionConfiguration": {
          "ReplicaKmsKeyID": "AWS KMS key IDs(in ARN format) to use for
encrypting object replicas"
        }
      },
      "SourceSelectionCriteria": {
        "SseKmsEncryptedObjects": {
          "Status": "Enabled"
        }
      }
    }
  ]
}
```

- b. Edit the JSON to provide values for the *destination* bucket, *KMS ID ARN* and *IAM-role-ARN*. Save the changes.
- c. Add the replication configuration to your *source* bucket. Be sure to provide the *source* bucket name.

```
$ aws s3api put-bucket-replication \
--replication-configuration file://replication.json \
--bucket source \
--profile acctA
```

6. Test the setup to verify that encrypted objects are replicated. In the Amazon S3 console:
 - a. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

- b. In the **source** bucket, create a folder named **Tax**.
- c. Add sample objects to the folder. Be sure to choose the encryption option and specify your KMS key to encrypt the objects.
- d. Verify that the **destination** bucket contains the object replicas and that they are encrypted using the KMS encryption key that you specified in the configuration.

Replicate Encrypted Objects (AWS SDK)

For a code example to add replication configuration, see [Configure Replication When Buckets Are Owned by the Same Account \(AWS SDK\)](#) (p. 579). You need to modify the replication configuration appropriately.

For conceptual information, see [Additional Replication Configuration: Replicating Objects Created with Server-Side Encryption \(SSE\) Using Encryption Keys stored in AWS KMS](#) (p. 570).

Replication Status Information

To get the replication status of the objects in a bucket, use the Amazon S3 inventory tool. Amazon S3 sends a CSV file to the destination bucket that you specify in the inventory configuration. You can also use Amazon Athena to query the replication status in the inventory report. For more information about Amazon S3 inventory, see [Amazon S3 Inventory](#) (p. 422).

In replication, you have a source bucket on which you configure replication and a destination bucket where Amazon S3 replicates objects. When you request an object (using **GET** object) or object metadata (using **HEAD** object) from these buckets, Amazon S3 returns the **x-amz-replication-status** header in the response:

- When you request an object from the source bucket, Amazon S3 returns the **x-amz-replication-status** header if the object in your request is eligible for replication.

For example, suppose that you specify the object prefix **TaxDocs** in your replication configuration to tell Amazon S3 to replicate only objects with the key name prefix **TaxDocs**. Any objects that you upload that have this key name prefix—for example, **TaxDocs/document1.pdf**—will be replicated. For object requests with this key name prefix, Amazon S3 returns the **x-amz-replication-status** header with one of the following values for the object's replication status: **PENDING**, **COMPLETED**, or **FAILED**.

Note

If object replication fails after you upload an object, you can't retry replication. You must upload the object again.

- When you request an object from the destination bucket, if the object in your request is a replica that Amazon S3 created, Amazon S3 returns the **x-amz-replication-status** header with the value **REPLICA**.

You can find the object replication status using the console, the AWS Command Line Interface (AWS CLI), or the AWS SDK.

- **Console:** Choose the object, and then choose **Properties** to view object properties, including replication status.
- **AWS CLI:** Use the **head-object** AWS CLI command to retrieve object metadata.

```
aws s3api head-object --bucket source-bucket --key object-key --version-id object-version-id
```

The command returns object metadata, including the `ReplicationStatus` as shown in the following example response.

```
{
  "AcceptRanges": "bytes",
  "ContentType": "image/jpeg",
  "LastModified": "Mon, 23 Mar 2015 21:02:29 GMT",
  "ContentLength": 3191,
  "ReplicationStatus": "COMPLETED",
  "VersionId": "jfnW.HIMOfYiD_9rGbSkmroXsFj3fqZ.",
  "ETag": "\"6805f2cfc46c0f04559748bb039d69ae\"",
  "Metadata": {
  }
}
```

- **AWS SDKs:** The following code fragments get replication status with the AWS SDK for Java and AWS SDK for .NET, respectively.
- AWS SDK for Java

```
GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest(bucketName,
    key);
ObjectMetadata metadata = s3Client.getObjectMetadata(metadataRequest);

System.out.println("Replication Status : " +
    metadata.getRawMetadataValue(Headers.OBJECT_REPLICATION_STATUS));
```

- AWS SDK for .NET

```
GetObjectMetadataRequest getmetadataRequest = new GetObjectMetadataRequest
{
    BucketName = sourceBucket,
    Key         = objectKey
};

GetObjectMetadataResponse getmetadataResponse =
    client.GetObjectMetadata(getmetadataRequest);
Console.WriteLine("Object replication status: {0}",
    getmetadataResponse.ReplicationStatus);
```

Note

Before deleting an object from a source bucket that has replication enabled, check the object's replication status to ensure that the object has been replicated.

If lifecycle configuration is enabled on the source bucket, Amazon S3 puts suspends lifecycle actions until it marks the objects status as either `COMPLETED` or `FAILED`.

Related Topics

[Replication \(p. 551\)](#)

Troubleshooting Replication

If object replicas don't appear in the destination bucket after you configure replication, use these troubleshooting tips to identify and fix issues.

- The time that it takes Amazon S3 to replicate an object depends on the size of the object. For large objects, replication can take up to several hours. If the object that is being replicated is large, check later to see if it appears in the destination bucket. You can also check the source object replication status. If object replication status is pending, then you know that Amazon S3 has not completed the replication. If object replication status is failed, check the replication configuration set on the source bucket.
- In the replication configuration on the source bucket, verify the following:
 - The Amazon Resource Name (ARN) of the destination bucket is correct.
 - The key name prefix is correct. For example, if you set the configuration to replicate objects with the prefix `Tax`, then only objects with key names such as `Tax/document1` or `Tax/document2` are replicated. An object with the key name `document3` is not replicated.
 - The status is `enabled`.
- If the destination bucket is owned by another AWS account, verify that the bucket owner has a bucket policy on the destination bucket that allows the source bucket owner to replicate objects. For an example, see [Example 2: Configuring Replication When the Source and Destination Buckets Are Owned by Different Accounts](#) (p. 584)
- If an object replica doesn't appear in the destination bucket, the following might have prevented replication:
 - Amazon S3 doesn't replicate an object in a source bucket that is a replica created by another replication configuration. For example, if you set replication configuration from bucket A to bucket B to bucket C, Amazon S3 doesn't replicate object replicas in bucket B to bucket C.
 - A source bucket owner can grant other AWS accounts permission to upload objects. By default, the source bucket owner doesn't have permissions for the objects created by other accounts. The replication configuration replicates only the objects for which the source bucket owner has access permissions. The source bucket owner can grant other AWS accounts permissions to create objects conditionally, requiring explicit access permissions on those objects. For an example policy, see [Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control](#) (p. 377).
- Suppose that in the replication configuration, you add a rule to replicate a subset of objects having a specific tag. In this case, you must assign the specific tag key and value at the time of creating the object for Amazon S3 to replicate the object. If you first create an object and then add the tag to the existing object, Amazon S3 does not replicate the object.

Related Topics

[Replication](#) (p. 551)

Replication Additional Considerations

Amazon S3 also supports bucket configurations for the following:

- Versioning — For more information, see [Using Versioning](#) (p. 432).
- Website hosting — For more information, see [Hosting a Static Website on Amazon S3](#) (p. 503).
- Bucket access through a bucket policy or access control list (ACL) — For more information, see [Using Bucket Policies and User Policies](#) (p. 341) and see [Managing Access with ACLs](#) (p. 403).
- Log storage — For more information, [Amazon S3 Server Access Logging](#) (p. 647).
- Lifecycle management for objects in a bucket — For more information, see [Object Lifecycle Management](#) (p. 119).

This topic explains how bucket replication configuration affects the behavior of these bucket configurations.

Topics

- [Lifecycle Configuration and Object Replicas \(p. 597\)](#)
- [Versioning Configuration and Replication Configuration \(p. 597\)](#)
- [Logging Configuration and Replication Configuration \(p. 597\)](#)
- [CRR and the Destination Region \(p. 598\)](#)
- [Pausing Replication \(p. 598\)](#)
- [Related Topics \(p. 598\)](#)

Lifecycle Configuration and Object Replicas

The time it takes for Amazon S3 to replicate an object depends on the size of the object. For large objects, it can take several hours. Although it might take a while before a replica is available in the destination bucket, it takes the same amount of time to create the replica as it took to create the corresponding object in the source bucket. If a lifecycle policy is enabled on the destination bucket, the lifecycle rules honor the original creation time of the object, not when the replica became available in the destination bucket.

If you have an object Expiration lifecycle policy in a non-versioned bucket, and you want to maintain the same permanent delete behavior when you enable versioning, you must add a noncurrent expiration policy to manage the deletions of the noncurrent object versions in the version-enabled bucket.

Replication configuration requires the bucket to be versioning-enabled. When you enable versioning on a bucket, keep the following in mind:

- If you have an object Expiration lifecycle policy, after you enable versioning, add a `NonCurrentVersionExpiration` policy to maintain the same permanent delete behavior as before you enabled versioning.
- If you have a Transition lifecycle policy, after you enable versioning, consider adding a `NonCurrentVersionTransition` policy.

Versioning Configuration and Replication Configuration

Both the source and destination buckets must be versioning-enabled when you configure replication on a bucket. After you enable versioning on both the source and destination buckets and configure replication on the source bucket, you will encounter the following issues:

- If you attempt to disable versioning on the source bucket, Amazon S3 returns an error. You must remove the replication configuration before you can disable versioning on the source bucket.
- If you disable versioning on the destination bucket, replication fails. The source object has the replication status `Failed`.

Logging Configuration and Replication Configuration

If Amazon S3 delivers logs to a bucket that has replication enabled, it replicates the log objects.

If server access logs ([Amazon S3 Server Access Logging \(p. 647\)](#)) or AWS CloudTrail Logs ([Logging Amazon S3 API Calls by Using AWS CloudTrail \(p. 621\)](#)) are enabled on your source or destination

bucket, Amazon S3 includes replication-related requests in the logs. For example, Amazon S3 logs each object that it replicates.

CRR and the Destination Region

In a cross-Region replication (CRR) configuration, the source and destination buckets must be in different AWS Regions. You might choose the Region for your destination bucket based on either your business needs or cost considerations. For example, interregion data transfer charges vary depending on the Regions that you choose. Suppose that you chose US East (N. Virginia) (us-east-1) as the Region for your source bucket. If you choose US West (Oregon) (us-west-2) as the Region for your destination bucket, you pay more than if you choose the US East (Ohio) (us-east-2) Region. For pricing information, see "Data Transfer Pricing" in [Amazon S3 Pricing](#). There are no data transfer charges associated with same-Region Replication (SRR)

Pausing Replication

To temporarily pause replication, disable the relevant rule in the replication configuration.

If replication is enabled and you remove the IAM role that grants Amazon S3 the required permissions, replication fails. Amazon S3 reports the replication status for affected objects as `Failed`.

Related Topics

[Replication \(p. 551\)](#)

Request Routing

Topics

- [Request Redirection and the REST API \(p. 599\)](#)
- [DNS Considerations \(p. 602\)](#)

Programs that make requests against buckets created using the <CreateBucketConfiguration> API must support redirects. Additionally, some clients that do not respect DNS TTLs might encounter issues.

This section describes routing and DNS issues to consider when designing your service or application for use with Amazon S3.

Request Redirection and the REST API

Amazon S3 uses the Domain Name System (DNS) to route requests to facilities that can process them. This system works effectively, but temporary routing errors can occur. If a request arrives at the wrong Amazon S3 location, Amazon S3 responds with a temporary redirect that tells the requester to resend the request to a new endpoint. If a request is incorrectly formed, Amazon S3 uses permanent redirects to provide direction on how to perform the request correctly.

Important

To use this feature, you must have an application that can handle Amazon S3 redirect responses. The only exception is for applications that work exclusively with buckets that were created without <CreateBucketConfiguration>. For more information about location constraints, see [Accessing a Bucket \(p. 55\)](#).

For all Regions that launched after March 20, 2019, if a request arrives at the wrong Amazon S3 location, Amazon S3 returns an HTTP 400 Bad Request error.

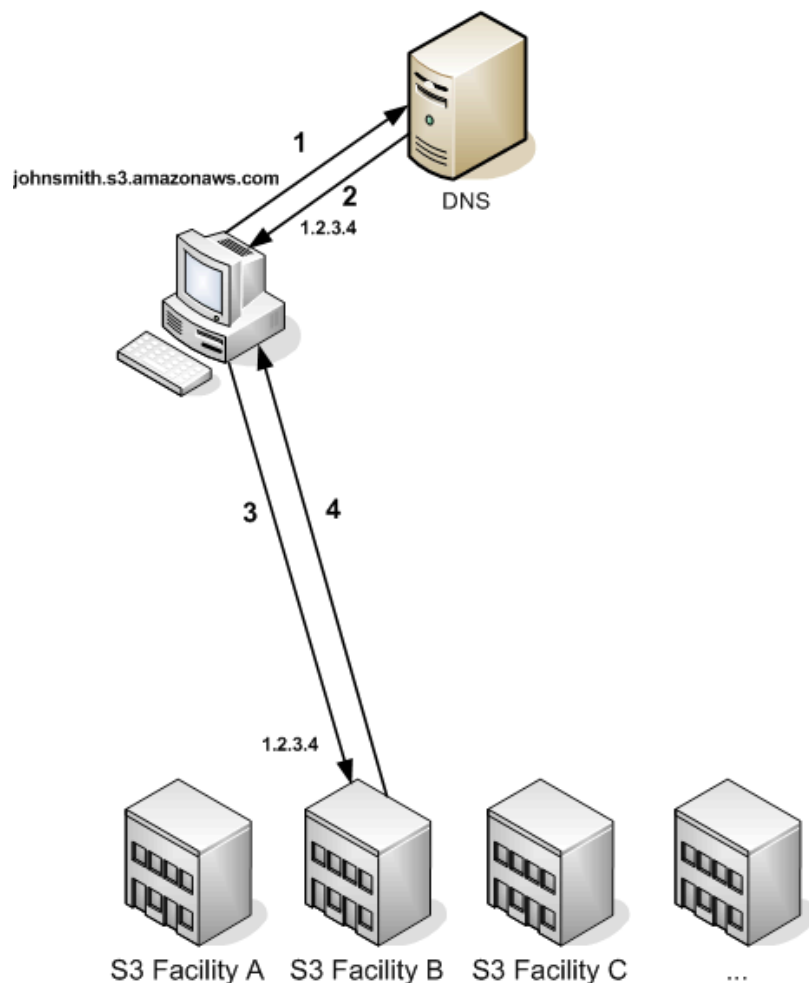
For more information about enabling or disabling an AWS Region, see [AWS Regions and Endpoints](#) in the *AWS General Reference*.

Topics

- [DNS Routing \(p. 599\)](#)
- [Temporary Request Redirection \(p. 600\)](#)
- [Permanent Request Redirection \(p. 602\)](#)
- [Request Redirection Examples \(p. 602\)](#)

DNS Routing

DNS routing routes requests to appropriate Amazon S3 facilities. The following figure and procedure show an example of DNS routing.



DNS routing request steps

1. The client makes a DNS request to get an object stored on Amazon S3.
2. The client receives one or more IP addresses for facilities that can process the request. In this example, the IP address is for Facility B.
3. The client makes a request to Amazon S3 Facility B.
4. Facility B returns a copy of the object to the client.

Temporary Request Redirection

A temporary redirect is a type of error response that signals to the requester that they should resend the request to a different endpoint. Due to the distributed nature of Amazon S3, requests can be temporarily routed to the wrong facility. This is most likely to occur immediately after buckets are created or deleted.

For example, if you create a new bucket and immediately make a request to the bucket, you might receive a temporary redirect, depending on the location constraint of the bucket. If you created the bucket in the US East (N. Virginia) AWS Region, you will not see the redirect because this is also the default Amazon S3 endpoint.

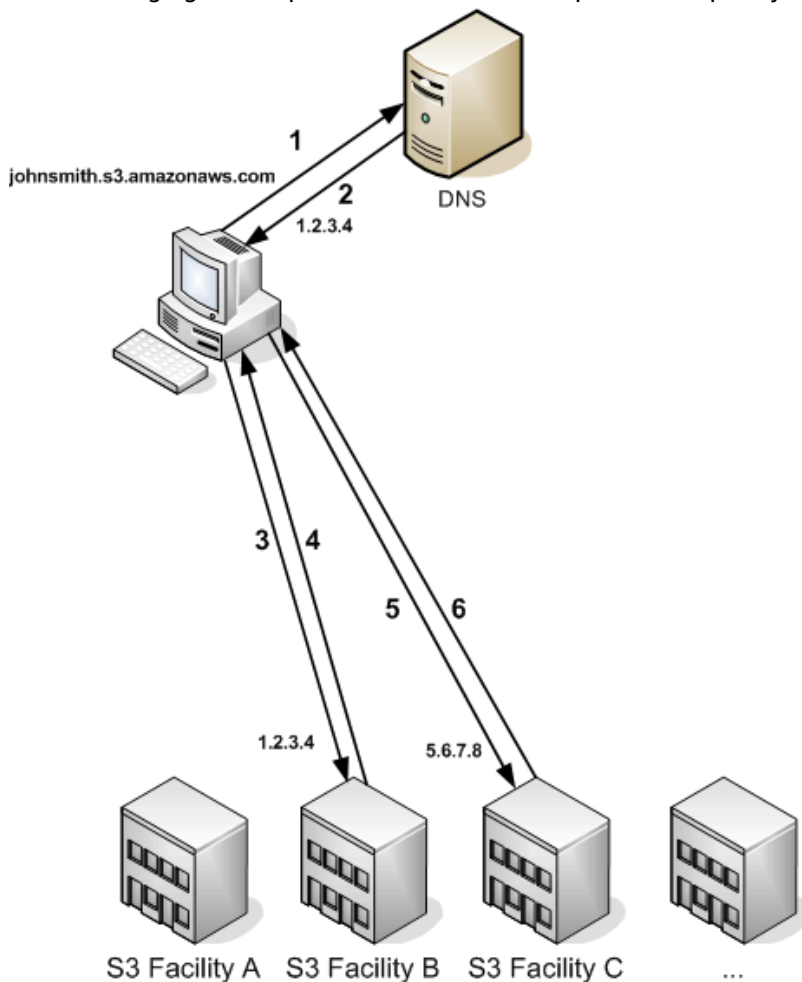
However, if the bucket is created in any other Region, any requests for the bucket go to the default endpoint while the bucket's DNS entry is propagated. The default endpoint redirects the request to the

correct endpoint with an HTTP 302 response. Temporary redirects contain a URI to the correct facility, which you can use to immediately resend the request.

Important

Don't reuse an endpoint provided by a previous redirect response. It might appear to work (even for long periods of time), but it might provide unpredictable results and will eventually fail without notice.

The following figure and procedure shows an example of a temporary redirect.



Temporary request redirection steps

1. The client makes a DNS request to get an object stored on Amazon S3.
2. The client receives one or more IP addresses for facilities that can process the request.
3. The client makes a request to Amazon S3 Facility B.
4. Facility B returns a redirect indicating the object is available from Location C.
5. The client resends the request to Facility C.
6. Facility C returns a copy of the object.

Permanent Request Redirection

A permanent redirect indicates that your request addressed a resource inappropriately. For example, permanent redirects occur if you use a path-style request to access a bucket that was created using `<CreateBucketConfiguration>`. For more information, see [Accessing a Bucket \(p. 55\)](#).

To help you find these errors during development, this type of redirect does not contain a Location HTTP header that allows you to automatically follow the request to the correct location. Consult the resulting XML error document for help using the correct Amazon S3 endpoint.

Request Redirection Examples

The following are examples of temporary request redirection responses.

REST API Temporary Redirect Response

```
HTTP/1.1 307 Temporary Redirect
Location: http://johnsmith.s3-gztb4pa9sq.amazonaws.com/photos/puppy.jpg?rk=e2c69a31
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 12 Oct 2007 01:12:56 GMT
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the specified temporary endpoint.
  Continue to use the original request endpoint for future requests.</Message>
  <Endpoint>johnsmith.s3-gztb4pa9sq.amazonaws.com</Endpoint>
</Error>
```

SOAP API Temporary Redirect Response

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

```
<soapenv:Body>
  <soapenv:Fault>
    <Faultcode>soapenv:Client.TemporaryRedirect</Faultcode>
    <Faultstring>Please re-send this request to the specified temporary endpoint.
    Continue to use the original request endpoint for future requests.</Faultstring>
    <Detail>
      <Bucket>images</Bucket>
      <Endpoint>s3-gztb4pa9sq.amazonaws.com</Endpoint>
    </Detail>
  </soapenv:Fault>
</soapenv:Body>
```

DNS Considerations

One of the design requirements of Amazon S3 is extremely high availability. One of the ways we meet this requirement is by updating the IP addresses associated with the Amazon S3 endpoint in DNS

as needed. These changes are automatically reflected in short-lived clients, but not in some long-lived clients. Long-lived clients will need to take special action to re-resolve the Amazon S3 endpoint periodically to benefit from these changes. For more information about virtual machines (VMs), refer to the following:

- For Java, Sun's JVM caches DNS lookups forever by default; go to the "InetAddress Caching" section of [the InetAddress documentation](#) for information on how to change this behavior.
- For PHP, the persistent PHP VM that runs in the most popular deployment configurations caches DNS lookups until the VM is restarted. Go to [the getHostByName PHP docs](#).

Best Practices Design Patterns: Optimizing Amazon S3 Performance

Your applications can easily achieve thousands of transactions per second in request performance when uploading and retrieving storage from Amazon S3. Amazon S3 automatically scales to high request rates. For example, your application can achieve at least 3,500 PUT/COPY/POST/DELETE and 5,500 GET/HEAD requests per second per prefix in a bucket. There are no limits to the number of prefixes in a bucket. You can increase your read or write performance by parallelizing reads. For example, if you create 10 prefixes in an Amazon S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second.

Some data lake applications on Amazon S3 scan millions or billions of objects for queries that run over petabytes of data. These data lake applications achieve single-instance transfer rates that maximize the network interface use for their [Amazon EC2](#) instance, which can be up to 100 Gb/s on a single instance. These applications then aggregate throughput across multiple instances to get multiple terabits per second.

Other applications are sensitive to latency, such as social media messaging applications. These applications can achieve consistent small object latencies (and first-byte-out latencies for larger objects) of roughly 100–200 milliseconds.

Other AWS services can also help accelerate performance for different application architectures. For example, if you want higher transfer rates over a single HTTP connection or single-digit millisecond latencies, use [Amazon CloudFront](#) or [Amazon ElastiCache](#) for caching with Amazon S3.

Additionally, if you want fast data transport over long distances between a client and an S3 bucket, use [Amazon S3 Transfer Acceleration](#) (p. 73). Transfer Acceleration uses the globally distributed edge locations in CloudFront to accelerate data transport over geographical distances. If your Amazon S3 workload uses server-side encryption with AWS Key Management Service (SSE-KMS), see [AWS KMS Limits](#) in the AWS Key Management Service Developer Guide for information about the request rates supported for your use case.

The following topics describe best practice guidelines and design patterns for optimizing performance for applications that use Amazon S3. This guidance supersedes any previous guidance on optimizing performance for Amazon S3. For example, previously Amazon S3 performance guidelines recommended randomizing prefix naming with hashed characters to optimize performance for frequent data retrievals. You no longer have to randomize prefix naming for performance, and can use sequential date-based naming for your prefixes. Refer to the [Performance Guidelines for Amazon S3](#) (p. 604) and [Performance Design Patterns for Amazon S3](#) (p. 606) for the most current information about performance optimization for Amazon S3.

Topics

- [Performance Guidelines for Amazon S3](#) (p. 604)
- [Performance Design Patterns for Amazon S3](#) (p. 606)

Performance Guidelines for Amazon S3

When building applications that upload and retrieve storage from Amazon S3, follow our best practices guidelines to optimize performance. We also offer more detailed [Performance Design Patterns](#) (p. 606).

To obtain the best performance for your application on Amazon S3, we recommend the following guidelines.

Topics

- [Measure Performance](#) (p. 605)
- [Scale Storage Connections Horizontally](#) (p. 605)
- [Use Byte-Range Fetches](#) (p. 605)
- [Retry Requests for Latency-Sensitive Applications](#) (p. 605)
- [Combine Amazon S3 \(Storage\) and Amazon EC2 \(Compute\) in the Same AWS Region](#) (p. 606)
- [Use Amazon S3 Transfer Acceleration to Minimize Latency Caused by Distance](#) (p. 606)
- [Use the Latest Version of the AWS SDKs](#) (p. 606)

Measure Performance

When optimizing performance, look at network throughput, CPU, and DRAM requirements. Depending on the mix of demands for these different resources, it might be worth evaluating different [Amazon EC2](#) instance types. For more information about instance types, see [Instance Types](#) in the *Amazon EC2 User Guide for Linux Instances*.

It's also helpful to look at DNS lookup time, latency, and data transfer speed using HTTP analysis tools when measuring performance.

Scale Storage Connections Horizontally

Spreading requests across many connections is a common design pattern to horizontally scale performance. When you build high performance applications, think of Amazon S3 as a very large distributed system, not as a single network endpoint like a traditional storage server. You can achieve the best performance by issuing multiple concurrent requests to Amazon S3. Spread these requests over separate connections to maximize the accessible bandwidth from Amazon S3. Amazon S3 doesn't have any limits for the number of connections made to your bucket.

Use Byte-Range Fetches

Using the Range HTTP header in a [GET Object](#) request, you can fetch a byte-range from an object, transferring only the specified portion. You can use concurrent connections to Amazon S3 to fetch different byte ranges from within the same object. This helps you achieve higher aggregate throughput versus a single whole-object request. Fetching smaller ranges of a large object also allows your application to improve retry times when requests are interrupted. For more information, see [Getting Objects](#) (p. 161).

Typical sizes for byte-range requests are 8 MB or 16 MB. If objects are PUT using a multipart upload, it's a good practice to GET them in the same part sizes (or at least aligned to part boundaries) for best performance. GET requests can directly address individual parts; for example, `GET ?partNumber=N`.

Retry Requests for Latency-Sensitive Applications

Aggressive timeouts and retries help drive consistent latency. Given the large scale of Amazon S3, if the first request is slow, a retried request is likely to take a different path and quickly succeed. The AWS SDKs have configurable timeout and retry values that you can tune to the tolerances of your specific application.

Combine Amazon S3 (Storage) and Amazon EC2 (Compute) in the Same AWS Region

Although S3 bucket names are [globally unique](#), each bucket is stored in a Region that you select when you create the bucket. To optimize performance, we recommend that you access the bucket from Amazon EC2 instances in the same AWS Region when possible. This helps reduce network latency and data transfer costs.

For more information about data transfer costs, see [Amazon S3 Pricing](#).

Use Amazon S3 Transfer Acceleration to Minimize Latency Caused by Distance

[Amazon S3 Transfer Acceleration \(p. 73\)](#) manages fast, easy, and secure transfers of files over long geographic distances between the client and an S3 bucket. Transfer Acceleration takes advantage of the globally distributed edge locations in [Amazon CloudFront](#). As the data arrives at an edge location, it is routed to Amazon S3 over an optimized network path. Transfer Acceleration is ideal for transferring gigabytes to terabytes of data regularly across continents. It's also useful for clients that upload to a centralized bucket from all over the world.

You can use the [Amazon S3 Transfer Acceleration Speed Comparison tool](#) to compare accelerated and non-accelerated upload speeds across Amazon S3 Regions. The Speed Comparison tool uses multipart uploads to transfer a file from your browser to various Amazon S3 Regions with and without using Amazon S3 Transfer Acceleration.

Use the Latest Version of the AWS SDKs

The AWS SDKs provide built-in support for many of the recommended guidelines for optimizing Amazon S3 performance. The SDKs provide a simpler API for taking advantage of Amazon S3 from within an application and are regularly updated to follow the latest best practices. For example, the SDKs include logic to automatically retry requests on HTTP 503 errors and are investing in code to respond and adapt to slow connections.

The SDKs also provide the [Transfer Manager](#), which automates horizontally scaling connections to achieve thousands of requests per second, using byte-range requests where appropriate. It's important to use the latest version of the AWS SDKs to obtain the latest performance optimization features.

You can also optimize performance when you are using HTTP REST API requests. When using the REST API, you should follow the same best practices that are part of the SDKs. Allow for timeouts and retries on slow requests, and multiple connections to allow fetching of object data in parallel. For information about using the REST API, see the [Amazon Simple Storage Service API Reference](#).

Performance Design Patterns for Amazon S3

When designing applications to upload and retrieve storage from Amazon S3, use our best practices design patterns for achieving the best performance for your application. We also offer [Performance Guidelines \(p. 604\)](#) for you to consider when planning your application architecture.

To optimize performance, you can use the following design patterns.

Topics

- [Using Caching for Frequently Accessed Content \(p. 607\)](#)

- [Timeouts and Retries for Latency-Sensitive Applications](#) (p. 607)
- [Horizontal Scaling and Request Parallelization for High Throughput](#) (p. 608)
- [Using Amazon S3 Transfer Acceleration to Accelerate Geographically Disparate Data Transfers](#) (p. 609)

Using Caching for Frequently Accessed Content

Many applications that store data in Amazon S3 serve a “working set” of data that is repeatedly requested by users. If a workload is sending repeated GET requests for a common set of objects, you can use a cache such as [Amazon CloudFront](#), [Amazon ElastiCache](#), or [AWS Elemental MediaStore](#) to optimize performance. Successful cache adoption can result in low latency and high data transfer rates. Applications that use caching also send fewer direct requests to Amazon S3, which can help reduce request costs.

Amazon CloudFront is a fast content delivery network (CDN) that transparently caches data from Amazon S3 in a large set of geographically distributed points of presence (PoPs). When objects might be accessed from multiple Regions, or over the internet, CloudFront allows data to be cached close to the users that are accessing the objects. This can result in high performance delivery of popular Amazon S3 content. For information about CloudFront, see the [Amazon CloudFront Developer Guide](#).

Amazon ElastiCache is a managed, in-memory cache. With ElastiCache, you can provision Amazon EC2 instances that cache objects in memory. This caching results in orders of magnitude reduction in GET latency and substantial increases in download throughput. To use ElastiCache, you modify application logic to both populate the cache with hot objects and check the cache for hot objects before requesting them from Amazon S3. For examples of using ElastiCache to improve Amazon S3 GET performance, see the blog post [Turbocharge Amazon S3 with Amazon ElastiCache for Redis](#).

AWS Elemental MediaStore is a caching and content distribution system specifically built for video workflows and media delivery from Amazon S3. MediaStore provides end-to-end storage APIs specifically for video, and is recommended for performance-sensitive video workloads. For information about MediaStore, see the [AWS Elemental MediaStore User Guide](#).

Timeouts and Retries for Latency-Sensitive Applications

There are certain situations where an application receives a response from Amazon S3 indicating that a retry is necessary. Amazon S3 maps bucket and object names to the object data associated with them. If an application generates high request rates (typically sustained rates of over 5,000 requests per second to a small number of objects), it might receive HTTP 503 *slowdown* responses. If these errors occur, each AWS SDK implements automatic retry logic using exponential backoff. If you are not using an AWS SDK, you should implement retry logic when receiving the HTTP 503 error. For information about back-off techniques, see [Error Retries and Exponential Backoff in AWS](#) in the *Amazon Web Services General Reference*.

Amazon S3 automatically scales in response to sustained new request rates, dynamically optimizing performance. While Amazon S3 is internally optimizing for a new request rate, you will receive HTTP 503 request responses temporarily until the optimization completes. After Amazon S3 internally optimizes performance for the new request rate, all requests are generally served without retries.

For latency-sensitive applications, Amazon S3 advises tracking and aggressively retrying slower operations. When you retry a request, we recommend using a new connection to Amazon S3 and performing a fresh DNS lookup.

When you make large variably sized requests (for example, more than 128 MB), we advise tracking the throughput being achieved and retrying the slowest 5 percent of the requests. When you make smaller

requests (for example, less than 512 KB), where median latencies are often in the tens of milliseconds range, a good guideline is to retry a GET or PUT operation after 2 seconds. If additional retries are needed, the best practice is to back off. For example, we recommend issuing one retry after 2 seconds and a second retry after an additional 4 seconds.

If your application makes fixed-size requests to Amazon S3, you should expect more consistent response times for each of these requests. In this case, a simple strategy is to identify the slowest 1 percent of requests and to retry them. Even a single retry is frequently effective at reducing latency.

If you are using AWS Key Management Service (AWS KMS) for server-side encryption, see [Limits](#) in the *AWS Key Management Service Developer Guide* for information about the request rates that are supported for your use case.

Horizontal Scaling and Request Parallelization for High Throughput

Amazon S3 is a very large distributed system. To help you take advantage of its scale, we encourage you to horizontally scale parallel requests to the Amazon S3 service endpoints. In addition to distributing the requests within Amazon S3, this type of scaling approach helps distribute the load over multiple paths through the network.

For high-throughput transfers, Amazon S3 advises using applications that use multiple connections to GET or PUT data in parallel. For example, this is supported by [Amazon S3 Transfer Manager](#) in the AWS Java SDK, and most of the other AWS SDKs provide similar constructs. For some applications, you can achieve parallel connections by launching multiple requests concurrently in different application threads, or in different application instances. The best approach to take depends on your application and the structure of the objects that you are accessing.

You can use the AWS SDKs to issue GET and PUT requests directly rather than employing the management of transfers in the AWS SDK. This approach lets you tune your workload more directly, while still benefiting from the SDK's support for retries and its handling of any HTTP 503 responses that might occur. As a general rule, when you download large objects within a Region from Amazon S3 to [Amazon EC2](#), we suggest making concurrent requests for byte ranges of an object at the granularity of 8–16 MB. Make one concurrent request for each 85–90 MB/s of desired network throughput. To saturate a 10 Gb/s network interface card (NIC), you might use about 15 concurrent requests over separate connections. You can scale up the concurrent requests over more connections to saturate faster NICs, such as 25 Gb/s or 100 Gb/s NICs.

Measuring performance is important when you tune the number of requests to issue concurrently. We recommend starting with a single request at a time. Measure the network bandwidth being achieved and the use of other resources that your application uses in processing the data. You can then identify the bottleneck resource (that is, the resource with the highest usage), and hence the number of requests that are likely to be useful. For example, if processing one request at a time leads to a CPU usage of 25 percent, it suggests that up to four concurrent requests can be accommodated. Measurement is essential, and it is worth confirming resource use as the request rate is increased.

If your application issues requests directly to Amazon S3 using the REST API, we recommend using a pool of HTTP connections and re-using each connection for a series of requests. Avoiding per-request connection setup removes the need to perform TCP slow-start and Secure Sockets Layer (SSL) handshakes on each request. For information about using the REST API, see the [Amazon Simple Storage Service API Reference](#).

Finally, it's worth paying attention to DNS and double-checking that requests are being spread over a wide pool of Amazon S3 IP addresses. DNS queries for Amazon S3 cycle through a large list of IP endpoints. But caching resolvers or application code that reuses a single IP address do not benefit from address diversity and the load balancing that follows from it. Network utility tools such as the `netstat` command line tool can show the IP addresses being used for communication with Amazon S3, and we

provide guidelines for DNS configurations to use. For more information about these guidelines, see [DNS Considerations](#) (p. 602).

Using Amazon S3 Transfer Acceleration to Accelerate Geographically Disparate Data Transfers

[Amazon S3 Transfer Acceleration](#) (p. 73) is effective at minimizing or eliminating the latency caused by geographic distance between globally dispersed clients and a regional application using Amazon S3. Transfer Acceleration uses the globally distributed edge locations in CloudFront for data transport. The AWS edge network has points of presence in more than 50 locations. Today, it is used to distribute content through CloudFront and to provide rapid responses to DNS queries made to [Amazon Route 53](#).

The edge network also helps to accelerate data transfers into and out of Amazon S3. It is ideal for applications that transfer data across or between continents, have a fast internet connection, use large objects, or have a lot of content to upload. As the data arrives at an edge location, data is routed to Amazon S3 over an optimized network path. In general, the farther away you are from an Amazon S3 Region, the higher the speed improvement you can expect from using Transfer Acceleration.

You can set up Transfer Acceleration on new or existing buckets. You can use a separate Amazon S3 Transfer Acceleration endpoint to use the AWS edge locations. The best way to test whether Transfer Acceleration helps client request performance is to use the [Amazon S3 Transfer Acceleration Speed Comparison tool](#). Network configurations and conditions vary from time to time and from location to location. So you are charged only for transfers where Amazon S3 Transfer Acceleration can potentially improve your upload performance. For information about using Transfer Acceleration with different AWS SDKs, see [Amazon S3 Transfer Acceleration Examples](#) (p. 76).

Monitoring Amazon S3

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon S3 and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multipoint failure if one occurs. But before you start monitoring Amazon S3, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

Topics

- [Monitoring Tools \(p. 610\)](#)
- [Monitoring Metrics with Amazon CloudWatch \(p. 611\)](#)
- [Metrics Configurations for Buckets \(p. 617\)](#)
- [Logging with Amazon S3 \(p. 619\)](#)
- [Logging Amazon S3 API Calls by Using AWS CloudTrail \(p. 621\)](#)
- [Using AWS CloudTrail to Identify Amazon S3 Requests \(p. 628\)](#)

Monitoring Tools

AWS provides various tools that you can use to monitor Amazon S3. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated Monitoring Tools

You can use the following automated monitoring tools to watch Amazon S3 and report when something is wrong:

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods. For more information, see [Monitoring Metrics with Amazon CloudWatch \(p. 611\)](#).
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see [Logging Amazon S3 API Calls by Using AWS CloudTrail \(p. 621\)](#).

Manual Monitoring Tools

Another important part of monitoring Amazon S3 involves manually monitoring those items that the CloudWatch alarms don't cover. The Amazon S3, CloudWatch, Trusted Advisor, and other AWS Management Console dashboards provide an at-a-glance view of the state of your AWS environment. You might want to enable server access logging, which tracks requests for access to your bucket. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. For more information, see [Amazon S3 Server Access Logging \(p. 647\)](#) in the *Amazon Simple Storage Service Developer Guide*.

- Amazon S3 dashboard shows:
 - Your buckets and the objects and properties they contain.
- CloudWatch home page shows:
 - Current alarms and status.
 - Graphs of alarms and resources.
 - Service health status.

In addition, you can use CloudWatch to do the following:

- Create [customized dashboards](#) to monitor the services you care about.
- Graph metric data to troubleshoot issues and discover trends.
- Search and browse all your AWS resource metrics.
- Create and edit alarms to be notified of problems.
- AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see [AWS Trusted Advisor](#).

Trusted Advisor has these checks that relate to Amazon S3:

- Checks of the logging configuration of Amazon S3 buckets.
- Security checks for Amazon S3 buckets that have open access permissions.
- Fault tolerance checks for Amazon S3 buckets that do not have versioning enabled, or have versioning suspended.

Monitoring Metrics with Amazon CloudWatch

Amazon CloudWatch metrics for Amazon S3 can help you understand and improve the performance of applications that use Amazon S3. There are two ways that you can use CloudWatch with Amazon S3.

- **Daily Storage Metrics for Buckets** - You can monitor bucket storage using CloudWatch, which collects and processes storage data from Amazon S3 into readable, daily metrics. These storage metrics for Amazon S3 are reported once per day and are provided to all customers at no additional cost.
- **Request metrics** - You can choose to monitor Amazon S3 requests to quickly identify and act on operational issues. The metrics are available at 1-minute intervals after some latency to process. These CloudWatch metrics are billed at the same rate as the Amazon CloudWatch custom metrics. For information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#). To learn how to opt in to getting these metrics, see [Metrics Configurations for Buckets \(p. 617\)](#).

When enabled, request metrics are reported for all object operations. By default, these 1-minute metrics are available at the Amazon S3 bucket level. You can also define a filter for the metrics collected using a shared prefix or object tag. This allows you to align metrics filters to specific business applications, workflows, or internal organizations.

All CloudWatch statistics are retained for a period of 15 months so that you can access historical information and gain a better perspective on how your web application or service is performing. For more information, see [What Is Amazon CloudWatch?](#) in the *Amazon CloudWatch User Guide*.

Metrics and Dimensions

The storage metrics and dimensions that Amazon S3 sends to CloudWatch are listed below.

Amazon S3 CloudWatch Daily Storage Metrics for Buckets

The AWS/S3 namespace includes the following daily storage metrics for buckets.

Metric	Description
BucketSizeBytes	<p>The amount of data in bytes stored in a bucket in the STANDARD storage class, INTELLIGENT_TIERING storage class, Standard - Infrequent Access (STANDARD_IA) storage class, OneZone - Infrequent Access (ONEZONE_IA), Reduced Redundancy Storage (RRS) class, Deep Archive Storage (DEEP_ARCHIVE) class or, Glacier (GLACIER) storage class. This value is calculated by summing the size of all objects in the bucket (both current and noncurrent objects), including the size of all parts for all incomplete multipart uploads to the bucket.</p> <p>Valid storage type filters: StandardStorage, IntelligentTieringStorage, StandardIAStorage, StandardIASizeOverhead, StandardIAObjectOverhead, OneZoneIAStorage, OneZoneIASizeOverhead, ReducedRedundancyStorage, GlacierStorage, GlacierStagingStorage, GlacierObjectOverhead, GlacierS3ObjectOverhead, DeepArchiveStorage, DeepArchiveObjectOverhead, DeepArchiveS3ObjectOverhead and, DeepArchiveStagingStorage (see the StorageType dimension)</p> <p>Units: Bytes</p> <p>Valid statistics: Average</p>
NumberOfObjects	<p>The total number of objects stored in a bucket for all storage classes except for the GLACIER storage class. This value is calculated by counting all objects in the bucket (both current and noncurrent objects) and the total number of parts for all incomplete multipart uploads to the bucket.</p> <p>Valid storage type filters: AllStorageTypes (see the StorageType dimension)</p> <p>Units: Count</p> <p>Valid statistics: Average</p>

Amazon S3 CloudWatch Request Metrics

The AWS/S3 namespace includes the following request metrics.

Metric	Description
AllRequests	<p>The total number of HTTP requests made to an Amazon S3 bucket, regardless of type. If you're using a metrics configuration with a filter, then this metric only returns the HTTP requests made to the objects in the bucket that meet the filter's requirements.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>
GetRequests	<p>The number of HTTP GET requests made for objects in an Amazon S3 bucket. This doesn't include list operations.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p> <p>Note Paginated list-oriented requests, like List Multipart Uploads, List Parts, Get Bucket Object versions, and others, are not included in this metric.</p>
PutRequests	<p>The number of HTTP PUT requests made for objects in an Amazon S3 bucket.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>
DeleteRequests	<p>The number of HTTP DELETE requests made for objects in an Amazon S3 bucket. This also includes Delete Multiple Objects requests. This metric shows the number of requests, not the number of objects deleted.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>
HeadRequests	<p>The number of HTTP HEAD requests made to an Amazon S3 bucket.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>
PostRequests	<p>The number of HTTP POST requests made to an Amazon S3 bucket.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p> <p>Note Delete Multiple Objects and SELECT Object Content requests are not included in this metric.</p>
SelectRequests	<p>The number of Amazon S3 SELECT Object Content requests made for objects in an Amazon S3 bucket.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>

Metric	Description
SelectScannedBytes	<p>The number of bytes of data scanned with Amazon S3 SELECT Object Content requests in an Amazon S3 bucket.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
SelectReturnedBytes	<p>The number of bytes of data returned with Amazon S3 SELECT Object Content requests in an Amazon S3 bucket.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
ListRequests	<p>The number of HTTP requests that list the contents of a bucket.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>
BytesDownloaded	<p>The number of bytes downloaded for requests made to an Amazon S3 bucket, where the response includes a body.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
BytesUploaded	<p>The number of bytes uploaded that contain a request body, made to an Amazon S3 bucket.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
4xxErrors	<p>The number of HTTP 4xx client error status code requests made to an Amazon S3 bucket with a value of either 0 or 1. The average statistic shows the error rate, and the sum statistic shows the count of that type of error, during each period.</p> <p>Units: Count</p> <p>Valid statistics: Average (reports per request), Sum (reports per period), Min, Max, Sample Count</p>
5xxErrors	<p>The number of HTTP 5xx server error status code requests made to an Amazon S3 bucket with a value of either 0 or 1. The average statistic shows the error rate, and the sum statistic shows the count of that type of error, during each period.</p> <p>Units: Counts</p> <p>Valid statistics: Average (reports per request), Sum (reports per period), Min, Max, Sample Count</p>

Metric	Description
FirstByteLatency	<p>The per-request time from the complete request being received by an Amazon S3 bucket to when the response starts to be returned.</p> <p>Units: Milliseconds</p> <p>Valid statistics: Average, Sum, Min, Max(same as p100), Sample Count, any percentile between p0.0 and p100</p>
TotalRequestLatency	<p>The elapsed per-request time from the first byte received to the last byte sent to an Amazon S3 bucket. This includes the time taken to receive the request body and send the response body, which is not included in FirstByteLatency.</p> <p>Units: Milliseconds</p> <p>Valid statistics: Average, Sum, Min, Max(same as p100), Sample Count, any percentile between p0.0 and p100</p>

Amazon S3 CloudWatch Dimensions

The following dimensions are used to filter Amazon S3 metrics.

Dimension	Description
BucketName	This dimension filters the data you request for the identified bucket only.
StorageType	<p>This dimension filters the data that you have stored in a bucket by the following types of storage:</p> <ul style="list-style-type: none">• StandardStorage - The number of bytes used for objects in the STANDARD storage class.• IntelligentTieringFASStorage - The number of bytes used for objects in the Frequent Access tier of INTELLIGENT_TIERING storage class.• IntelligentTieringIASStorage - The number of bytes used for objects in the Infrequent Access tier of INTELLIGENT_TIERING storage class.• StandardIASStorage - The number of bytes used for objects in the Standard - Infrequent Access (STANDARD_IA) storage class.• StandardIASizeOverhead - The number of bytes used for objects smaller than 128 KB in size in the STANDARD_IA storage class.• OneZoneIASStorage - The number of bytes used for objects in the OneZone - Infrequent Access (ONEZONE_IA) storage class.• OneZoneIASizeOverhead - The number of bytes used for objects smaller than 128 KB in size in the ONEZONE_IA storage class.• ReducedRedundancyStorage - The number of bytes used for objects in the Reduced Redundancy Storage (RRS) class.• GlacierStorage - The number of bytes used for objects in the GLACIER storage class.

Dimension	Description
	<ul style="list-style-type: none"> GlacierStagingStorage - The number of bytes used for parts of Multipart objects before the CompleteMultipartUpload request is completed on objects in the GLACIER storage class. GlacierObjectOverhead - For each archived object, GLACIER adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged GLACIER rates for this additional storage. GlacierS3ObjectOverhead - For each object archived to GLACIER, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. You are charged STANDARD rates for this additional storage. DeepArchiveStorage - The number of bytes used for objects in the DEEP_ARCHIVE storage class. DeepArchiveObjectOverhead - For each archived object, DEEP_ARCHIVE adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged DEEP_ARCHIVE rates for this additional storage. DeepArchiveS3ObjectOverhead - For each object archived to DEEP_ARCHIVE, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. You are charged STANDARD rates for this additional storage. DeepArchiveStagingStorage - The number of bytes used for parts of Multipart objects before the CompleteMultipartUpload request is completed on objects in the DEEP_ARCHIVE storage class.
FilterId	This dimension filters metrics configurations that you specify for request metrics on a bucket, for example, a prefix or a tag. You specify a filter id when you create a metrics configuration. For more information, see Metrics Configurations for Buckets .

Accessing CloudWatch Metrics

You can use the following procedures to view the storage metrics for Amazon S3. To get the Amazon S3 metrics involved, you must set a start and end timestamp. For metrics for any given 24-hour period, set the time period to 86400 seconds, the number of seconds in a day. Also, remember to set the `BucketName` and `StorageType` dimensions.

For example, if you use the AWS CLI to get the average of a specific bucket's size, in bytes, you could use the following command.

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --namespace AWS/S3
--start-time 2016-10-19T00:00:00Z --end-time 2016-10-20T00:00:00Z --statistics Average
--unit Bytes --region us-west-2 --dimensions Name=BucketName,Value=ExampleBucket
Name=StorageType,Value=StandardStorage --period 86400 --output json
```

This example produces the following output.

```
{
  "Datapoints": [
    {
      "Timestamp": "2016-10-19T00:00:00Z",
```

```
        "Average": 1025328.0,  
        "Unit": "Bytes"  
    },  
    ],  
    "Label": "BucketSizeBytes"  
}
```

To view metrics using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **S3** namespace.
4. (Optional) To view a metric, enter the metric name in the search field.
5. (Optional) To filter by the **StorageType** dimension, enter the name of the storage class in the search field.

To view a list of valid metrics stored for your AWS account using the AWS CLI

- At a command prompt, use the following command.

```
aws cloudwatch list-metrics --namespace "AWS/S3"
```

Related Resources

- [Amazon CloudWatch Logs API Reference](#)
- [Amazon CloudWatch User Guide](#)
- [list-metrics](#) action in the *AWS CLI Command Reference*.
- [get-metric-statistics](#) action in the *AWS CLI Command Reference*.
- [Metrics Configurations for Buckets](#) (p. 617).

Metrics Configurations for Buckets

With CloudWatch request metrics for Amazon S3, you can receive 1-minute CloudWatch metrics, set CloudWatch alarms, and access CloudWatch dashboards to view near-real-time operations and performance of your Amazon S3 storage. For applications that depend on cloud storage, these metrics let you quickly identify and act on operational issues. When enabled, these 1-minute metrics are available at the Amazon S3 bucket-level, by default.

If you want to get the CloudWatch request metrics for the objects in a bucket, you must create a metrics configuration for the bucket. You can also define a filter for the metrics collected using a shared prefix or object tags. This allows you to align metrics filters to specific business applications, workflows, or internal organizations.

For more information about the CloudWatch metrics that are available and the differences between storage and request metrics, see [Monitoring Metrics with Amazon CloudWatch](#) (p. 611).

Keep the following in mind when using metrics configurations:

- You can have a maximum of 1,000 metrics configurations per bucket.
- You can choose which objects in a bucket to include in metrics configurations by using filters. Filtering on a shared prefix or object tag allows you to align metrics filters to specific business applications,

workflows, or internal organizations. To request metrics for the entire bucket, create a metrics configuration without a filter.

- Metrics configurations are necessary only to enable request metrics. Bucket-level daily storage metrics are always turned on, and are provided at no additional cost. Currently, it's not possible to get daily storage metrics for a filtered subset of objects.
- Each metrics configuration enables the full set of [available request metrics \(p. 612\)](#). Operation-specific metrics (such as `PostRequests`) are reported only if there are requests of that type for your bucket or your filter.
- Request metrics are reported for object-level operations. They are also reported for operations that list bucket contents, like [GET Bucket \(List Objects\)](#), [GET Bucket Object Versions](#), and [List Multipart Uploads](#), but they are not reported for other operations on buckets.
- Request metrics support filtering by prefixes but storage metrics do not.

Best-Effort CloudWatch Metrics Delivery

CloudWatch metrics are delivered on a best-effort basis. Most requests for an Amazon S3 object that have request metrics result in a data point being sent to CloudWatch.

The completeness and timeliness of metrics is not guaranteed. The data point for a particular request might be returned with a timestamp that is later than when the request was actually processed. Or the data point for a minute might be delayed before being available through CloudWatch, or it might not be delivered at all. CloudWatch request metrics give you an idea of the nature of traffic against your bucket in near-real time. It is not meant to be a complete accounting of all requests.

It follows from the best-effort nature of this feature that the reports available at the [Billing & Cost Management Dashboard](#) might include one or more access requests that do not appear in the bucket metrics.

Filtering Metrics Configurations

When working with CloudWatch metric configurations, you have the option of filtering the configuration into groups of related objects within a single bucket. You can filter objects in a bucket for inclusion in a metrics configuration based on one or more of the following elements:

- **Object key name prefix** – Although the Amazon S3 data model is a flat structure, you can infer hierarchy by using a prefix. The Amazon S3 console supports these prefixes with the concept of folders. If you filter by prefix, objects that have the same prefix are included in the metrics configuration.
- **Tag** – You can add tags, which are key-value name pairs, to objects. Tags help you find and organize objects easily. You can also use tags as a filter for metrics configurations.

If you specify a filter, only requests that operate on single objects can match the filter and be included in the reported metrics. Requests like [Delete Multiple Objects](#) and List requests don't return any metrics for configurations with filters.

To request more complex filtering, choose two or more elements. Only objects that have all of those elements are included in the metrics configuration. If you don't set filters, all of the objects in the bucket are included in the metrics configuration.

How to Add Metrics Configurations

You can add metrics configurations to a bucket through the Amazon S3 console, with the AWS CLI, or with the Amazon S3 REST API. For information about how to do this in the AWS Management Console,

see the [How Do I Configure Request Metrics for an S3 Bucket?](#) in the Amazon Simple Storage Service Console User Guide.

To add metrics configurations using the AWS CLI

1. Install and set up the AWS CLI. For instructions, see [Getting Set Up with the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.
2. Open a terminal.
3. Run the following command to add a metrics configuration.

```
aws s3api put-bucket-metrics-configuration --endpoint http://s3-us-west-2.amazonaws.com
--bucket bucket-name --id metrics-config-id --metrics-configuration '{"Id":"metrics-
config-id", "Filter": {"Prefix": "prefix1"}}'
```

4. To verify that the configuration was added, execute the following command.

```
aws s3api get-bucket-metrics-configuration --endpoint http://s3-us-west-2.amazonaws.com
--bucket bucket-name --id metrics-config-id
```

This returns the following response.

```
{
  "MetricsConfiguration": {
    "Filter": {
      "Prefix": "prefix1"
    },
    "Id": "metrics-config-id"
  }
}
```

You can also add metrics configurations programmatically with the Amazon S3 REST API. For more information, see the following topics in the *Amazon Simple Storage Service API Reference*:

- [PUT Bucket Metric Configuration](#)
- [GET Bucket Metric Configuration](#)
- [List Bucket Metric Configuration](#)
- [DELETE Bucket Metric Configuration](#)

Logging with Amazon S3

You can record the actions that are taken by users, roles, or AWS services on Amazon S3 resources and maintain log records for auditing and compliance purposes. To do this, you can use [Server Access Logging](#) (p. 647), [AWS CloudTrail logs](#), or a combination of both. We recommend that you use AWS CloudTrail for logging bucket and object-level actions for your Amazon S3 resources.

The following table lists the key properties of AWS CloudTrail logs and Amazon S3 server access logs.

Log Properties	AWS CloudTrail	Amazon S3 Server Logs
Can be forwarded to other systems (CloudWatch Logs, CloudWatch Events)	Yes	

Log Properties	AWS CloudTrail	Amazon S3 Server Logs
Deliver logs to more than one destination (for example, send the same logs to two different buckets)	Yes	
Turn on logs for a subset of objects (prefix)	Yes	
Cross-account log delivery (target and source bucket owned by different accounts)	Yes	
Integrity validation of log file using digital signature/hashing	Yes	
Default/choice of encryption for log files	Yes	
Object operations (using Amazon S3 APIs)	Yes	Yes
Bucket operations (using Amazon S3 APIs)	Yes	Yes
Searchable UI for logs	Yes	
Fields for object lock parameters, Amazon S3 select properties for log records	Yes	
Fields for Object Size, Total Time, Turn-Around Time, and HTTP Referrer for log records		Yes
Lifecycle transitions, expirations, restores		Yes
Logging of keys in a batch delete operation		Yes
Authentication failures ¹		Yes
Accounts where logs get delivered	Bucket owner ² , and requester	Bucket owner only
Performance and Cost	AWS CloudTrail	Amazon S3 Server Logs
Price	Management events (first delivery) are free; data events incur a fee, in addition to storage of logs	No additional cost in addition to storage of logs
Speed of log delivery	Data events every 5 mins; management events every 15 mins	Within a few hours
Log format	JSON	Log file with space-separated, newline-delimited records

Notes:

1. CloudTrail does not deliver logs for requests that fail authentication (in which the provided credentials are not valid). However, it does include logs for requests in which authorization fails (`AccessDenied`) and requests that are made by anonymous users.
2. The S3 bucket owner receives CloudTrail logs only if the account also owns or has full access to the object in the request. For more information, see [Object-Level Actions in Cross-Account Scenarios](#) (p. 624).

Logging Amazon S3 API Calls by Using AWS CloudTrail

Amazon S3 is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon S3. CloudTrail captures a subset of API calls for Amazon S3 as events, including calls from the Amazon S3 console and from code calls to the Amazon S3 APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon S3. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon S3, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

Amazon S3 Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon S3, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for Amazon S3, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

Amazon S3 Bucket-Level Actions Tracked by CloudTrail Logging

By default, CloudTrail logs bucket-level actions. Amazon S3 records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

The tables in this section list the Amazon S3 bucket-level actions that are supported for logging by CloudTrail.

Amazon S3 Bucket-Level Actions Tracked by CloudTrail Logging

REST API Name	API Event Name Used in CloudTrail Log
DELETE Bucket	DeleteBucket
DELETE Bucket cors	DeleteBucketCors
DELETE Bucket encryption	DeleteBucketEncryption
DELETE Bucket lifecycle	DeleteBucketLifecycle
DELETE Bucket policy	DeleteBucketPolicy
DELETE Bucket replication	DeleteBucketReplication
DELETE Bucket tagging	DeleteBucketTagging
DELETE Bucket website	DeleteBucketWebsite
GET Bucket acl	GetBucketAcl
GET Bucket cors	GetBucketCors
GET Bucket encryption	GetBucketEncryption
GET Bucket lifecycle	GetBucketLifecycle
GET Bucket location	GetBucketLocation
GET Bucket logging	GetBucketLogging
GET Bucket notification	GetBucketNotification
GET Bucket policy	GetBucketPolicy
GET Bucket replication	GetBucketReplication
GET Bucket requestPayment	GetBucketRequestPay
GET Bucket tagging	GetBucketTagging
GET Bucket versioning	GetBucketVersioning
GET Bucket website	GetBucketWebsite
GET Service (List all buckets)	ListBuckets

REST API Name	API Event Name Used in CloudTrail Log
PUT Bucket	CreateBucket
PUT Bucket acl	PutBucketAcl
PUT Bucket cors	PutBucketCors
PUT Bucket encryption	PutBucketEncryption
PUT Bucket lifecycle	PutBucketLifecycle
PUT Bucket logging	PutBucketLogging
PUT Bucket notification	PutBucketNotification
PUT Bucket policy	PutBucketPolicy
PUT Bucket replication	PutBucketReplication
PUT Bucket requestPayment	PutBucketRequestPay
PUT Bucket tagging	PutBucketTagging
PUT Bucket versioning	PutBucketVersioning
PUT Bucket website	PutBucketWebsite

In addition to these API operations, you can also use the [OPTIONS object](#) object-level action. This action is treated like a bucket-level action in CloudTrail logging because the action checks the cors configuration of a bucket.

Amazon S3 Object-Level Actions Tracked by CloudTrail Logging

You can also get CloudTrail logs for object-level Amazon S3 actions. To do this, specify the Amazon S3 object for your trail. When an object-level action occurs in your account, CloudTrail evaluates your trail settings. If the event matches the object that you specified in a trail, the event is logged. For more information, see [How Do I Enable Object-Level Logging for an S3 Bucket with AWS CloudTrail Data Events?](#) in the *Amazon Simple Storage Service Console User Guide* and [Data Events](#) in the *AWS CloudTrail User Guide*. The following table lists the object-level actions that CloudTrail can log:

REST API Name	API Event Name Used in CloudTrail Log
Abort Multipart Upload	AbortMultipartUpload
Complete Multipart Upload	CompleteMultipartUpload
Delete Multiple Objects	DeleteObjects
DELETE Object	DeleteObject
GET Object	GetObject
GET Object ACL	GetObjectAcl
GET Object tagging	GetObjectTagging
GET Object torrent	GetObjectTorrent
HEAD Object	HeadObject

REST API Name	API Event Name Used in CloudTrail Log
Initiate Multipart Upload	CreateMultipartUpload
List Parts	ListParts
POST Object	PostObject
POST Object restore	RestoreObject
PUT Object	PutObject
PUT Object acl	PutObjectAcl
PUT Object tagging	PutObjectTagging
PUT Object - Copy	CopyObject
SELECT Object Content	SelectObjectContent
Upload Part	UploadPart
Upload Part - Copy	UploadPartCopy

In addition to these operations, you can use the following bucket-level operations to get CloudTrail logs as object-level Amazon S3 actions under certain conditions:

- [GET Bucket \(List Objects\) Version 2](#) – Select a prefix specified in the trail.
- [GET Bucket Object versions](#) – Select a prefix specified in the trail.
- [HEAD Bucket](#) – Specify a bucket and an empty prefix.
- [Delete Multiple Objects](#) – Specify a bucket and an empty prefix.

Note

CloudTrail does not log key names for the keys that are deleted using the Delete Multiple Objects operation.

Object-Level Actions in Cross-Account Scenarios

The following are special use cases involving the object-level API calls in cross-account scenarios and how CloudTrail logs are reported. CloudTrail always delivers logs to the requester (who made the API call). When setting up cross-account access, consider the examples in this section.

Note

The examples assume that CloudTrail logs are appropriately configured.

Example 1: CloudTrail Delivers Access Logs to the Bucket Owner

CloudTrail delivers access logs to the bucket owner only if the bucket owner has permissions for the same object API. Consider the following cross-account scenario:

- Account-A owns the bucket.
- Account-B (the requester) tries to access an object in that bucket.

CloudTrail always delivers object-level API access logs to the requester. In addition, CloudTrail also delivers the same logs to the bucket owner only if the bucket owner has permissions for the same API actions on that object.

Note

If the bucket owner is also the object owner, the bucket owner gets the object access logs. Otherwise, the bucket owner must get permissions, through the object ACL, for the same object API to get the same object-access API logs.

Example 2: CloudTrail Does Not Proliferate Email Addresses Used in Setting Object ACLs

Consider the following cross-account scenario:

- Account-A owns the bucket.
- Account-B (the requester) sends a request to set an object ACL grant using an email address. For information about ACLs, see [Access Control List \(ACL\) Overview \(p. 403\)](#).

The request gets the logs along with the email information. However, the bucket owner—if they are eligible to receive logs, as in example 1—gets the CloudTrail log reporting the event. However, the bucket owner doesn't get the ACL configuration information, specifically the grantee email and the grant. The only information that the log tells the bucket owner is that an ACL API call was made by Account-B.

CloudTrail Tracking with Amazon S3 SOAP API Calls

CloudTrail tracks Amazon S3 SOAP API calls. Amazon S3 SOAP support over HTTP is deprecated, but it is still available over HTTPS. For more information about Amazon S3 SOAP support, see [Appendix A: Using the SOAP API \(p. 682\)](#).

Important

Newer Amazon S3 features are not supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 SOAP Actions Tracked by CloudTrail Logging

SOAP API Name	API Event Name Used in CloudTrail Log
ListAllMyBuckets	ListBuckets
CreateBucket	CreateBucket
DeleteBucket	DeleteBucket
GetBucketAccessControlPolicy	GetBucketAcl
SetBucketAccessControlPolicy	PutBucketAcl
GetBucketLoggingStatus	GetBucketLogging
SetBucketLoggingStatus	PutBucketLogging

Using CloudTrail Logs with Amazon S3 Server Access Logs and CloudWatch Logs

AWS CloudTrail logs provide a record of actions taken by a user, role, or an AWS service in Amazon S3, while Amazon S3 server access logs provides detailed records for the requests that are made to an S3 bucket. For more information on how the different logs work and their properties, performance and costs, see [the section called “Logging with Amazon S3” \(p. 619\)](#). You can use AWS CloudTrail logs together with server access logs for Amazon S3. CloudTrail logs provide you with detailed API tracking

for Amazon S3 bucket-level and object-level operations. Server access logs for Amazon S3 provide you visibility into object-level operations on your data in Amazon S3. For more information about server access logs, see [Amazon S3 Server Access Logging](#) (p. 647).

You can also use CloudTrail logs together with CloudWatch for Amazon S3. CloudTrail integration with CloudWatch Logs delivers S3 bucket-level API activity captured by CloudTrail to a CloudWatch log stream in the CloudWatch log group that you specify. You can create CloudWatch alarms for monitoring specific API activity and receive email notifications when the specific API activity occurs. For more information about CloudWatch alarms for monitoring specific API activity, see the [AWS CloudTrail User Guide](#). For more information about using CloudWatch with Amazon S3, see [Monitoring Metrics with Amazon CloudWatch](#) (p. 611).

Example: Amazon S3 Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source. It includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the [GET Service](#), [PUT Bucket acl](#), and [GET Bucket versioning](#) actions.

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "111122223333",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2019-02-01T03:18:19Z",
      "eventSource": "s3.amazonaws.com",
      "eventName": "ListBuckets",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "[]",
      "requestParameters": {
        "host": [
          "s3.us-west-2.amazonaws.com"
        ]
      },
      "responseElements": null,
      "additionalEventData": {
        "SignatureVersion": "SigV2",
        "AuthenticationMethod": "QueryString"
      },
      "requestID": "47B8E8D397DCE7A6",
      "eventID": "cdc4b7ed-e171-4cef-975a-ad829d4123e8",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    },
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "111122223333",
```

```
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
    },
    "eventTime": "2019-02-01T03:22:33Z",
    "eventSource": "s3.amazonaws.com",
    "eventName": "PutBucketAcl",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "",
    "userAgent": "[]",
    "requestParameters": {
        "bucketName": "",
        "AccessControlPolicy": {
            "AccessControlList": {
                "Grant": {
                    "Grantee": {
                        "xsi:type": "CanonicalUser",
                        "xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
                        "ID":
"d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"
                    },
                    "Permission": "FULL_CONTROL"
                }
            }
        },
        "xmlns": "http://s3.amazonaws.com/doc/2006-03-01/",
        "Owner": {
            "ID": "d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"
        }
    }
    "host": [
        "s3-us-west-2.amazonaws.com"
    ],
    "acl": [
        ""
    ]
},
"responseElements": null,
"additionalEventData": {
    "SignatureVersion": "SigV4",
    "CipherSuite": "ECDHE-RSA-AES128-SHA",
    "AuthenticationMethod": "AuthHeader"
},
"requestID": "BD8798EACDD16751",
"eventID": "607b9532-1423-41c7-b048-ec2641693c47",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
},
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "111122223333",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
    },
    "eventTime": "2019-02-01T03:26:37Z",
    "eventSource": "s3.amazonaws.com",
    "eventName": "GetBucketVersioning",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "",
    "userAgent": "[]",
    "requestParameters": {
        "host": [
```

```
        "s3.us-west-2.amazonaws.com"
      ],
      "bucketName": "myawsbucket",
      "versioning": [
        ""
      ]
    },
    "responseElements": null,
    "additionalEventData": {
      "SignatureVersion": "SigV4",
      "CipherSuite": "ECDHE-RSA-AES128-SHA",
      "AuthenticationMethod": "AuthHeader",
    },
    "requestID": "07D681279BD94AED",
    "eventID": "f2b287f3-0df1-4961-a2f4-c4bdfed47657",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
  }
]
```

Related Resources

- [AWS CloudTrail User Guide](#)
- [CloudTrail Event Reference](#)
- [Using AWS CloudTrail to Identify Amazon S3 Requests \(p. 628\)](#)

Using AWS CloudTrail to Identify Amazon S3 Requests

Amazon S3 lets you identify requests using an AWS CloudTrail event log. AWS CloudTrail is the preferred way of identifying Amazon S3 requests, but if you are using Amazon S3 server access logs, see [the section called “Using Amazon S3 access logs to identify Amazon S3 requests” \(p. 662\)](#).

Topics

- [How CloudTrail Captures Requests Made to Amazon S3 \(p. 628\)](#)
- [Enabling CloudTrail Event Logging for S3 Buckets and Objects \(p. 629\)](#)
- [Identifying Requests Made to Amazon S3 in a CloudTrail Log \(p. 629\)](#)
- [Using AWS CloudTrail to Identify Amazon S3 Signature Version 2 Requests \(p. 631\)](#)
- [Using AWS CloudTrail to Identify Access to Amazon S3 Objects \(p. 633\)](#)
- [Related Resources \(p. 628\)](#)

How CloudTrail Captures Requests Made to Amazon S3

By default, CloudTrail logs S3 bucket-level API calls that were made in the last 90 days, but not log requests made to objects. Bucket-level calls include events like `CreateBucket`, `DeleteBucket`, `PutBucketLifecycle`, `PutBucketPolicy`, etc. You can see bucket-level events on the CloudTrail console. However, you can't view data events (Amazon S3 object-level calls) there—you must parse or query CloudTrail logs for them.

For information about what Amazon S3 API calls are captured by CloudTrail, see [Amazon S3 Information in CloudTrail](#) (p. 621).

Enabling CloudTrail Event Logging for S3 Buckets and Objects

CloudTrail data events allow you to get information about bucket and object-level requests. To enable CloudTrail data events for a specific bucket, see [How Do I Enable Object-Level Logging for an S3 Bucket with AWS CloudTrail Data Events?](#) in the *Amazon Simple Storage Service Console User Guide*.

To enable CloudTrail data events for all your buckets or for a list of specific buckets, you must [create a trail manually in CloudTrail](#).

Note

- The default setting for CloudTrail is to find only management events. Check to ensure that you have the data events enabled for your account.
- With an S3 bucket that is generating a high workload, you could quickly generate thousands of logs in a short amount of time. Be mindful of how long you choose to enable CloudTrail data events for a busy bucket.

CloudTrail stores Amazon S3 data event logs in an S3 bucket of your choosing. You should consider using a bucket in a separate AWS account to better organize events from multiple buckets you might own into a central place for easier querying and analysis. AWS Organizations makes it easy to create an AWS account that is linked the account owning the bucket you are monitoring. For more information, see [What Is AWS Organizations?](#) in the *AWS Organizations User Guide*.

When you create a trail in CloudTrail, in the data events section, you can select the **Select all S3 buckets in your account** check box to log all object level events.

Note

- It's a best practice to [create an Amazon S3 lifecycle policy](#) for your AWS CloudTrail data event bucket. Configure the lifecycle policy to periodically remove log files after the period of time you believe you need to audit them. Doing so reduces the amount of data that Athena analyzes for each query.
- For information about logging format, see [Logging Amazon S3 API Calls by Using AWS CloudTrail](#).
- For examples of how to query CloudTrail logs, see [Analyze Security, Compliance, and Operational Activity Using AWS CloudTrail and Amazon Athena](#).

Identifying Requests Made to Amazon S3 in a CloudTrail Log

Events logged by CloudTrail are stored as compressed, GZipped JSON objects in your S3 bucket. To efficiently find requests, you should use a service like Amazon Athena to index and query the CloudTrail logs. For more information about CloudTrail and Athena, see [Querying AWS CloudTrail Logs](#).

Using Athena with CloudTrail Logs

After you set up CloudTrail to deliver events to a bucket, you should start to see objects go to your destination bucket on the Amazon S3 console. These are formatted as follows:

```
s3://<myawsexamplebucket>/AWSLogs/<111122223333>/CloudTrail/<Region>/<yyyy>/<mm>/<dd>
```

Example — Use Athena to query CloudTrail event logs for specific requests

Locate your CloudTrail event logs:

s3://myawsexamplebucket/AWSLogs/111122223333/CloudTrail/us-east-2/2019/04/14

With CloudTrail event logs, you can now create an Athena database and table to query them as follows:

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Change the AWS Region to be the same as your CloudTrail destination S3 bucket.
3. In the query window, create an Athena database for your CloudTrail events:

```
CREATE DATABASE s3_cloudtrail_events_db
```

4. Use the following query to create a table for all of your CloudTrail events in the bucket. Be sure to change the bucket name from `<CloudTrail_myawsexamplebucket>` to your bucket's name. Also provide the `AWS_account_ID` CloudTrail that is used in your bucket.

```
CREATE EXTERNAL TABLE s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table(  
    eventversion STRING,  
    useridentity STRUCT<  
        type:STRING,  
        principalid:STRING,  
        arn:STRING,  
        accountid:STRING,  
        invokedby:STRING,  
        accesskeyid:STRING,  
        userName:STRING,  
        sessioncontext:STRUCT<  
            attributes:STRUCT<  
                mfaauthenticated:STRING,  
                creationdate:STRING>,  
                sessionissuer:STRUCT<  
                    type:STRING,  
                    principalid:STRING,  
                    arn:STRING,  
                    accountId:STRING,  
                    userName:STRING>  
            >  
        >,  
    eventtime STRING,  
    eventsource STRING,  
    eventname STRING,  
    awsregion STRING,  
    sourceipaddress STRING,  
    useragent STRING,  
    errorcode STRING,  
    errormessage STRING,  
    requestparameters STRING,  
    responseelements STRING,  
    additionaleventdata STRING,  
    requestid STRING,  
    eventid STRING,  
    resources ARRAY<STRUCT<  
        ARN:STRING,  
        accountId:STRING,  
        type:STRING>>,  
    eventtype STRING,  
    apiversion STRING,  
    readonly STRING,  
    recipientaccountid STRING,  
    serviceeventdetails STRING,
```



```
sharedeventid STRING,  
vpcepointid STRING  
)  
ROW FORMAT SERDE 'com.amazon.emr.hive.serde.CloudTrailSerde'  
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'  
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'  
LOCATION 's3://<myawsexamplebucket>/AWSLogs/<111122223333>/';
```

5. Test Athena to ensure that the query works.

```
SELECT * FROM s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table  
WHERE eventsource='s3.amazonaws.com'  
LIMIT 2;
```

Using AWS CloudTrail to Identify Amazon S3 Signature Version 2 Requests

Amazon S3 lets you identify what API signature version was used to sign a request using an AWS CloudTrail event log. This capability is important because support for Signature Version 2 will be turned off (deprecated). After that, Amazon S3 will no longer accept requests that use Signature Version 2, and all requests must use *Signature Version 4* signing.

We strongly recommend that you use CloudTrail to help determine whether any of your workflows are using Signature Version 2 signing. Remediate them by upgrading your libraries and code to use Signature Version 4 instead to prevent any impact to your business.

For more information, see [Announcement: AWS CloudTrail for Amazon S3 adds new fields for enhanced security auditing](#) in the AWS Discussion Forums.

Note

CloudTrail events for Amazon S3 include the signature version in the request details under the key name of 'additionalEventData'. To find the signature version on requests made for objects in Amazon S3 like GETs, PUTs, and DELETEs, you must enable CloudTrail data events because it is turned off by default.

AWS CloudTrail is the preferred method for identifying Signature Version 2 requests, if you are using Amazon S3 server access logs, see [Using Amazon S3 Access Logs to Identify Signature Version 2 Requests](#) (p. 666)

Athena Query Examples for Identifying Amazon S3 Signature Version 2 Requests

Example — Select all events that are Signature Version 2, and print only EventTime, S3 Action, Request_Parameters, Region, SourceIP, and UserAgent

In the following Athena query, replace

`<s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table>` with your Athena details and increase or remove the limit as needed.

```
SELECT EventTime, EventName as S3_Action, requestParameters as Request_Parameters,  
awsregion as AWS_Region, sourceipaddress as Source_IP, userAgent as User_Agent
```

```
FROM s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table
WHERE eventsource='s3.amazonaws.com'
AND json_extract_scalar(additionalEventData, '$.SignatureVersion')='SigV2'
LIMIT 10;
```

Example — Select all requesters that are sending Signature Version 2 traffic

```
SELECT useridentity.arn, Count(requestid) as RequestCount
FROM s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table
WHERE eventsource='s3.amazonaws.com'
    and json_extract_scalar(additionalEventData, '$.SignatureVersion')='SigV2'
Group by useridentity.arn
```

Partitioning Signature Version 2 Data

If you have a large amount of data that you need to query, you can reduce the costs and runtime of Athena by creating a partitioned table.

To do this, create a new table with partitions as follows.

```
CREATE EXTERNAL TABLE
s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table_partitioned(
    eventversion STRING,
    useridentity STRUCT<
        type:STRING,
        principalid:STRING,
        arn:STRING,
        accountid:STRING,
        invokedby:STRING,
        accesskeyid:STRING,
        userName:STRING,
        sessioncontext:STRUCT<
            attributes:STRUCT<
                mfaauthenticated:STRING,
                creationdate:STRING>,
            sessionIssuer:STRUCT<
                type:STRING,
                principalId:STRING,
                arn:STRING,
                accountId:STRING,
                userName:STRING>
        >
    >,
    eventTime STRING,
    eventSource STRING,
    eventName STRING,
    awsRegion STRING,
    sourceIpAddress STRING,
    userAgent STRING,
    errorCode STRING,
    errorMessage STRING,
    requestParameters STRING,
    responseElements STRING,
    additionalEventData STRING,
    requestId STRING,
    eventId STRING,
    resources ARRAY<STRUCT<ARN:STRING,accountId: STRING,type:STRING>>,
    eventType STRING,
```

```
apiVersion STRING,  
readOnly STRING,  
recipientAccountId STRING,  
serviceEventDetails STRING,  
sharedEventID STRING,  
vpcEndpointId STRING  
)  
PARTITIONED BY (region string, year string, month string, day string)  
ROW FORMAT SERDE 'com.amazon.emr.hive.serde.CloudTrailSerde'  
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'  
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'  
LOCATION 's3://myawsexamplebucket/AWSLogs/11112223333/';
```

Then, create the partitions individually. You can't get results from dates that you have not created.

```
ALTER TABLE s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table_partitioned ADD  
PARTITION (region= 'us-east-1', year= '2019', month= '02', day= '19') LOCATION 's3://  
myawsexamplebucket/AWSLogs/11112223333/CloudTrail/us-east-1/2019/02/19/'  
PARTITION (region= 'us-west-1', year= '2019', month= '02', day= '19') LOCATION 's3://  
myawsexamplebucket/AWSLogs/11112223333/CloudTrail/us-west-1/2019/02/19/'  
PARTITION (region= 'us-west-2', year= '2019', month= '02', day= '19') LOCATION 's3://  
myawsexamplebucket/AWSLogs/11112223333/CloudTrail/us-west-2/2019/02/19/'  
PARTITION (region= 'ap-southeast-1', year= '2019', month= '02', day= '19') LOCATION  
's3://myawsexamplebucket/AWSLogs/11112223333/CloudTrail/ap-southeast-1/2019/02/19/'  
PARTITION (region= 'ap-southeast-2', year= '2019', month= '02', day= '19') LOCATION  
's3://myawsexamplebucket/AWSLogs/11112223333/CloudTrail/ap-southeast-2/2019/02/19/'  
PARTITION (region= 'ap-northeast-1', year= '2019', month= '02', day= '19') LOCATION  
's3://myawsexamplebucket/AWSLogs/11112223333/CloudTrail/ap-northeast-1/2019/02/19/'  
PARTITION (region= 'eu-west-1', year= '2019', month= '02', day= '19') LOCATION 's3://  
myawsexamplebucket/AWSLogs/11112223333/CloudTrail/eu-west-1/2019/02/19/'  
PARTITION (region= 'sa-east-1', year= '2019', month= '02', day= '19') LOCATION 's3://  
myawsexamplebucket/AWSLogs/11112223333/CloudTrail/sa-east-1/2019/02/19/';
```

You can then make the request based on these partitions, and you don't need to load the full bucket.

```
SELECT useridentity.arn,  
Count(requestid) AS RequestCount  
FROM s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table_partitioned  
WHERE eventsource='s3.amazonaws.com'  
AND json_extract_scalar(additionalEventData, '$.SignatureVersion')='SigV2'  
AND region='us-east-1'  
AND year='2019'  
AND month='02'  
AND day='19'  
Group by useridentity.arn
```

Using AWS CloudTrail to Identify Access to Amazon S3 Objects

You can use your AWS CloudTrail event log to identify Amazon S3 object access requests for data events such as GetObject, DeleteObject, and PutObject, and discover further information about those requests.

The following example shows how to get all PUT object requests for Amazon S3 from the AWS CloudTrail event log.

Athena Query Example for Identifying Amazon S3 Object Access Requests

In the following Athena query examples, replace `<s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table>` with your Athena details, and modify the date range as needed.

Example — Select all events that have PUT object access requests, and print only EventTime, EventSource, SourceIP, UserAgent, BucketName, Object, and UserARN

```
SELECT
    eventTime,
    eventName,
    eventSource,
    sourceIpAddress,
    userAgent,
    requestParameters.bucketName as bucketName,
    requestParameters.key as object,
    userIdentity.arn as userArn
FROM
    s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table
WHERE
    eventName = 'PutObject'
    AND eventTime BETWEEN "2019-07-05T00:00:00Z" and "2019-07-06T00:00:00Z"
```

Example — Select all events that have GET object access requests, and print only EventTime, EventSource, SourceIP, UserAgent, BucketName, Object, and UserARN

```
SELECT
    eventTime,
    eventName,
    eventSource,
    sourceIpAddress,
    userAgent,
    requestParameters.bucketName as bucketName,
    requestParameters.key as object,
    userIdentity.arn as userArn
FROM
    s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table
WHERE
    eventName = 'GetObject'
    AND eventTime BETWEEN "2019-07-05T00:00:00Z" and "2019-07-06T00:00:00Z"
```

Example — Select all anonymous requester events to a bucket in a certain period and print only EventTime, EventSource, SourceIP, UserAgent, BucketName, UserIdentity, and UserARN

```
SELECT
    eventTime,
    eventName,
    eventSource,
    sourceIpAddress,
    userAgent,
    requestParameters.bucketName as bucketName,
    userIdentity.arn as userArn,
    userIdentity.principalId
FROM
    s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket_table
```

```
WHERE  
  userIdentity.principalId='ANONYMOUS_PRINCIPAL'  
AND  eventTime BETWEEN "2019-07-05T00:00:00Z" and "2019-07-06T00:00:00Z"
```

Note

- These query examples may also be useful for security monitoring. You can review the results for `PutObject` or `GetObject` calls from unexpected or unauthorized IP addresses/requesters and for identifying any anonymous requests to your buckets.
- This query only retrieves information from the time at which logging was enabled.

If you are using Amazon S3 server access logs, see [Using Amazon S3 Access Logs to Identify Object Access Requests](#) (p. 667).

Related Resources

- [AWS CloudTrail User Guide](#)
- [CloudTrail Event Reference](#)

Using BitTorrent with Amazon S3

Topics

- [How You are Charged for BitTorrent Delivery \(p. 636\)](#)
- [Using BitTorrent to Retrieve Objects Stored in Amazon S3 \(p. 637\)](#)
- [Publishing Content Using Amazon S3 and BitTorrent \(p. 637\)](#)

BitTorrent is an open, peer-to-peer protocol for distributing files. You can use the BitTorrent protocol to retrieve any publicly-accessible object in Amazon S3. This section describes why you might want to use BitTorrent to distribute your data out of Amazon S3 and how to do so.

Amazon S3 supports the BitTorrent protocol so that developers can save costs when distributing content at high scale. Amazon S3 is useful for simple, reliable storage of any data. The default distribution mechanism for Amazon S3 data is via client/server download. In client/server distribution, the entire object is transferred point-to-point from Amazon S3 to every authorized user who requests that object. While client/server delivery is appropriate for a wide variety of use cases, it is not optimal for everybody. Specifically, the costs of client/server distribution increase linearly as the number of users downloading objects increases. This can make it expensive to distribute popular objects.

BitTorrent addresses this problem by recruiting the very clients that are downloading the object as distributors themselves: Each client downloads some pieces of the object from Amazon S3 and some from other clients, while simultaneously uploading pieces of the same object to other interested "peers." The benefit for publishers is that for large, popular files the amount of data actually supplied by Amazon S3 can be substantially lower than what it would have been serving the same clients via client/server download. Less data transferred means lower costs for the publisher of the object.

Note

- Amazon S3 does not support the BitTorrent protocol in AWS Regions launched after May 30, 2016.
- You can get torrent only for objects that are less than 5 GB in size.

How You are Charged for BitTorrent Delivery

There is no extra charge for use of BitTorrent with Amazon S3. Data transfer via the BitTorrent protocol is metered at the same rate as client/server delivery. To be precise, whenever a downloading BitTorrent client requests a "piece" of an object from the Amazon S3 "seeder," charges accrue just as if an anonymous request for that piece had been made using the REST or SOAP protocol. These charges will appear on your Amazon S3 bill and usage reports in the same way. The difference is that if a lot of clients are requesting the same object simultaneously via BitTorrent, then the amount of data Amazon S3 must serve to satisfy those clients will be lower than with client/server delivery. This is because the BitTorrent clients are simultaneously uploading and downloading amongst themselves.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The data transfer savings achieved from use of BitTorrent can vary widely depending on how popular your object is. Less popular objects require heavier use of the "seeder" to serve clients, and thus the difference between BitTorrent distribution costs and client/server distribution costs might be small for such objects. In particular, if only one client is ever downloading a particular object at a time, the cost of BitTorrent delivery will be the same as direct download.

Using BitTorrent to Retrieve Objects Stored in Amazon S3

Any object in Amazon S3 that can be read anonymously can also be downloaded via BitTorrent. Doing so requires use of a BitTorrent client application. Amazon does not distribute a BitTorrent client application, but there are many free clients available. The Amazon S3BitTorrent implementation has been tested to work with the official BitTorrent client (go to <http://www.bittorrent.com/>).

The starting point for a BitTorrent download is a .torrent file. This small file describes for BitTorrent clients both the data to be downloaded and where to get started finding that data. A .torrent file is a small fraction of the size of the actual object to be downloaded. Once you feed your BitTorrent client application an Amazon S3 generated .torrent file, it should start downloading immediately from Amazon S3 and from any "peer" BitTorrent clients.

Retrieving a .torrent file for any publicly available object is easy. Simply add a "?torrent" query string parameter at the end of the REST GET request for the object. No authentication is required. Once you have a BitTorrent client installed, downloading an object using BitTorrent download might be as easy as opening this URL in your web browser.

There is no mechanism to fetch the .torrent for an Amazon S3 object using the SOAP API.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Example

This example retrieves the Torrent file for the "Nelson" object in the "quotes" bucket.

Sample Request

```
GET /quotes/Nelson?torrent HTTP/1.0
Date: Wed, 25 Nov 2009 12:00:00 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-request-id: 7CD745EBB7AB5ED9
Date: Wed, 25 Nov 2009 12:00:00 GMT
Content-Disposition: attachment; filename=Nelson.torrent;
Content-Type: application/x-bittorrent
Content-Length: 537
Server: AmazonS3

<body: a Bencoded dictionary as defined by the BitTorrent specification>
```

Publishing Content Using Amazon S3 and BitTorrent

Every anonymously readable object stored in Amazon S3 is automatically available for download using BitTorrent. The process for changing the ACL on an object to allow anonymous READ operations is described in [Identity and Access Management in Amazon S3 \(p. 301\)](#).

You can direct your clients to your BitTorrent accessible objects by giving them the .torrent file directly or by publishing a link to the ?torrent URL of your object. One important thing to note is that the .torrent file describing an Amazon S3 object is generated on-demand, the first time it is requested (via the REST ?torrent resource). Generating the .torrent for an object takes time proportional to the size of that object. For large objects, this time can be significant. Therefore, before publishing a ?torrent link, we suggest making the first request for it yourself. Amazon S3 might take several minutes to respond to this first request, as it generates the .torrent file. Unless you update the object in question, subsequent requests for the .torrent will be fast. Following this procedure before distributing a ?torrent link will ensure a smooth BitTorrent downloading experience for your customers.

To stop distributing a file using BitTorrent, simply remove anonymous access to it. This can be accomplished by either deleting the file from Amazon S3, or modifying your access control policy to prohibit anonymous reads. After doing so, Amazon S3 will no longer act as a "seeder" in the BitTorrent network for your file, and will no longer serve the .torrent file via the ?torrent REST API. However, after a .torrent for your file is published, this action might not stop public downloads of your object that happen exclusively using the BitTorrent peer to peer network.

Handling REST and SOAP Errors

Topics

- [The REST Error Response \(p. 639\)](#)
- [The SOAP Error Response \(p. 640\)](#)
- [Amazon S3 Error Best Practices \(p. 641\)](#)

This section describes REST and SOAP errors and how to handle them.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The REST Error Response

Topics

- [Response Headers \(p. 639\)](#)
- [Error Response \(p. 640\)](#)

If a REST request results in an error, the HTTP reply has:

- An XML error document as the response body
- Content-Type: application/xml
- An appropriate 3xx, 4xx, or 5xx HTTP status code

Following is an example of a REST Error Response.

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>NoSuchKey</Code>
  <Message>The resource you requested does not exist</Message>
  <Resource>/mybucket/myfoto.jpg</Resource>
  <RequestId>4442587FB7D0A2F9</RequestId>
</Error>
```

For more information about Amazon S3 errors, go to [ErrorCodeList](#).

Response Headers

Following are response headers returned by all operations:

- **x-amz-request-id**: A unique ID assigned to each request by the system. In the unlikely event that you have problems with Amazon S3, Amazon can use this to help troubleshoot the problem.
- **x-amz-id-2**: A special token that will help us to troubleshoot problems.

Error Response

Topics

- [Error Code \(p. 640\)](#)
- [Error Message \(p. 640\)](#)
- [Further Details \(p. 640\)](#)

When an Amazon S3 request is in error, the client receives an error response. The exact format of the error response is API specific: For example, the REST error response differs from the SOAP error response. However, all error responses have common elements.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Error Code

The error code is a string that uniquely identifies an error condition. It is meant to be read and understood by programs that detect and handle errors by type. Many error codes are common across SOAP and REST APIs, but some are API-specific. For example, `NoSuchKey` is universal, but `UnexpectedContent` can occur only in response to an invalid REST request. In all cases, SOAP fault codes carry a prefix as indicated in the table of error codes, so that a `NoSuchKey` error is actually returned in SOAP as `Client.NoSuchKey`.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Error Message

The error message contains a generic description of the error condition in English. It is intended for a human audience. Simple programs display the message directly to the end user if they encounter an error condition they don't know how or don't care to handle. Sophisticated programs with more exhaustive error handling and proper internationalization are more likely to ignore the error message.

Further Details

Many error responses contain additional structured data meant to be read and understood by a developer diagnosing programming errors. For example, if you send a `Content-MD5` header with a REST PUT request that doesn't match the digest calculated on the server, you receive a `BadDigest` error. The error response also includes as detail elements the digest we calculated, and the digest you told us to expect. During development, you can use this information to diagnose the error. In production, a well-behaved program might include this information in its error log.

The SOAP Error Response

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

In SOAP, an error result is returned to the client as a SOAP fault, with the HTTP response code 500. If you do not receive a SOAP fault, then your request was successful. The Amazon S3 SOAP fault code is

comprised of a standard SOAP 1.1 fault code (either "Server" or "Client") concatenated with the Amazon S3-specific error code. For example: "Server.InternalError" or "Client.NoSuchBucket". The SOAP fault string element contains a generic, human readable error message in English. Finally, the SOAP fault detail element contains miscellaneous information relevant to the error.

For example, if you attempt to delete the object "Fred", which does not exist, the body of the SOAP response contains a "NoSuchKey" SOAP fault.

Example

```
<soapenv:Body>
  <soapenv:Fault>
    <Faultcode>soapenv:Client.NoSuchKey</Faultcode>
    <Faultstring>The specified key does not exist.</Faultstring>
    <Detail>
      <Key>Fred</Key>
    </Detail>
  </soapenv:Fault>
</soapenv:Body>
```

For more information about Amazon S3 errors, go to [ErrorCodeList](#).

Amazon S3 Error Best Practices

When designing an application for use with Amazon S3, it is important to handle Amazon S3 errors appropriately. This section describes issues to consider when designing your application.

Retry InternalErrors

Internal errors are errors that occur within the Amazon S3 environment.

Requests that receive an InternalError response might not have processed. For example, if a PUT request returns InternalError, a subsequent GET might retrieve the old value or the updated value.

If Amazon S3 returns an InternalError response, retry the request.

Tune Application for Repeated SlowDown errors

As with any distributed system, S3 has protection mechanisms which detect intentional or unintentional resource over-consumption and react accordingly. SlowDown errors can occur when a high request rate triggers one of these mechanisms. Reducing your request rate will decrease or eliminate errors of this type. Generally speaking, most users will not experience these errors regularly; however, if you would like more information or are experiencing high or unexpected SlowDown errors, please post to our Amazon S3 developer forum <https://forums.aws.amazon.com/> or sign up for AWS Premium Support <https://aws.amazon.com/premiumsupport/>.

Isolate Errors

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 provides a set of error codes that are used by both the SOAP and REST API. The SOAP API returns standard Amazon S3 error codes. The REST API is designed to look like a standard HTTP server

and interact with existing HTTP clients (e.g., browsers, HTTP client libraries, proxies, caches, and so on). To ensure the HTTP clients handle errors properly, we map each Amazon S3 error to an HTTP status code.

HTTP status codes are less expressive than Amazon S3 error codes and contain less information about the error. For example, the `NoSuchKey` and `NoSuchBucket` Amazon S3 errors both map to the HTTP `404 Not Found` status code.

Although the HTTP status codes contain less information about the error, clients that understand HTTP, but not the Amazon S3 API, will usually handle the error correctly.

Therefore, when handling errors or reporting Amazon S3 errors to end users, use the Amazon S3 error code instead of the HTTP status code as it contains the most information about the error. Additionally, when debugging your application, you should also consult the human readable `<Details>` element of the XML error response.

Troubleshooting Amazon S3

This section describes how to troubleshoot Amazon S3 and explains how to get request IDs that you'll need when you contact AWS Support.

Topics

- [Troubleshooting Amazon S3 by Symptom \(p. 643\)](#)
- [Getting Amazon S3 Request IDs for AWS Support \(p. 644\)](#)
- [Related Topics \(p. 646\)](#)

Troubleshooting Amazon S3 by Symptom

The following topics lists symptoms to help you troubleshoot some of the issues that you might encounter when working with Amazon S3.

Symptoms

- [Significant Increases in HTTP 503 Responses to Amazon S3 Requests to Buckets with Versioning Enabled \(p. 643\)](#)
- [Unexpected Behavior When Accessing Buckets Set with CORS \(p. 643\)](#)

Significant Increases in HTTP 503 Responses to Amazon S3 Requests to Buckets with Versioning Enabled

If you notice a significant increase in the number of HTTP 503-slow down responses received for Amazon S3 PUT or DELETE object requests to a bucket that has versioning enabled, you might have one or more objects in the bucket for which there are millions of versions. When you have objects with millions of versions, Amazon S3 automatically throttles requests to the bucket to protect the customer from an excessive amount of request traffic, which could potentially impede other requests made to the same bucket.

To determine which S3 objects have millions of versions, use the Amazon S3 inventory tool. The inventory tool generates a report that provides a flat file list of the objects in a bucket. For more information, see [Amazon S3 Inventory \(p. 422\)](#).

The Amazon S3 team encourages customers to investigate applications that repeatedly overwrite the same S3 object, potentially creating millions of versions for that object, to determine whether the application is working as intended. If you have a use case that requires millions of versions for one or more S3 objects, contact the AWS Support team at [AWS Support](#) to discuss your use case and to help us assist you in determining the optimal solution for your use case scenario.

Unexpected Behavior When Accessing Buckets Set with CORS

If you encounter unexpected behavior when accessing buckets set with the cross-origin resource sharing (CORS) configuration, see [Troubleshooting CORS Issues \(p. 160\)](#).

Getting Amazon S3 Request IDs for AWS Support

Whenever you need to contact AWS Support due to encountering errors or unexpected behavior in Amazon S3, you will need to get the request IDs associated with the failed action. Getting these request IDs enables AWS Support to help you resolve the problems you're experiencing. Request IDs come in pairs, are returned in every response that Amazon S3 processes (even the erroneous ones), and can be accessed through verbose logs. There are a number of common methods for getting your request IDs including, S3 access logs and CloudTrail events/data events.

After you've recovered these logs, copy and retain those two values, because you'll need them when you contact AWS Support. For information about contacting AWS Support, see [Contact Us](#).

Topics

- [Using HTTP to Obtain Request IDs \(p. 644\)](#)
- [Using a Web Browser to Obtain Request IDs \(p. 644\)](#)
- [Using AWS SDKs to Obtain Request IDs \(p. 644\)](#)
- [Using the AWS CLI to Obtain Request IDs \(p. 646\)](#)

Using HTTP to Obtain Request IDs

You can obtain your request IDs, `x-amz-request-id` and `x-amz-id-2` by logging the bits of an HTTP request before it reaches the target application. There are a variety of third-party tools that can be used to recover verbose logs for HTTP requests. Choose one you trust, and run the tool, listening on the port that your Amazon S3 traffic travels on, as you send out another Amazon S3 HTTP request.

For HTTP requests, the pair of request IDs will look like the following examples.

```
x-amz-request-id: 79104EXAMPLEB723
x-amz-id-2: IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK+Jd1vEXAMPLEa3Km
```

Note

HTTPS requests are encrypted and hidden in most packet captures.

Using a Web Browser to Obtain Request IDs

Most web browsers have developer tools that allow you to view request headers.

For web browser-based requests that return an error, the pair of request IDs will look like the following examples.

```
<Error><Code>AccessDenied</Code><Message>Access Denied</Message>
<RequestId>79104EXAMPLEB723</RequestId><HostId>IOWQ4fDEXAMPLEQM
+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK+Jd1vEXAMPLEa3Km</HostId></Error>
```

For obtaining the request ID pair from successful requests, you'll need to use the developer tools to look at the HTTP response headers. For information about developer tools for specific browsers, see **Amazon S3 Troubleshooting - How to recover your S3 request IDs** in the AWS Developer Forums.

Using AWS SDKs to Obtain Request IDs

The following sections include information for configuring logging using an AWS SDK. While you can enable verbose logging on every request and response, you should not enable logging in production systems since large requests/responses can cause significant slowdown in an application.

For AWS SDK requests, the pair of request IDs will look like the following examples.

```
Status Code: 403, AWS Service: Amazon S3, AWS Request ID: 79104EXAMPLEB723
AWS Error Code: AccessDenied AWS Error Message: Access Denied
S3 Extended Request ID: IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK+Jd1vEXAMPLEa3Km
```

Using the SDK for PHP to Obtain Request IDs

You can configure logging using PHP. For more information, see [How can I see what data is sent over the wire?](#) in the FAQ for the *AWS SDK for PHP*.

Using the SDK for Java to Obtain Request IDs

You can enable logging for specific requests or responses, allowing you to catch and return only the relevant headers. To do this, import the `com.amazonaws.services.s3.s3ResponseMetadata` class. Afterwards, you can store the request in a variable before performing the actual request. Call `getCachedResponseMetadata(AmazonWebServiceRequest request).getRequestID()` to get the logged request or response.

Example

```
PutObjectRequest req = new PutObjectRequest(bucketName, key, createSampleFile());
s3.putObject(req);
S3ResponseMetadata md = s3.getCachedResponseMetadata(req);
System.out.println("Host ID: " + md.getHostId() + " RequestID: " + md.getRequestId());
```

Alternatively, you can use verbose logging of every Java request and response. For more information, see [Verbose Wire Logging](#) in the Logging AWS SDK for Java Calls topic in the *AWS SDK for Java Developer Guide*.

Using the AWS SDK for .NET to Obtain Request IDs

You can configure logging in AWS SDK for .NET using the built-in `System.Diagnostics` logging tool. For more information, see the [Logging with the AWS SDK for .NET](#) AWS Developer Blog post.

Note

By default, the returned log contains only error information. The config file needs to have `AWSTLogMetrics` (and optionally, `AWSResponseLogging`) added to get the request IDs.

Using the SDK for Python to Obtain Request IDs

You can configure logging in Python by adding the following lines to your code to output debug information to a file.

```
import logging
logging.basicConfig(filename="mylog.log", level=logging.DEBUG)
```

If you're using the Boto Python interface for AWS, you can set the debug level to two as per the Boto docs, [here](#).

Using the SDK for Ruby to Obtain Request IDs

You can get your request IDs using either the SDK for Ruby - Version 1, Version 2, or Version 3.

- **Using the SDK for Ruby - Version 1**– You can enable HTTP wire logging globally with the following line of code.

```
s3 = AWS::S3.new(:logger => Logger.new($stdout), :http_wire_trace => true)
```

- **Using the SDK for Ruby - Version 2 or Version 3**– You can enable HTTP wire logging globally with the following line of code.

```
s3 = Aws::S3::Client.new(:logger => Logger.new($stdout), :http_wire_trace => true)
```

Using the AWS CLI to Obtain Request IDs

You can get your request IDs in the AWS CLI by adding `--debug` to your command.

Related Topics

For other troubleshooting and support topics, see the following:

- [Troubleshooting CORS Issues \(p. 160\)](#)
- [Handling REST and SOAP Errors \(p. 639\)](#)
- [AWS Support Documentation](#)

For troubleshooting information regarding third-party tools, see [Getting Amazon S3 request IDs](#) in the AWS Developer Forums.

Amazon S3 Server Access Logging

Server access logging provides detailed records for the requests that are made to a bucket. Server access logs are useful for many applications. For example, access log information can be useful in security and access audits. It can also help you learn about your customer base and understand your Amazon S3 bill.

Note

Server access logs do not log information regarding wrong-region redirect errors for Regions that launched after March 20, 2019. Wrong-region redirect errors occur when a request for an object/bucket is made to an endpoint other than the endpoint of the Region in which the bucket exists.

Topics

- [How to Enable Server Access Logging \(p. 647\)](#)
- [Log Object Key Format \(p. 648\)](#)
- [How Are Logs Delivered? \(p. 649\)](#)
- [Best Effort Server Log Delivery \(p. 649\)](#)
- [Bucket Logging Status Changes Take Effect Over Time \(p. 649\)](#)
- [Enabling Logging Using the Console \(p. 649\)](#)
- [Enabling Logging Programmatically \(p. 650\)](#)
- [Amazon S3 Server Access Log Format \(p. 653\)](#)
- [Deleting Amazon S3 Log Files \(p. 661\)](#)
- [Using Amazon S3 Access Logs to Identify Requests \(p. 662\)](#)

How to Enable Server Access Logging

To track requests for access to your bucket, you can enable server access logging. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and an error code, if relevant.

Note

There is no extra charge for enabling server access logging on an Amazon S3 bucket. However, any log files that the system delivers to you accrue the usual charges for storage. (You can delete the log files at any time.) No data transfer charges are assessed for log file delivery, but access to the delivered log files is charged the same as any other data transfer.

By default, logging is disabled. When logging is enabled, logs are saved to a bucket in the same AWS Region as the source bucket.

To enable access logging, you must do the following:

- Turn on the log delivery by adding logging configuration on the bucket for which you want Amazon S3 to deliver access logs. We refer to this bucket as the *source bucket*.
- Grant the Amazon S3 Log Delivery group write permission on the bucket where you want the access logs saved. We refer to this bucket as the *target bucket*.

Note

- Amazon S3 only supports granting permission to deliver access logs via bucket ACL, not via bucket policy.

- Adding *deny* conditions to a bucket policy may prevent Amazon S3 from delivering access logs.
- [Default bucket encryption](#) on the destination bucket *may only be used* if **AES256 (SSE-S3)** is selected. SSE-KMS encryption is not supported.
- Amazon S3 object lock cannot be enabled on the log destination bucket.

To turn on log delivery, you provide the following logging configuration information:

- The name of the target bucket where you want Amazon S3 to save the access logs as objects. You can have logs delivered to any bucket that you own that is in the same Region as the source bucket, including the source bucket itself.

We recommend that you save access logs in a different bucket so that you can easily manage the logs. If you choose to save access logs in the source bucket, we recommend that you specify a prefix for all log object keys so that the object names begin with a common string and the log objects are easier to identify.

When your source bucket and target bucket are the same bucket, additional logs are created for the logs that are written to the bucket. This behavior might not be ideal for your use case because it could result in a small increase in your storage billing. In addition, the extra logs about logs might make it harder to find the log that you're looking for.

Note

Both the source and target buckets must be owned by the same AWS account, and the buckets must both be in the same Region.

- (Optional) A prefix for Amazon S3 to assign to all log object keys. The prefix makes it simpler for you to locate the log objects.

For example, if you specify the prefix value `logs/`, each log object that Amazon S3 creates begins with the `logs/` prefix in its key, as in this example:

```
logs/2013-11-01-21-32-16-E568B2907131C0C0
```

The key prefix can help when you delete the logs. For example, you can set a lifecycle configuration rule for Amazon S3 to delete objects with a specific key prefix. For more information, see [Deleting Amazon S3 Log Files \(p. 661\)](#).

- (Optional) Permissions so that others can access the generated logs. By default, the bucket owner always has full access to the log objects. You can optionally grant access to other users.

For more information about enabling server access logging, see [Enabling Logging Using the Console \(p. 649\)](#) and [Enabling Logging Programmatically \(p. 650\)](#).

Log Object Key Format

Amazon S3 uses the following object key format for the log objects it uploads in the target bucket:

```
TargetPrefixYYYY-mm-DD-HH-MM-SS-UniqueString
```

In the key, `YYYY`, `mm`, `DD`, `HH`, `MM`, and `SS` are the digits of the year, month, day, hour, minute, and seconds (respectively) when the log file was delivered, these dates and times are in Coordinated Universal time (UTC).

A log file delivered at a specific time can contain records written at any point before that time. There is no way to know whether all log records for a certain time interval have been delivered or not.

The `UniqueString` component of the key is there to prevent overwriting of files. It has no meaning, and log processing software should ignore it.

How Are Logs Delivered?

Amazon S3 periodically collects access log records, consolidates the records in log files, and then uploads log files to your target bucket as log objects. If you enable logging on multiple source buckets that identify the same target bucket, the target bucket will have access logs for all those source buckets. However, each log object reports access log records for a specific source bucket.

Amazon S3 uses a special log delivery account, called the Log Delivery group, to write access logs. These writes are subject to the usual access control restrictions. You must grant the Log Delivery group write permission on the target bucket by adding a grant entry in the bucket's access control list (ACL). If you use the Amazon S3 console to enable logging on a bucket, the console both enables logging on the source bucket and updates the ACL on the target bucket to grant write permission to the Log Delivery group.

Best Effort Server Log Delivery

Server access log records are delivered on a best effort basis. Most requests for a bucket that is properly configured for logging result in a delivered log record. Most log records are delivered within a few hours of the time that they are recorded, but they can be delivered more frequently.

The completeness and timeliness of server logging is not guaranteed. The log record for a particular request might be delivered long after the request was actually processed, or *it might not be delivered at all*. The purpose of server logs is to give you an idea of the nature of traffic against your bucket. It is rare to lose log records, but server logging is not meant to be a complete accounting of all requests.

It follows from the best-effort nature of the server logging feature that the usage reports available at the AWS portal (Billing and Cost Management reports on the [AWS Management Console](#)) might include one or more access requests that do not appear in a delivered server log.

Bucket Logging Status Changes Take Effect Over Time

Changes to the logging status of a bucket take time to actually affect the delivery of log files. For example, if you enable logging for a bucket, some requests made in the following hour might be logged, while others might not. If you change the target bucket for logging from bucket A to bucket B, some logs for the next hour might continue to be delivered to bucket A, while others might be delivered to the new target bucket B. In all cases, the new settings eventually take effect without any further action on your part.

Enabling Logging Using the Console

For information about enabling [Amazon S3 Server Access Logging \(p. 647\)](#) in the [AWS Management Console](#), see [How Do I Enable Server Access Logging for an S3 Bucket?](#) in the *Amazon Simple Storage Service Console User Guide*.

When you enable logging on a bucket, the console both enables logging on the source bucket and adds a grant in the target bucket's access control list (ACL) granting write permission to the Log Delivery group.

For information about how to enable logging programmatically, see [Enabling Logging Programmatically](#) (p. 650).

For information about the log record format, including the list of fields and their descriptions, see [Amazon S3 Server Access Log Format](#) (p. 653).

Enabling Logging Programmatically

You can enable or disable logging programmatically by using either the Amazon S3 API or the AWS SDKs. To do so, you both enable logging on the bucket and grant the Log Delivery group permission to write logs to the target bucket.

Topics

- [Enabling Logging](#) (p. 650)
- [Granting the Log Delivery Group WRITE and READ_ACP Permissions](#) (p. 650)
- [Example: AWS SDK for .NET](#) (p. 651)
- [Related Resources](#) (p. 652)

Enabling Logging

To enable logging, you submit a [PUT Bucket logging](#) request to add the logging configuration on the source bucket. The request specifies the target bucket and, optionally, the prefix to be used with all log object keys. The following example identifies logbucket as the target bucket and logs/ as the prefix.

```
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>logbucket</TargetBucket>
    <TargetPrefix>logs/</TargetPrefix>
  </LoggingEnabled>
</BucketLoggingStatus>
```

The log objects are written and owned by the Log Delivery account, and the bucket owner is granted full permissions on the log objects. In addition, you can optionally grant permissions to other users so that they can access the logs. For more information, see [PUT Bucket logging](#).

Amazon S3 also provides the [GET Bucket logging](#) API to retrieve logging configuration on a bucket. To delete the logging configuration, you send the PUT Bucket logging request with an empty BucketLoggingStatus.

```
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
</BucketLoggingStatus>
```

You can use either the Amazon S3 API or the AWS SDK wrapper libraries to enable logging on a bucket.

Granting the Log Delivery Group WRITE and READ_ACP Permissions

Amazon S3 writes the log files to the target bucket as a member of the predefined Amazon S3 group Log Delivery. These writes are subject to the usual access control restrictions. You must grant s3:GetObjectAcl and s3:PutObject permissions to this group by adding grants to the access control list (ACL) of the target bucket. The Log Delivery group is represented by the following URL.

```
http://acs.amazonaws.com/groups/s3/LogDelivery
```

To grant WRITE and READ_ACP permissions, add the following grants. For information about ACLs, see [Managing Access with ACLs \(p. 403\)](#).

```
<Grant>
  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
    <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
  </Grantee>
  <Permission>WRITE</Permission>
</Grant>
<Grant>
  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
    <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
  </Grantee>
  <Permission>READ_ACP</Permission>
</Grant>
```

For examples of adding ACL grants programmatically using the AWS SDKs, see [Managing ACLs Using the AWS SDK for Java \(p. 409\)](#) and [Managing ACLs Using the AWS SDK for .NET \(p. 411\)](#).

Example: AWS SDK for .NET

The following C# example enables logging on a bucket. You need to create two buckets, a source bucket and a target bucket. The example first grants the Log Delivery group the necessary permission to write logs to the target bucket and then enables logging on the source bucket. For more information, see [Enabling Logging Programmatically \(p. 650\)](#). For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 678\)](#).

Example

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ServerAccessLoggingTest
    {
        private const string bucketName = "*** bucket name for which to enable logging ***";
        private const string targetBucketName = "*** bucket name where you want access logs stored ***";
        private const string logObjectKeyPrefix = "Logs";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            {
                client = new AmazonS3Client(bucketRegion);
                EnableLoggingAsync().Wait();
            }

            private static async Task EnableLoggingAsync()
            {
                {
                    try
                    {

```

```

        // Step 1 - Grant Log Delivery group permission to write log to the target
        bucket.
            await GrantPermissionsToWriteLogsAsync();
            // Step 2 - Enable logging on the source bucket.
            await EnableDisableLoggingAsync();
        }
        catch (AmazonS3Exception e)
        {
            Console.WriteLine("Error encountered on server. Message:'{0}' when writing
an object", e.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
        }
    }

    private static async Task GrantPermissionsToWriteLogsAsync()
    {
        var bucketACL = new S3AccessControlList();
        var aclResponse = client.GetACL(new GetACLRequest { BucketName =
targetBucketName });
        bucketACL = aclResponse.AccessControlList;
        bucketACL.AddGrant(new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" }, S3Permission.WRITE);
        bucketACL.AddGrant(new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" }, S3Permission.READ_ACP);
        var setACLRequest = new PutACLRequest
        {
            AccessControlList = bucketACL,
            BucketName = targetBucketName
        };
        await client.PutACLAsync(setACLRequest);
    }

    private static async Task EnableDisableLoggingAsync()
    {
        var loggingConfig = new S3BucketLoggingConfig
        {
            TargetBucketName = targetBucketName,
            TargetPrefix = logObjectKeyPrefix
        };

        // Send request.
        var putBucketLoggingRequest = new PutBucketLoggingRequest
        {
            BucketName = bucketName,
            LoggingConfig = loggingConfig
        };
        await client.PutBucketLoggingAsync(putBucketLoggingRequest);
    }
}

```

Related Resources

- [Amazon S3 Server Access Logging \(p. 647\)](#)
- [AWS::S3::Bucket](#) in the *AWS CloudFormation User Guide*

Amazon S3 Server Access Log Format

This section describes the Amazon S3 server access log files.

Topics

- [Additional Logging for Copy Operations \(p. 657\)](#)
- [Custom Access Log Information \(p. 661\)](#)
- [Programming Considerations for Extensible Server Access Log Format \(p. 661\)](#)

The server access log files consist of a sequence of newline-delimited log records. Each log record represents one request and consists of space-delimited fields. The following is an example log consisting of five log records.

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket [06/Feb/2019:00:00:38 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be 3E57427F3EXAMPLE
REST.GET.VERSIONING - "GET /awsexamplebucket?versioning HTTP/1.1" 200 - 113 - 7 - "-"
"S3Console/0.4" - s9lzHYrFp76ZVxRcpX9+5cjAnEH2ROuNkd2BHfIa6UkFVdtjf5mKR3/eTPFvsiP/XV/
VLI31234= SigV2 ECDHE-RSA-AES128-GCM-SHA256 AuthHeader awsexamplebucket.s3.amazonaws.com
TL SV1.1
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket [06/Feb/2019:00:00:38 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be 891CE47D2EXAMPLE
REST.GET.LOGGING_STATUS - "GET /awsexamplebucket?logging HTTP/1.1" 200 - 242
- 11 - "-" "S3Console/0.4" - 9vKBE6vMhrNiWHZmb2L0mXOcPGzQOI5XlnCtZNPxev+Hf
+7tpT6sxDwDty4LHBUOZJG96N1234= SigV2 ECDHE-RSA-AES128-GCM-SHA256 AuthHeader
awsexamplebucket.s3.amazonaws.com TL SV1.1
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket [06/Feb/2019:00:00:38 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be A1206F460EXAMPLE
REST.GET.BUCKETPOLICY - "GET /awsexamplebucket?policy HTTP/1.1" 404
NoSuchBucketPolicy 297 - 38 - "-" "S3Console/0.4" - BNaBsXZQQDbssi6xMBdBU2sLt
+Yf5kZDmeBUP35sFoKa3sLLeMC78iweIWXs99CRUrbS4n11234= SigV2 ECDHE-RSA-AES128-GCM-SHA256
AuthHeader awsexamplebucket.s3.amazonaws.com TL SV1.1
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket [06/Feb/2019:00:01:00 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be 7B4A0FABBEXAMPLE
REST.GET.VERSIONING - "GET /awsexamplebucket?versioning HTTP/1.1" 200 - 113 - 33 - "-"
"S3Console/0.4" - KelbUcazaN1jWuUlpJaxF64cQVpUEhoZKEG/hmy/gijN/I1DeWqDfFvnpbyfEseEME/
u7ME1234= SigV2 ECDHE-RSA-AES128-GCM-SHA256 AuthHeader awsexamplebucket.s3.amazonaws.com
TL SV1.1
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket [06/Feb/2019:00:01:57 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
DD6CC733AEXAMPLE REST.PUT.OBJECT s3-dg.pdf "PUT /awsexamplebucket/
s3-dg.pdf HTTP/1.1" 200 - - 4406583 41754 28 "-" "S3Console/0.4" -
10S62Zv81kBW7BB6SX4XJ48o6kpcl6LPwEoizZQQxJd5qDSCTLX0TgS37kYUBKQW3+bPdrG1234= SigV4 ECDHE-
RSA-AES128-SHA AuthHeader awsexamplebucket.s3.amazonaws.com TL SV1.1
```

Note

Any field can be set to - to indicate that the data was unknown or unavailable, or that the field was not applicable to this request.

The following list describes the log record fields.

Bucket Owner

The canonical user ID of the owner of the source bucket. The canonical user ID is another form of the AWS account ID. For more information about the canonical user ID, see [AWS Account Identifiers](#).

For information about how to find the canonical user ID for your account, see [Finding Your Account Canonical User ID](#).

Example Entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Bucket

The name of the bucket that the request was processed against. If the system receives a malformed request and cannot determine the bucket, the request will not appear in any server access log.

Example Entry

```
awsexamplebucket
```

Time

The time at which the request was received; these dates and times are in Coordinated Universal time (UTC). The format, using `strftime()` terminology, is as follows: `[%d/%b/%Y:%H:%M:%S %z]`

Example Entry

```
[06/Feb/2019:00:00:38 +0000]
```

Remote IP

The apparent internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.

Example Entry

```
192.0.2.3
```

Requester

The canonical user ID of the requester, or a – for unauthenticated requests. If the requester was an IAM user, this field returns the requester's IAM user name along with the AWS root account that the IAM user belongs to. This identifier is the same one used for access control purposes.

Example Entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Request ID

A string generated by Amazon S3 to uniquely identify each request.

Example Entry

```
3E57427F33A59F07
```

Operation

The operation listed here is declared as SOAP.*operation*, REST.*HTTP_method.resource_type*, WEBSITE.*HTTP_method.resource_type*, or BATCH.DELETE.OBJECT.

Example Entry

```
REST.PUT.OBJECT
```

Key

The "key" part of the request, URL encoded, or "-" if the operation does not take a key parameter.

Example Entry

```
/photos/2019/08/puppy.jpg
```

Request-URI

The Request-URI part of the HTTP request message.

Example Entry

```
"GET /awsexamplebucket/photos/2019/08/puppy.jpg?x-foo=bar HTTP/1.1"
```

HTTP status

The numeric HTTP status code of the response.

Example Entry

```
200
```

Error Code

The Amazon S3 [Error Code \(p. 640\)](#), or "-" if no error occurred.

Example Entry

```
NoSuchBucket
```

Bytes Sent

The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.

Example Entry

```
2662992
```

Object Size

The total size of the object in question.

Example Entry

```
3462992
```

Total Time

The number of milliseconds the request was in flight from the server's perspective. This value is measured from the time your request is received to the time that the last byte of the response is sent. Measurements made from the client's perspective might be longer due to network latency.

Example Entry

```
70
```

Turn-Around Time

The number of milliseconds that Amazon S3 spent processing your request. This value is measured from the time the last byte of your request was received until the time the first byte of the response was sent.

Example Entry

```
10
```

Referrer

The value of the HTTP Referrer header, if present. HTTP user-agents (for example, browsers) typically set this header to the URL of the linking or embedding page when making a request.

Example Entry

```
"http://www.amazon.com/webservices"
```

User-Agent

The value of the HTTP User-Agent header.

Example Entry

```
"curl/7.15.1"
```

Version Id

The version ID in the request, or "-" if the operation does not take a `versionId` parameter.

Example Entry

```
3HL4kqtJvjVBH40NrjfkD
```

Host Id

The x-amz-id-2 or Amazon S3 extended request ID.

Example Entry

```
s9lzHYrFp76ZVxRcpX9+5cjAnEH2ROuNkd2BHfIa6UkFVdtjf5mKR3/eTPFvsiP/XV/VLi31234=
```

Signature Version

The signature version, `SigV2` or `SigV4`, that was used to authenticate the request or a - for unauthenticated requests.

Example Entry

```
SigV2
```

Cipher Suite

The Secure Sockets Layer (SSL) cipher that was negotiated for HTTPS request or a – for HTTP.

Example Entry

```
ECDHE-RSA-AES128-GCM-SHA256
```

Authentication Type

The type of request authentication used, `AuthHeader` for authentication headers, `QueryString` for query string (pre-signed URL) or a – for unauthenticated requests.

Example Entry

```
AuthHeader
```

Host Header

The endpoint used to connect to Amazon S3

Example Entry

```
s3-us-west-2.amazonaws.com
```

TLS version

The Transport Layer Security (TLS) version negotiated by the client. The value is one of following: `TLSv1`, `TLSv1.1`, `TLSv1.2`; or – if TLS wasn't used.

Example Entry

```
TLSv1.2
```

Additional Logging for Copy Operations

A copy operation involves a `GET` and a `PUT`. For that reason, we log two records when performing a copy operation. The previous table describes the fields related to the `PUT` part of the operation. The following list describes the fields in the record that relate to the `GET` part of the copy operation.

Bucket Owner

The canonical user ID of the bucket that stores the object being copied. The canonical user ID is another form of the AWS account ID. For more information about the canonical user ID, see [AWS Account Identifiers](#). For information about how to find the canonical user ID for your account, see [Finding Your Account Canonical User ID](#).

Example Entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Bucket

The name of the bucket that stores the object being copied.

Example Entry

```
awsexamplebucket
```

Time

The time at which the request was received; these dates and times are in Coordinated Universal time (UTC). The format, using `strftime()` terminology, is as follows: `[%d/%B/%Y:%H:%M:%S %z]`

Example Entry

```
[06/Feb/2019:00:00:38 +0000]
```

Remote IP

The apparent internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.

Example Entry

```
192.0.2.3
```

Requester

The canonical user ID of the requester, or a `-` for unauthenticated requests. If the requester was an IAM user, this field will return the requester's IAM user name along with the AWS root account that the IAM user belongs to. This identifier is the same one used for access control purposes.

Example Entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Request ID

A string generated by Amazon S3 to uniquely identify each request.

Example Entry

```
3E57427F33A59F07
```

Operation

The operation listed here is declared as SOAP.*operation*, REST.*HTTP_method.resource_type*, WEBSITE.*HTTP_method.resource_type*, or BATCH.DELETE.OBJECT.

Example Entry

```
REST.COPY.OBJECT_GET
```

Key

The "key" of the object being copied or `-` if the operation does not take a key parameter.

Example Entry

```
/photos/2019/08/puppy.jpg
```

Request-URI

The Request-URI part of the HTTP request message.

Example Entry

```
"GET /awsexamplebucket/photos/2019/08/puppy.jpg?x-foo=bar"
```

HTTP status

The numeric HTTP status code of the `GET` portion of the copy operation.

Example Entry

```
200
```

Error Code

The Amazon S3 [Error Code \(p. 640\)](#), of the `GET` portion of the copy operation or "-" if no error occurred.

Example Entry

```
NoSuchBucket
```

Bytes Sent

The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.

Example Entry

```
2662992
```

Object Size

The total size of the object in question.

Example Entry

```
3462992
```

Total Time

The number of milliseconds the request was in flight from the server's perspective. This value is measured from the time your request is received to the time that the last byte of the response is sent. Measurements made from the client's perspective might be longer due to network latency.

Example Entry

```
70
```

Turn-Around Time

The number of milliseconds that Amazon S3 spent processing your request. This value is measured from the time the last byte of your request was received until the time the first byte of the response was sent.

Example Entry

10

Referrer

The value of the HTTP Referrer header, if present. HTTP user-agents (for example, browsers) typically set this header to the URL of the linking or embedding page when making a request.

Example Entry

"http://www.amazon.com/webservices"

User-Agent

The value of the HTTP User-Agent header.

Example Entry

"curl/7.15.1"

Version Id

The version ID of the object being copied or "-" if the `x-amz-copy-source` header didn't specify a `versionId` parameter as part of the copy source.

Example Entry

3HL4kqtJvjVBH40NrjfkD

Host Id

The `x-amz-id-2` or Amazon S3 extended request ID.

Example Entry

s9lzHYrFp76ZVxRcpX9+5cjAnEH2ROuNkd2BHfIa6UkFVdtjf5mKR3/eTPFvsiP/XV/VLi31234=

Signature Version

The signature version, `SigV2` or `SigV4`, that was used to authenticate the request or a - for unauthenticated requests.

Example Entry

SigV2

Cipher Suite

The Secure Sockets Layer (SSL) cipher that was negotiated for HTTPS request or a - for HTTP.

Example Entry

ECDHE-RSA-AES128-GCM-SHA256

Authentication Type

The type of request authentication used, `AuthHeader` for authentication headers, `QueryString` for query string (pre-signed URL) or a - for unauthenticated requests.

Example Entry

```
AuthHeader
```

Host Header

The endpoint used to connect to Amazon S3

Example Entry

```
s3-us-west-2.amazonaws.com
```

TLS version

The Transport Layer Security (TLS) version negotiated by the client. The value is one of following: TLSv1, TLSv1.1, TLSv1.2; or – if TLS wasn't used.

Example Entry

```
TLSv1.2
```

Custom Access Log Information

You can include custom information to be stored in the access log record for a request by adding a custom query-string parameter to the URL for the request. Amazon S3 ignores query-string parameters that begin with "x-", but includes those parameters in the access log record for the request, as part of the Request-URI field of the log record. For example, a GET request for "s3.amazonaws.com/awsexamplebucket/photos/2019/08/puppy.jpg?x-user=johndoe" works the same as the same request for "s3.amazonaws.com/awsexamplebucket/photos/2019/08/puppy.jpg", except that the "x-user=johndoe" string is included in the Request-URI field for the associated log record. This functionality is available in the REST interface only.

Programming Considerations for Extensible Server Access Log Format

From time to time, we might extend the access log record format by adding new fields to the end of each line. Code that parses server access logs must be written to handle trailing fields that it does not understand.

Deleting Amazon S3 Log Files

An S3 bucket with server access logging enabled can accumulate many server log objects over time. Your application might need these access logs for a specific period after creation, and after that, you might want to delete them. You can use Amazon S3 lifecycle configuration to set rules so that Amazon S3 automatically queues these objects for deletion at the end of their life.

You can define a lifecycle configuration for a subset of objects in your S3 bucket by using a shared prefix (that is, objects that have names that begin with a common string). If you specified a prefix in your server access logging configuration, you can set a lifecycle configuration rule to delete log objects that have that prefix. For example, if your log objects have the prefix `logs/`, you can set a lifecycle configuration rule to delete all objects in the bucket that have the prefix `/logs` after a specified period of time. For more information about lifecycle configuration, see [Object Lifecycle Management \(p. 119\)](#).

Related Resources

[Amazon S3 Server Access Logging \(p. 647\)](#)

Using Amazon S3 Access Logs to Identify Requests

You can identify Amazon S3 requests using Amazon S3 access logs.

Note

- We recommend that you use AWS CloudTrail data events instead of Amazon S3 access logs. CloudTrail data events are easier to set up and contain more information. For more information, see [Using AWS CloudTrail to Identify Amazon S3 Requests \(p. 628\)](#).
- Depending on how many access requests you get, it may require more resources and/or more time to analyze your logs.

Topics

- [Enabling Amazon S3 Access Logs for Requests \(p. 662\)](#)
- [Querying Amazon S3 Access Logs for Requests \(p. 664\)](#)
- [Using Amazon S3 Access Logs to Identify Signature Version 2 Requests \(p. 666\)](#)
- [Using Amazon S3 Access Logs to Identify Object Access Requests \(p. 667\)](#)
- [Related Resources \(p. 668\)](#)

Enabling Amazon S3 Access Logs for Requests

We recommend that you create a dedicated logging bucket in each AWS Region that you have S3 buckets in. Then have the Amazon S3 access log delivered to that S3 bucket.

Example — Enable access logs with five buckets across two Regions

In this example, you have the following five buckets:

- 1-awsexamplebucket-us-east-1
- 2-awsexamplebucket-us-east-1
- 3-awsexamplebucket-us-east-1
- 1-awsexamplebucket-us-west-2
- 2-awsexamplebucket-us-west-2

1. Create two logging buckets in the following Regions:

- awsexamplebucket-logs-us-east-1
- awsexamplebucket-logs-us-west-2

2. Then enable the Amazon S3 access logs as follows:

- 1-awsexamplebucket-us-east-1 logs to `s3://awsexamplebucket-logs-us-east-1/1-awsexamplebucket-us-east-1`
- 2-awsexamplebucket-us-east-1 logs to `s3://awsexamplebucket-logs-us-east-1/2-awsexamplebucket-us-east-1`

- 1-awsexamplebucket-us-east-1 logs to s3://awsexamplebucket-logs-us-east-1/3-awsexamplebucket-us-east-1
 - 1-awsexamplebucket-us-west-2 logs to s3://awsexamplebucket-logs-us-west-2/1-awsexamplebucket-us-west-2
 - 2-awsexamplebucket-us-west-2 logs to s3://awsexamplebucket-logs-us-west-2/2-awsexamplebucket-us-west-2
3. You can then enable the Amazon S3 access logs using the following methods:
- Using the [AWS Management Console](#) or,
 - [Enabling Logging Programmatically \(p. 650\)](#) or,
 - Using the [AWS CLI put-bucket-logging command](#) to programmatically enable access logs on a bucket using the following commands:

1. First, grant Amazon S3 permission using put-bucket-acl:

```
aws s3api put-bucket-acl --bucket awsexamplebucket-logs --grant-write
URI=http://acs.amazonaws.com/groups/s3/LogDelivery --grant-read-acp URI=http://
acs.amazonaws.com/groups/s3/LogDelivery
```

2. Then, apply the logging policy:

```
aws s3api put-bucket-logging --bucket awsexamplebucket --bucket-logging-status
file://logging.json
```

Logging.json is a JSON document in the current folder that contains the logging policy:

```
{
  "LoggingEnabled": {
    "TargetBucket": "awsexamplebucket-logs",
    "TargetPrefix": "awsexamplebucket/",
    "TargetGrants": [
      {
        "Grantee": {
          "Type": "AmazonCustomerByEmail",
          "EmailAddress": "user@example.com"
        },
        "Permission": "FULL_CONTROL"
      }
    ]
  }
}
```

Note

The put-bucket-acl command is required to grant the Amazon S3 log delivery system the necessary permissions (write and read-acp permissions).

3. Use a bash script to add access logging for all the buckets in your account:

```
loggingBucket='awsexamplebucket-logs'
region='us-west-2'
```

```
# Create Logging bucket
aws s3 mb s3://$loggingBucket --region $region

aws s3api put-bucket-acl --bucket $loggingBucket --grant-write URI=http://
acs.amazonaws.com/groups/s3/LogDelivery --grant-read-acp URI=http://
acs.amazonaws.com/groups/s3/LogDelivery

# List buckets in this account
buckets="$(aws s3 ls | awk '{print $3}')"

# Put bucket logging on each bucket
for bucket in $bucenable-logging-programmingkets
do printf '{
  "LoggingEnabled": {
    "TargetBucket": "%s",
    "TargetPrefix": "%s/"
  }
}' "$loggingBucket" "$bucket" > logging.json
aws s3api put-bucket-logging --bucket $bucket --bucket-logging-status
file://logging.json
echo "$bucket done"
done

rm logging.json

echo "Complete"
```

Note

This only works if all your buckets are in the same Region. If you have buckets in multiple Regions, you must adjust the script.

Querying Amazon S3 Access Logs for Requests

Amazon S3 stores server access logs as objects in an S3 bucket. It is often easier to use a tool that can analyze the logs in Amazon S3. Athena supports analysis of S3 objects and can be used to query Amazon S3 access logs.

Example

The following example shows how you can query Amazon S3 server access logs in Amazon Athena.

Note

To specify the Amazon S3 location in an Athena query, you need the target bucket name and the target prefix, as follows: `s3://awsexamplebucket-logs/prefix/`

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. In the Query Editor, run a command similar to the following:

```
create database s3_access_logs_db
```

Note

It's a best practice to create the database in the same AWS Region as your S3 bucket.

3. In the Query Editor, run a command similar to the following to create a table schema in the database that you created in step 2. The `STRING` and `BIGINT` data type values are the access log properties. You can query these properties in Athena. For `LOCATION`, enter the S3 bucket and prefix path as noted earlier.

```
CREATE EXTERNAL TABLE IF NOT EXISTS s3_access_logs_db.mybucket_logs(
  BucketOwner STRING,
  Bucket STRING,
  RequestDateTime STRING,
  RemoteIP STRING,
  Requester STRING,
  RequestID STRING,
  Operation STRING,
  Key STRING,
  RequestURI_operation STRING,
  RequestURI_key STRING,
  RequestURI_httpProtoversion STRING,
  HTTPstatus STRING,
  ErrorCode STRING,
  BytesSent BIGINT,
  ObjectSize BIGINT,
  TotalTime STRING,
  TurnAroundTime STRING,
  Referrer STRING,
  UserAgent STRING,
  VersionId STRING,
  HostId STRING,
  SigV STRING,
  CipherSuite STRING,
  AuthType STRING,
  HostHeader STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = '1', 'input.regex' = '([^\ ]*) ([^\ ]*)
\\[(.?.?)\\] ([^\ ]*) ([^\ ]*) ([^\ ]*) ([^\ ]*) ([^\ ]*)
\\\"([^\ ]*) ([^\ ]*) (- |([^\ ]*)\\\" (-|[0-9]*) ([^\ ]*) ([^\ ]*) ([^\ ]*) ([^\ ]*)
([^\ ]*)
([^\ ]*) (\\\"[^\"]*\") ([^\ ]*) ([^\ ]*) ([^\ ]*) ([^\ ]*) ([^\ ]*) ([^\ ]*).*$' )
LOCATION 's3://awsexamplebucket-logs/prefix'
```

4. In the navigation pane, under **Database**, choose your database.
5. Under **Tables**, choose **Preview table** next to your table name.

In the **Results** pane, you should see data from the server access logs, such as bucketowner, bucket, requestdatetime, and so on. This means that you successfully created the Athena table. You can now query the Amazon S3 server access logs.

Example — Show who deleted an object and when (timestamp, IP address, and IAM user)

```
SELECT RequestDateTime, RemoteIP, Requester, Key
FROM s3_access_logs_db.mybucket_logs
WHERE key = 'images/picture.jpg' AND operation like '%DELETE%';
```

Example — Show all operations executed by an IAM user

```
SELECT *
FROM s3_access_logs_db.mybucket_logs
WHERE requester='arn:aws:iam::123456789123:user/user_name';
```

Example — Show all operations that were performed on an object in a specific time period

```
SELECT *
FROM s3_access_logs_db.mybucket_logs
WHERE Key='prefix/images/picture.jpg'
      AND parse_datetime(RequestDateTime, 'dd/MMM/yyyy:HH:mm:ss Z')
      BETWEEN parse_datetime('2017-02-18:07:00:00', 'yyyy-MM-dd:HH:mm:ss')
      AND parse_datetime('2017-02-18:08:00:00', 'yyyy-MM-dd:HH:mm:ss');
```

Example — Show how much data was transferred by a specific IP address in a specific time period

```
SELECT SUM(bytesent) AS uploadTotal,
       SUM(objectsize) AS downloadTotal,
       SUM(bytesent + objectsize) AS Total
FROM s3_access_logs_db.mybucket_logs
WHERE RemoteIP='1.2.3.4'
      AND parse_datetime(RequestDateTime, 'dd/MMM/yyyy:HH:mm:ss Z')
      BETWEEN parse_datetime('2017-06-01', 'yyyy-MM-dd')
      AND parse_datetime('2017-07-01', 'yyyy-MM-dd');
```

Note

To reduce the time that you retain your log, you can [create an Amazon S3 lifecycle policy](#) for your server access logs bucket. Configure the lifecycle policy to remove log files periodically. Doing so reduces the amount of data that Athena analyzes for each query.

Using Amazon S3 Access Logs to Identify Signature Version 2 Requests

Amazon S3 support for Signature Version 2 will be turned off (deprecated). After that, Amazon S3 will no longer accept requests that use Signature Version 2, and all requests must use *Signature Version 4* signing. You can identify Signature Version 2 access requests using Amazon S3 access logs.

Note

- We recommend that you use AWS CloudTrail data events instead of Amazon S3 access logs. CloudTrail data events are easier to set up and contain more information. For more information, see [Using AWS CloudTrail to Identify Amazon S3 Signature Version 2 Requests](#) (p. 631).

Example — Show all requesters that are sending Signature Version 2 traffic

```
SELECT requester, Sigv, Count(Sigv) as SigCount
FROM s3_access_logs_db.mybucket_logs
GROUP BY requester, Sigv;
```

Using Amazon S3 Access Logs to Identify Object Access Requests

You can use queries on Amazon S3 server access logs to identify Amazon S3 object access requests, for operations such as GET, PUT, and DELETE, and discover further information about those requests.

The following Amazon Athena query example shows how to get all PUT object requests for Amazon S3 from the server access log.

Example — Show all requesters that are sending PUT object requests in a certain period

```
SELECT Bucket, Requester, RemoteIP, Key, HTTPStatus, ErrorCode, RequestDateTime
FROM s3_access_logs_db
WHERE Operation='REST.PUT.OBJECT' AND
parse_datetime(RequestDateTime,'dd/MMM/yyyy:HH:mm:ss Z')
BETWEEN parse_datetime('2019-07-01:00:42:42','yyyy-MM-dd:HH:mm:ss')
AND
parse_datetime('2019-07-02:00:42:42','yyyy-MM-dd:HH:mm:ss')
```

The following Amazon Athena query example shows how to get all GET object requests for Amazon S3 from the server access log.

Example — Show all requesters that are sending GET object requests in a certain period

```
SELECT Bucket, Requester, RemoteIP, Key, HTTPStatus, ErrorCode, RequestDateTime
FROM s3_access_logs_db
WHERE Operation='REST.GET.OBJECT' AND
parse_datetime(RequestDateTime,'dd/MMM/yyyy:HH:mm:ss Z')
BETWEEN parse_datetime('2019-07-01:00:42:42','yyyy-MM-dd:HH:mm:ss')
AND
parse_datetime('2019-07-02:00:42:42','yyyy-MM-dd:HH:mm:ss')
```

The following Amazon Athena query example shows how to get all anonymous requests to your S3 buckets from the server access log.

Example — Show all anonymous requesters that are making requests to a bucket in a certain period

```
SELECT Bucket, Requester, RemoteIP, Key, HTTPStatus, ErrorCode, RequestDateTime
FROM s3_access_logs_db.mybucket_logs
WHERE Requester IS NULL AND
parse_datetime(RequestDateTime,'dd/MMM/yyyy:HH:mm:ss Z')
BETWEEN parse_datetime('2019-07-01:00:42:42','yyyy-MM-dd:HH:mm:ss')
AND
parse_datetime('2019-07-02:00:42:42','yyyy-MM-dd:HH:mm:ss')
```

Note

- You can modify the date range as needed to suit your needs.
- These query examples may also be useful for security monitoring. You can review the results for PutObject or GetObject calls from unexpected or unauthorized IP addresses/requesters and for identifying any anonymous requests to your buckets.
- This query only retrieves information from the time at which logging was enabled.

- If you are using Amazon S3 AWS CloudTrail logs, see [Using AWS CloudTrail to Identify Access to Amazon S3 Objects \(p. 633\)](#).

Related Resources

- [Amazon S3 Server Access Log Format \(p. 653\)](#)
- [Querying AWS Service Logs](#)

Using the AWS SDKs, CLI, and Explorers

You can use the AWS SDKs when developing applications with Amazon S3. The AWS SDKs simplify your programming tasks by wrapping the underlying REST API. The AWS Mobile SDKs and the AWS Amplify JavaScript library are also available for building connected mobile and web applications using AWS.

This section provides an overview of using AWS SDKs for developing Amazon S3 applications. This section also describes how you can test the AWS SDK code examples provided in this guide.

Topics

- [Specifying the Signature Version in Request Authentication](#) (p. 670)
- [Setting Up the AWS CLI](#) (p. 675)
- [Using the AWS SDK for Java](#) (p. 676)
- [Using the AWS SDK for .NET](#) (p. 677)
- [Using the AWS SDK for PHP and Running PHP Examples](#) (p. 678)
- [Using the AWS SDK for Ruby - Version 3](#) (p. 679)
- [Using the AWS SDK for Python \(Boto\)](#) (p. 681)
- [Using the AWS Mobile SDKs for iOS and Android](#) (p. 681)
- [Using the AWS Amplify JavaScript Library](#) (p. 681)

In addition to the AWS SDKs, AWS Explorers are available for Visual Studio and Eclipse for Java IDE. In this case, the SDKs and the explorers are available bundled together as AWS Toolkits.

You can also use the AWS Command Line Interface (AWS CLI) to manage Amazon S3 buckets and objects.

AWS Toolkit for Eclipse

The AWS Toolkit for Eclipse includes both the AWS SDK for Java and AWS Explorer for Eclipse. The AWS Explorer for Eclipse is an open source plugin for Eclipse for Java IDE that makes it easier for developers to develop, debug, and deploy Java applications using AWS. The easy-to-use GUI enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as managing your buckets and objects and setting IAM policies, while developing applications, all from within the context of Eclipse for Java IDE. For set up instructions, see [Set up the Toolkit](#). For examples of using the explorer, see [How to Access AWS Explorer](#).

AWS Toolkit for Visual Studio

AWS Explorer for Visual Studio is an extension for Microsoft Visual Studio that makes it easier for developers to develop, debug, and deploy .NET applications using Amazon Web Services. The easy-to-use GUI enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as managing your buckets and objects or setting IAM policies, while developing applications, all from within the context of Visual Studio. For setup instructions, go to [Setting Up the AWS Toolkit for Visual Studio](#). For examples of using Amazon S3 using the explorer, see [Using Amazon S3 from AWS Explorer](#).

AWS SDKs

You can download only the SDKs. For information about downloading the SDK libraries, see [Sample Code Libraries](#).

AWS CLI

The AWS CLI is a unified tool to manage your AWS services, including Amazon S3. For information about downloading the AWS CLI, see [AWS Command Line Interface](#).

Specifying the Signature Version in Request Authentication

Amazon S3 supports only AWS Signature Version 4 in most AWS Regions. In some of the older AWS Regions, Amazon S3 supports both Signature Version 4 and Signature Version 2. However, Signature Version 2 is being turned off (deprecated). For more information about the end of support for Signature Version 2, see [AWS Signature Version 2 Turned Off \(Deprecated\) for Amazon S3 \(p. 671\)](#).

For a list of all the Amazon S3 Regions and the signature versions they support, see [Regions and Endpoints](#) in the *AWS General Reference*.

For all AWS Regions, AWS SDKs use Signature Version 4 by default to authenticate requests. When using AWS SDKs that were released before May 2016, you might be required to request Signature Version 4, as shown in the following table.

SDK	Requesting Signature Version 4 for Request Authentication
AWS CLI	For the default profile, run the following command:
	<pre>\$ aws configure set default.s3.signature_version s3v4</pre>
	For a custom profile, run the following command:
	<pre>\$ aws configure set profile.your_profile_name.s3.signature_version s3v4</pre>
Java SDK	Add the following in your code:
	<pre>System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY, "true");</pre>
	Or, on the command line, specify the following:
	<pre>-Dcom.amazonaws.services.s3.enableV4</pre>
JavaScript SDK	Set the signatureVersion parameter to v4 when constructing the client:
	<pre>var s3 = new AWS.S3({signatureVersion: 'v4'});</pre>
PHP SDK	Set the signature parameter to v4 when constructing the Amazon S3 service client for PHP SDK v2:
	<pre><?php \$client = S3Client::factory(['region' => 'YOUR-REGION', 'version' => 'latest',</pre>

SDK	Requesting Signature Version 4 for Request Authentication
	<pre>'signature' => 'v4'</pre> <pre>]);</pre> <p>When using the PHP SDK v3, set the <code>signature_version</code> parameter to <code>v4</code> during construction of the Amazon S3 service client:</p> <pre><?php \$s3 = new Aws\S3\S3Client(['version' => '2006-03-01', 'region' => 'YOUR-REGION', 'signature_version' => 'v4']);</pre>
Python-Boto SDK	<p>Specify the following in the boto default config file:</p> <pre>[s3] use-sigv4 = True</pre>
Ruby SDK	<p>Ruby SDK - Version 1: Set the <code>:s3_signature_version</code> parameter to <code>:v4</code> when constructing the client:</p> <pre>s3 = AWS::S3::Client.new(:s3_signature_version => :v4)</pre> <p>Ruby SDK - Version 3: Set the <code>signature_version</code> parameter to <code>v4</code> when constructing the client:</p> <pre>s3 = Aws::S3::Client.new(signature_version: 'v4')</pre>
.NET SDK	<p>Add the following to the code before creating the Amazon S3 client:</p> <pre>AWSConfigsS3.UseSignatureVersion4 = true;</pre> <p>Or, add the following to the config file:</p> <pre><appSettings> <add key="AWS.S3.UseSignatureVersion4" value="true" /> </appSettings></pre>

AWS Signature Version 2 Turned Off (Deprecated) for Amazon S3

Signature Version 2 is being turned off (deprecated) in Amazon S3. Amazon S3 will then only accept API requests that are signed using Signature Version 4.

This section provides answers to common questions regarding the end of support for Signature Version 2.

What is Signature Version 2/4, and What Does It Mean to Sign Requests?

The Signature Version 2 or Signature Version 4 signing process is used to authenticate your Amazon S3 API requests. Signing requests enables Amazon S3 to identify who is sending the request and protects your requests from bad actors.

For more information about signing AWS requests, see [Signing AWS API Requests](#) in the *AWS General Reference*.

What Update Are You Making?

We currently support Amazon S3 API requests that are signed using Signature Version 2 and Signature Version 4 processes. After that, Amazon S3 will only accept requests that are signed using Signature Version 4.

For more information about signing AWS requests, see [Changes in Signature Version 4](#) in the *AWS General Reference*.

Why Are You Making the Update?

Signature Version 4 provides improved security by using a signing key instead of your secret access key. Signature Version 4 is currently supported in all AWS Regions, whereas Signature Version 2 is only supported in Regions that were launched before January 2014. This update allows us to provide a more consistent experience across all Regions.

How Do I Ensure That I'm Using Signature Version 4, and What Updates Do I Need?

The signature version that is used to sign your requests is usually set by the tool or the SDK on the client side. By default, the latest versions of our AWS SDKs use Signature Version 4. For third-party software, contact the appropriate support team for your software to confirm what version you need. If you are sending direct REST calls to Amazon S3, you must modify your application to use the Signature Version 4 signing process.

For information about which version of the AWS SDKs to use when moving to Signature Version 4, see [Moving from Signature Version 2 to Signature Version 4](#) (p. 673).

For information about using Signature Version 4 with the Amazon S3 REST API, see [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

What Happens if I Don't Make Updates?

Requests signed with Signature Version 2 that are made after that will fail to authenticate with Amazon S3. Requesters will see errors stating that the request must be signed with Signature Version 4.

Should I Make Changes Even if I'm Using a Presigned URL That Requires Me to Sign for More than 7 Days?

If you are using a presigned URL that requires you to sign for more than 7 days, no action is currently needed. You can continue to use AWS Signature Version 2 to sign and authenticate the presigned URL. We will follow up and provide more details on how to migrate to Signature Version 4 for a presigned URL scenario.

More Info

- For more information about using Signature Version 4, see [Signing AWS API Requests](#).
- View the list of changes between Signature Version 2 and Signature Version 4 in [Changes in Signature Version 4](#).
- View the post [AWS Signature Version 4 to replace AWS Signature Version 2 for signing Amazon S3 API requests](#) in the AWS forums.

- If you have any questions or concerns, contact [AWS Support](#).

Moving from Signature Version 2 to Signature Version 4

If you currently use Signature Version 2 for Amazon S3 API request authentication, you should move to using Signature Version 4. Support is ending for Signature Version 2, as described in [AWS Signature Version 2 Turned Off \(Deprecated\) for Amazon S3 \(p. 671\)](#).

For information about using Signature Version 4 with the Amazon S3 REST API, see [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

The following table lists the SDKs with the necessary minimum version to use Signature Version 4 (SigV4).

If you are using presigned URLs with the AWS Java, JavaScript (Node.js), or Python (Boto/CLI) SDKs, you must set the correct AWS Region and set Signature Version 4 in the client configuration. For information about setting sigv4 in the client configuration, see [Specifying the Signature Version in Request Authentication \(p. 670\)](#).

If you use this SDK/Product	Upgrade to this SDK version	Code change needed to the client to use Sigv4?	Link to SDK documentation
AWS SDK for Java v1	Upgrade to Java 1.11.x or v2 in Q4 2018.	Yes	Specifying the Signature Version in Request Authentication (p. 670)
AWS SDK for Java v2 (preview)	No SDK upgrade is needed.	No	AWS SDK for Java
AWS SDK for .NET v1	Upgrade to 3.1.10 or later.	Yes	AWS SDK for .NET
AWS SDK for .NET v2	Upgrade to 3.1.10 or later.	No	AWS SDK for .NET v2
AWS SDK for .NET v3	No SDK upgrade is needed.	Yes	AWS SDK for .NET v3
AWS SDK for JavaScript v1	Upgrade to 2.68.0 or later.	Yes	AWS SDK for JavaScript
AWS SDK for JavaScript v2	Upgrade to 2.68.0 or later.	Yes	AWS SDK for JavaScript
AWS SDK for JavaScript v3	No action is currently needed. Upgrade to major version V3 in Q3 2019.	No	AWS SDK for JavaScript

If you use this SDK/Product	Upgrade to this SDK version	Code change needed to the client to use Sigv4?	Link to SDK documentation
AWS SDK for PHP v1	Recommend to upgrade to the most recent version of PHP or, at least to v2.7.4 with the signature parameter set to v4 in the S3 client's configuration.	Yes	AWS SDK for PHP
AWS SDK for PHP v2	Recommend to upgrade to the most recent version of PHP or, at least to v2.7.4 with the signature parameter set to v4 in the S3 client's configuration.	No	AWS SDK for PHP
AWS SDK for PHP v3	No SDK upgrade is needed.	No	AWS SDK for PHP
Boto2	Upgrade to Boto2 v2.49.0.	Yes	Boto 2 Upgrade
Boto3	Upgrade to 1.5.71 (Botocore), 1.4.6 (Boto3).	Yes	Boto 3 - AWS SDK for Python
AWS CLI	Upgrade to 1.11.108.	Yes	AWS Command Line Interface
AWS CLI v2 (preview)	No SDK upgrade is needed.	No	AWS Command Line Interface version 2
AWS SDK for Ruby v1	Upgrade to Ruby V3.	Yes	Ruby V3 for AWS
AWS SDK for Ruby v2	Upgrade to Ruby V3.	Yes	Ruby V3 for AWS
AWS SDK for Ruby v3	No SDK upgrade is needed.	No	Ruby V3 for AWS

If you use this SDK/Product	Upgrade to this SDK version	Code change needed to the client to use Sigv4?	Link to SDK documentation
Go	No SDK upgrade is needed.	No	AWS SDK for Go
C++	No SDK upgrade is needed.	No	AWS SDK for C++

AWS Tools for Windows PowerShell or AWS Tools for PowerShell Core

If you are using module versions *earlier* than 3.3.99, you must upgrade to 3.3.99.

To get the version information, use the `Get-Module` cmdlet:

```
Get-Module -Name AWSPowerShell
Get-Module -Name AWSPowerShell.NetCore
```

To update the 3.3.99 version, use the `Update-Module` cmdlet:

```
Update-Module -Name AWSPowerShell
Update-Module -Name AWSPowerShell.NetCore
```

You can use presigned URLs that are valid for more than 7 days that you will send Signature Version 2 traffic on.

Setting Up the AWS CLI

Follow the steps to download and configure AWS Command Line Interface (AWS CLI).

Note

Services in AWS, such as Amazon S3, require that you provide credentials when you access them. The service can then determine whether you have permissions to access the resources that it owns. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API. However, we don't recommend that you access AWS using the credentials for your AWS account. Instead, we recommend that you use AWS Identity and Access Management (IAM). Create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user that you created. You can then access AWS using a special URL and that IAM user's credentials. For instructions, go to [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:
 - [Getting Set Up with the AWS Command Line Interface](#)

- [Configuring the AWS Command Line Interface](#)
2. Add a named profile for the administrator user in the AWS CLI config file. You use this profile when executing the AWS CLI commands.

```
[adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

For a list of available AWS Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.

3. Verify the setup by typing the following commands at the command prompt.

- Try the `help` command to verify that the AWS CLI is installed on your computer:

```
aws help
```

- Try an S3 command to verify that the user can reach Amazon S3. This command lists buckets in your account. The AWS CLI uses the `adminuser` credentials to authenticate the request.

```
aws s3 ls --profile adminuser
```

Using the AWS SDK for Java

The AWS SDK for Java provides an API for the Amazon S3 bucket and object operations. For object operations, in addition to providing the API to upload objects in a single operation, the SDK provides an API to upload large objects in parts. For more information, see [Uploading Objects Using Multipart Upload API](#) (p. 175).

Topics

- [The Java API Organization](#) (p. 677)
- [Testing the Amazon S3 Java Code Examples](#) (p. 677)

The AWS SDK for Java gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, such as create, update, and delete operations that apply to buckets and objects. When you upload large objects using the low-level multipart upload API, it provides greater control. For example, it lets you pause and resume multipart uploads, vary part sizes during the upload, or begin uploads when you don't know the size of the data in advance. If you don't have these requirements, use the high-level API to upload objects.

High-Level API

For uploading objects, the SDK provides a higher level of abstraction by providing the `TransferManager` class. The high-level API is a simpler API, where in just a few lines of code you can upload files and streams to Amazon S3. You should use this API to upload data unless you need to control the upload as described in the preceding Low-Level API section.

For smaller data size, the `TransferManager` API uploads data in a single operation. However, the `TransferManager` switches to using the multipart upload API when the data size reaches a certain threshold. When possible, the `TransferManager` uses multiple threads to concurrently upload the parts. If a part upload fails, the API retries the failed part upload up to three times. However, these are configurable options using the `TransferManagerConfiguration` class.

Note

When you're using a stream for the source of data, the `TransferManager` class does not do concurrent uploads.

The Java API Organization

The following packages in the AWS SDK for Java provide the API:

- **com.amazonaws.services.s3**—Provides the APIs for creating Amazon S3 clients and working with buckets and objects. For example, it enables you to create buckets, upload objects, get objects, delete objects, and list keys.
- **com.amazonaws.services.s3.transfer**—Provides the high-level API data operations.

This high-level API is designed to simplify transferring objects to and from Amazon S3. It includes the `TransferManager` class, which provides asynchronous methods for working with, querying, and manipulating transfers. It also includes the `TransferManagerConfiguration` class, which you can use to configure the minimum part size for uploading parts and the threshold in bytes of when to use multipart uploads.

- **com.amazonaws.services.s3.model**—Provides the low-level API classes to create requests and process responses. For example, it includes the `GetObjectRequest` class to describe your get object request, the `ListObjectsRequest` class to describe your list keys requests, and the `InitiateMultipartUploadRequest` class to create multipart uploads.

For more information about the AWS SDK for Java API, see [AWS SDK for Java API Reference](#).

Testing the Amazon S3 Java Code Examples

The Java examples in this guide are compatible with the AWS SDK for Java version 1.11.321. For instructions on setting up and running code samples, see [Getting Started with the AWS SDK for Java](#) in the AWS SDK for Java Developer Guide.

Using the AWS SDK for .NET

The AWS SDK for .NET provides the API for the Amazon S3 bucket and object operations. For object operations, in addition to providing the API to upload objects in a single operation, the SDK provides the API to upload large objects in parts (see [Uploading Objects Using Multipart Upload API \(p. 175\)](#)).

Topics

- [The .NET API Organization \(p. 678\)](#)
- [Running the Amazon S3 .NET Code Examples \(p. 678\)](#)

The AWS SDK for .NET gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, including the create, update, and delete operations that apply to buckets and objects. When you upload large objects using the low-level multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 175\)](#)), it provides greater control. For example, it lets you pause and resume multipart uploads, vary part sizes during the upload, or begin uploads when you don't know the size of the data in advance. If you do not have these requirements, use the high-level API for uploading objects.

High-Level API

For uploading objects, the SDK provides a higher level of abstraction by providing the `TransferUtility` class. The high-level API is a simpler API, where in just a few lines of code, you can upload files and streams to Amazon S3. You should use this API to upload data unless you need to control the upload as described in the preceding Low-Level API section.

For smaller data size, the `TransferUtility` API uploads data in a single operation. However, the `TransferUtility` switches to using the multipart upload API when the data size reaches a certain threshold. By default, it uses multiple threads to concurrently upload the parts. If a part upload fails, the API retries the failed part upload up to three times. However, these are configurable options.

Note

When you're using a stream for the source of data, the `TransferUtility` class does not do concurrent uploads.

The .NET API Organization

When writing Amazon S3 applications using the AWS SDK for .NET, you use the `AWSSDK.dll`. The following namespaces in this assembly provide the multipart upload API:

- **Amazon.S3.Transfer**—Provides the high-level API to upload your data in parts.

It includes the `TransferUtility` class that enables you to specify a file, directory, or stream for uploading your data. It also includes the `TransferUtilityUploadRequest` and `TransferUtilityUploadDirectoryRequest` classes to configure advanced settings, such as the number of concurrent threads, part size, object metadata, the storage class (STANDARD, REDUCED_REDUNDANCY), and object access control list (ACL).

- **Amazon.S3**—Provides the implementation for the low-level APIs.

It provides methods that correspond to the Amazon S3 REST multipart upload API (see [Using the REST API for Multipart Upload \(p. 206\)](#)).

- **Amazon.S3.Model**—Provides the low-level API classes to create requests and process responses. For example, it provides the `InitiateMultipartUploadRequest` and `InitiateMultipartUploadResponse` classes you can use when initiating a multipart upload, and the `UploadPartRequest` and `UploadPartResponse` classes when uploading parts.
- **Amazon.S3.Encryption**— Provides `AmazonS3EncryptionClient`.
- **Amazon.S3.Util**— Provides various utility classes such as `AmazonS3Util` and `BucketRegionDetector`.

For more information about the AWS SDK for .NET API, see [AWS SDK for .NET Version 3 API Reference](#).

Running the Amazon S3 .NET Code Examples

The .NET code examples in this guide are compatible with the AWS SDK for .NET version 3.0. For information about setting up and running the code examples, see [Getting Started with the AWS SDK for .NET](#) in the *AWS SDK for .NET Developer Guide*.

Using the AWS SDK for PHP and Running PHP Examples

The AWS SDK for PHP provides access to the API for Amazon S3 bucket and object operations. The SDK gives you the option of using the service's low-level API or using higher-level abstractions.

The SDK is available at [AWS SDK for PHP](#), which also has instructions for installing and getting started with the SDK.

The setup for using the AWS SDK for PHP depends on your environment and how you want to run your application. To set up your environment to run the examples in this documentation, see the [AWS SDK for PHP Getting Started Guide](#).

Topics

- [AWS SDK for PHP Levels \(p. 679\)](#)
- [Running PHP Examples \(p. 679\)](#)
- [Related Resources \(p. 679\)](#)

AWS SDK for PHP Levels

The AWS SDK for PHP gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, including the create, update, and delete operations on buckets and objects. The low-level APIs provide greater control over these operations. For example, you can batch your requests and execute them in parallel. Or, when using the multipart upload API, you can manage the object parts individually. Note that these low-level API calls return a result that includes all of the Amazon S3 response details. For more information about the multipart upload API, see [Uploading Objects Using Multipart Upload API \(p. 175\)](#).

High-Level Abstractions

The high-level abstractions are intended to simplify common use cases. For example, for uploading large objects using the low-level API, you call `Aws\S3\S3Client::createMultipartUpload()`, call the `Aws\S3\S3Client::uploadPart()` method to upload the object parts, then call the `Aws\S3\S3Client::completeMultipartUpload()` method to complete the upload. You can use the higher-level `Aws\S3\MultipartUploader` object that simplifies creating a multipart upload instead.

As another example, when enumerating objects in a bucket, you can use the iterators feature of the AWS SDK for PHP to return all of the object keys, regardless of how many objects you have stored in the bucket. If you use the low-level API, the response returns a maximum of 1,000 keys. If a bucket contains more than 1,000 objects, the result is truncated and you have to manage the response and check for truncation.

Running PHP Examples

To set up and use the Amazon S3 samples for version 3 of the AWS SDK for PHP, see [Installation](#) in the AWS SDK for PHP Developer Guide.

Related Resources

- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)
- [AWS SDK for PHP API for Amazon S3](#)

Using the AWS SDK for Ruby - Version 3

The AWS SDK for Ruby provides an API for Amazon S3 bucket and object operations. For object operations, you can use the API to upload objects in a single operation or upload large objects in parts

(see [Using the AWS SDK for Ruby for Multipart Upload \(p. 205\)](#)). However, the API for a single operation upload can also accept large objects and behind the scenes manage the upload in parts for you, thereby reducing the amount of script you need to write.

The Ruby API Organization

When creating Amazon S3 applications using the AWS SDK for Ruby, you must install the SDK for Ruby gem. For more information, see the [AWS SDK for Ruby - Version 3](#). Once installed, you can access the API, including the following key classes:

- **Aws::S3::Resource**—Represents the interface to Amazon S3 for the Ruby SDK and provides methods for creating and enumerating buckets.

The `S3` class provides the `#buckets` instance method for accessing existing buckets or creating new ones.

- **Aws::S3::Bucket**—Represents an Amazon S3 bucket.

The `Bucket` class provides the `#object(key)` and `#objects` methods for accessing the objects in a bucket, as well as methods to delete a bucket and return information about a bucket, like the bucket policy.

- **Aws::S3::Object**—Represents an Amazon S3 object identified by its key.

The `Object` class provides methods for getting and setting properties of an object, specifying the storage class for storing objects, and setting object permissions using access control lists. The `Object` class also has methods for deleting, uploading and copying objects. When uploading objects in parts, this class provides options for you to specify the order of parts uploaded and the part size.

For more information about the AWS SDK for Ruby API, go to [AWS SDK for Ruby - Version 2](#).

Testing the Ruby Script Examples

The easiest way to get started with the Ruby script examples is to install the latest AWS SDK for Ruby gem. For information about installing or updating to the latest gem, go to [AWS SDK for Ruby - Version 3](#). The following tasks guide you through the creation and testing of the Ruby script examples assuming that you have installed the AWS SDK for Ruby.

General Process of Creating and Testing Ruby Script Examples

1	To access AWS, you must provide a set of credentials for your SDK for Ruby application. For more information, see Configuring the AWS SDK for Ruby .
2	<div>Create a new SDK for Ruby script and add the following lines to the top of the script.</div> <div><pre>#!/usr/bin/env ruby require 'rubygems' require 'aws-sdk-s3'</pre></div> <div>The first line is the interpreter directive and the two <code>require</code> statements import two required gems into your script.</div>
3	Copy the code from the section you are reading to your script.
4	Update the code by providing any required data. For example, if uploading a file, provide the file path and the bucket name.

5	Run the script. Verify changes to buckets and objects by using the AWS Management Console. For more information about the AWS Management Console, go to https://aws.amazon.com/console/ .
---	---

Ruby Samples

The following links contain samples to help get you started with the SDK for Ruby - Version 3:

- [Using the AWS SDK for Ruby Version 3 \(p. 62\)](#)
- [Upload an Object Using the AWS SDK for Ruby \(p. 173\)](#)

Using the AWS SDK for Python (Boto)

Boto is a Python package that provides interfaces to AWS including Amazon S3. For more information about Boto, go to the [AWS SDK for Python \(Boto\)](#). The getting started link on this page provides step-by-step instructions to get started.

Using the AWS Mobile SDKs for iOS and Android

You can use the AWS Mobile SDKs for Android and iOS, together with [AWS Mobile Hub](#), to quickly and easily integrate robust cloud backends into your existing mobile apps. You can configure and use features like user sign-in, databases, push notifications, and more, without being an AWS expert.

The AWS Mobile SDKs provide easy access to Amazon S3 and many other AWS services. To get started using the AWS Mobile SDKs, see [Getting Started with the AWS Mobile SDKs](#).

More Info

[Using the AWS Amplify JavaScript Library \(p. 681\)](#)

Using the AWS Amplify JavaScript Library

AWS Amplify is an open source JavaScript library for web and mobile developers who build cloud-enabled applications. AWS Amplify provides customizable UI components and a declarative interface to work with an S3 bucket, along with other high-level categories for AWS services.

To get started using the AWS Amplify JavaScript library, choose one of the following links:

- [Getting Started with the AWS Amplify Library for the Web](#)
- [Getting Started with the AWS Amplify Library for React Native](#)

For more information about AWS Amplify, see [AWS Amplify](#) on [GitHub](#).

More Info

[Using the AWS Mobile SDKs for iOS and Android \(p. 681\)](#)

Appendices

This Amazon Simple Storage Service Developer Guide appendix include the following sections.

Topics

- [Appendix A: Using the SOAP API \(p. 682\)](#)
- [Appendix B: Authenticating Requests \(AWS Signature Version 2\) \(p. 684\)](#)

Appendix A: Using the SOAP API

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

This section contains information specific to the Amazon S3 SOAP API.

Note

SOAP requests, both authenticated and anonymous, must be sent to Amazon S3 using SSL. Amazon S3 returns an error when you send a SOAP request over HTTP.

Common SOAP API Elements

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

You can interact with Amazon S3 using SOAP 1.1 over HTTP. The Amazon S3 WSDL, which describes the Amazon S3 API in a machine-readable way, is available at: <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>. The Amazon S3 schema is available at <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.xsd>.

Most users will interact with Amazon S3 using a SOAP toolkit tailored for their language and development environment. Different toolkits will expose the Amazon S3 API in different ways. Please refer to your specific toolkit documentation to understand how to use it. This section illustrates the Amazon S3 SOAP operations in a toolkit-independent way by exhibiting the XML requests and responses as they appear "on the wire."

Common Elements

You can include the following authorization-related elements with any SOAP request:

- `AWSSessionToken`: The AWS Access Key ID of the requester
- `Timestamp`: The current time on your system
- `Signature`: The signature for the request

Authenticating SOAP Requests

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Every non-anonymous request must contain authentication information to establish the identity of the principal making the request. In SOAP, the authentication information is put into the following elements of the SOAP request:

- Your AWS Access Key ID

Note

When making authenticated SOAP requests, temporary security credentials are not supported. For more information about types of credentials, see [Making Requests \(p. 10\)](#).

- **Timestamp:** This must be a dateTime (go to <http://www.w3.org/TR/xmlschema-2/#dateTime>) in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as 2009-01-01T12:00:00.000Z. Authorization will fail if this timestamp is more than 15 minutes away from the clock on Amazon S3 servers.
- **Signature:** The RFC 2104 HMAC-SHA1 digest (go to <http://www.ietf.org/rfc/rfc2104.txt>) of the concatenation of "AmazonS3" + OPERATION + Timestamp, using your AWS Secret Access Key as the key. For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2009-01-01T12:00:00.000Z":

For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2009-01-01T12:00:00.000Z":

Example

```
<CreateBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Acl>private</Acl>
  <AWSSecretAccessKey>AKIAIOSFODNN7EXAMPLE</AWSSecretAccessKey>
  <Timestamp>2009-01-01T12:00:00.000Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</CreateBucket>
```

Note

SOAP requests, both authenticated and anonymous, must be sent to Amazon S3 using SSL. Amazon S3 returns an error when you send a SOAP request over HTTP.

Important

Due to different interpretations regarding how extra time precision should be dropped, .NET users should take care not to send Amazon S3 overly specific time stamps. This can be accomplished by manually constructing DateTime objects with only millisecond precision.

Setting Access Policy with SOAP

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Access control can be set at the time a bucket or object is written by including the "AccessControlList" element with the request to CreateBucket, PutObjectInline, or PutObject. The AccessControlList element is described in [Identity and Access Management in Amazon S3 \(p. 301\)](#). If no access control list is specified with these operations, the resource is created with a default access policy that gives the requester FULL_CONTROL access (this is the case even if the request is a PutObjectInline or PutObject request for an object that already exists).

Following is a request that writes data to an object, makes the object readable by anonymous principals, and gives the specified user FULL_CONTROL rights to the bucket (Most developers will want to give themselves FULL_CONTROL access to their own bucket).

Example

Following is a request that writes data to an object and makes the object readable by anonymous principals.

Sample Request

```
<PutObjectInline xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <Metadata>
    <Name>Content-Type</Name>
    <Value>text/plain</Value>
  </Metadata>
  <Data>aGEtaGE=</Data>
  <ContentLength>5</ContentLength>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
        <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>
        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2009-03-01T12:00:00.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</PutObjectInline>
```

Sample Response

```
<PutObjectInlineResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <PutObjectInlineResponse>
    <ETag>"828ef3fdfa96f00ad9f27c383fc9ac7f"</ETag>
    <LastModified>2009-01-01T12:00:00.000Z</LastModified>
  </PutObjectInlineResponse>
</PutObjectInlineResponse>
```

The access control policy can be read or set for an existing bucket or object using the `GetBucketAccessControlPolicy`, `GetObjectAccessControlPolicy`, `SetBucketAccessControlPolicy`, and `SetObjectAccessControlPolicy` methods. For more information, see the detailed explanation of these methods.

Appendix B: Authenticating Requests (AWS Signature Version 2)

Important

This section describes how to authenticate requests using AWS Signature Version 2. Signature Version 2 is being turned off (deprecated), Amazon S3 will only accept API requests that are signed using Signature Version 4. For more information, see [AWS Signature Version 2 Turned Off \(Deprecated\) for Amazon S3 \(p. 671\)](#)

Signature Version 4 is supported in all AWS Regions, and it is the only version that is supported for new Regions. For more information, see [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

Amazon S3 offers you the ability to identify what API signature version was used to sign a request. It is important to identify if any of your workflows are utilizing Signature Version 2 signing and upgrading them to use Signature Version 4 to prevent impact to your business.

- If you are using CloudTrail event logs(recommended option), please see [Using AWS CloudTrail to Identify Amazon S3 Signature Version 2 Requests](#) (p. 631) on how to query and identify such requests.
- If you are using the Amazon S3 Server Access logs, see [Using Amazon S3 Access Logs to Identify Signature Version 2 Requests](#) (p. 666)

Topics

- [Authenticating Requests Using the REST API](#) (p. 686)
- [Signing and Authenticating REST Requests](#) (p. 688)
- [Browser-Based Uploads Using POST \(AWS Signature Version 2\)](#) (p. 697)

Authenticating Requests Using the REST API

When accessing Amazon S3 using REST, you must provide the following items in your request so the request can be authenticated:

Request Elements

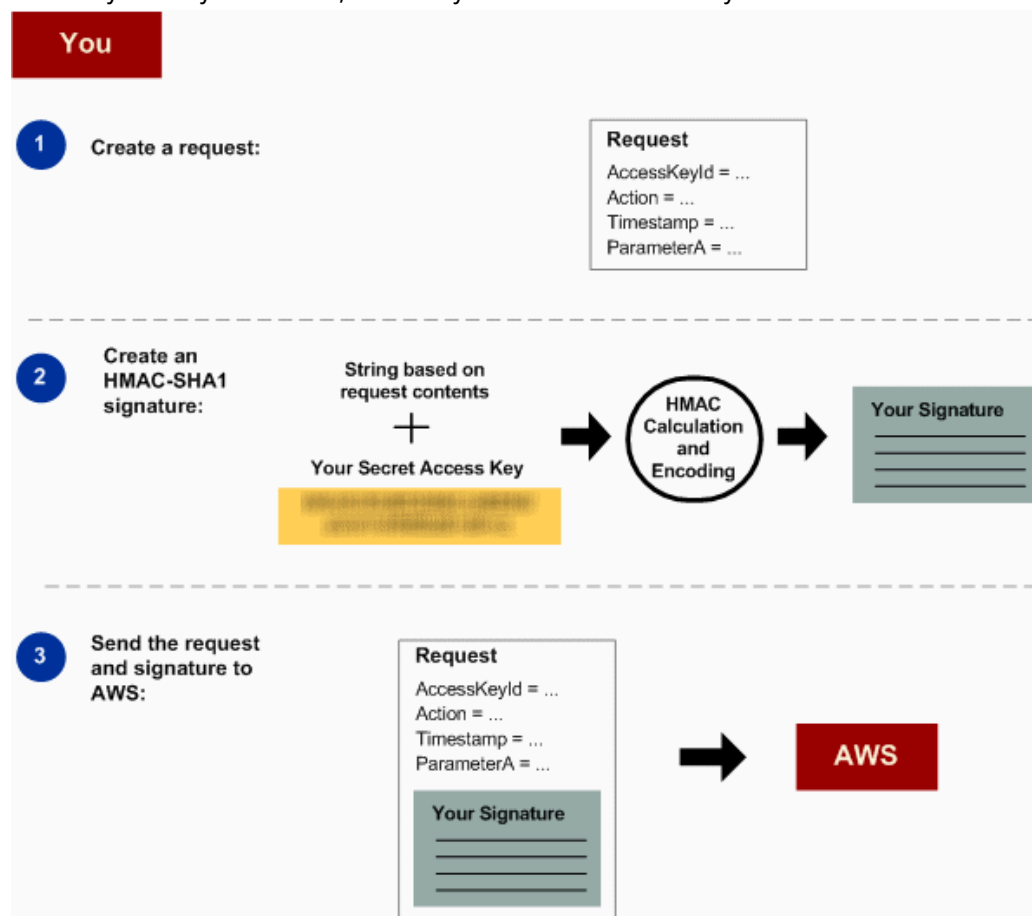
- **AWS Access Key Id** – Each request must contain the access key ID of the identity you are using to send your request.
- **Signature** – Each request must contain a valid request signature, or the request is rejected.

A request signature is calculated using your secret access key, which is a shared secret known only to you and AWS.

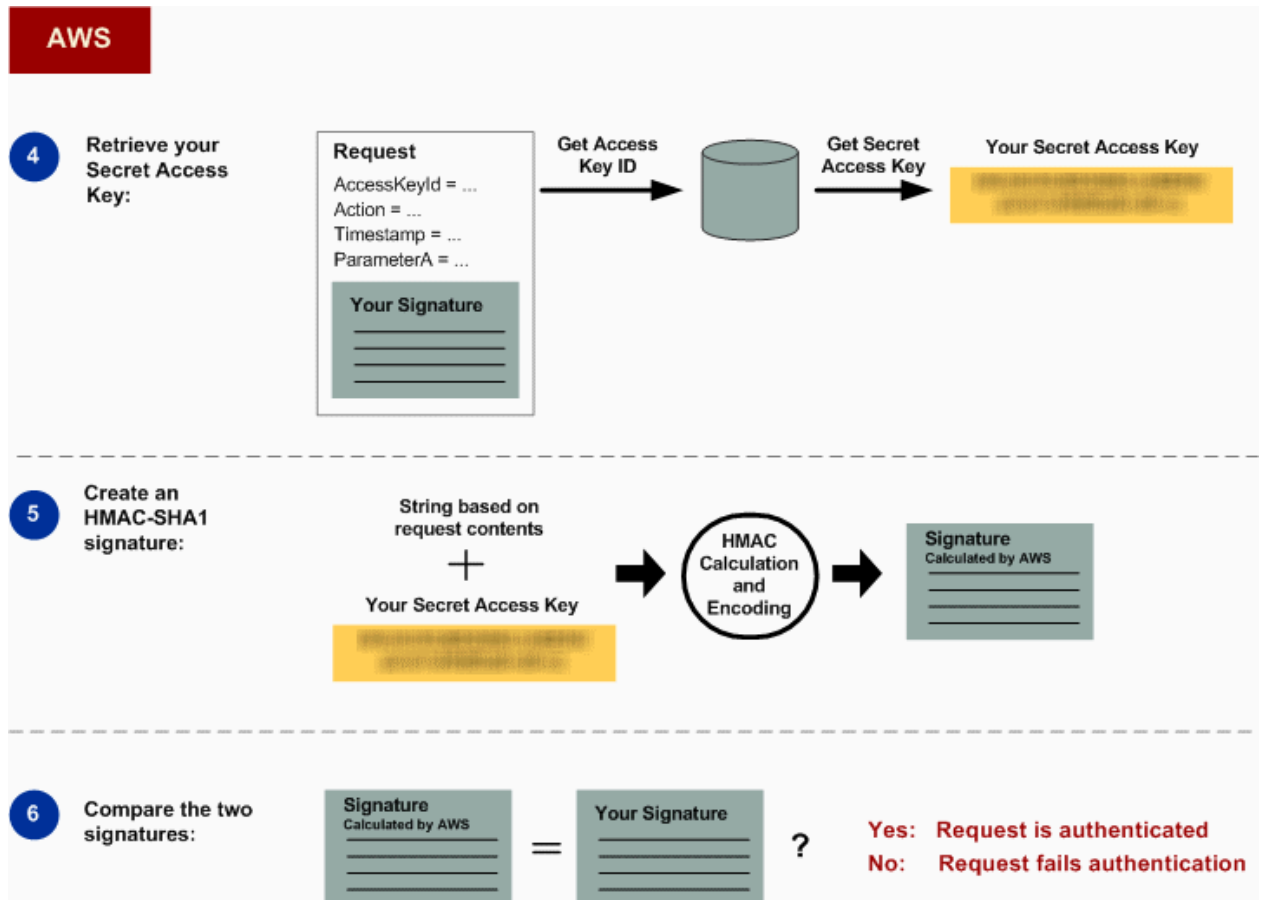
- **Time stamp** – Each request must contain the date and time the request was created, represented as a string in UTC.
- **Date** – Each request must contain the time stamp of the request.

Depending on the API action you're using, you can provide an expiration date and time for the request instead of or in addition to the time stamp. See the authentication topic for the particular action to determine what it requires.

Following are the general steps for authenticating requests to Amazon S3. It is assumed you have the necessary security credentials, access key ID and secret access key.



1	Construct a request to AWS.
2	Calculate the signature using your secret access key.
3	Send the request to Amazon S3. Include your access key ID and the signature in your request. Amazon S3 performs the next three steps.



4	Amazon S3 uses the access key ID to look up your secret access key.
5	Amazon S3 calculates a signature from the request data and the secret access key using the same algorithm that you used to calculate the signature you sent in the request.
6	If the signature generated by Amazon S3 matches the one you sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and Amazon S3 returns an error response.

Detailed Authentication Information

For detailed information about REST authentication, see [Signing and Authenticating REST Requests](#) (p. 688).

Signing and Authenticating REST Requests

Topics

- [Using Temporary Security Credentials](#) (p. 689)
- [The Authentication Header](#) (p. 689)
- [Request Canonicalization for Signing](#) (p. 690)
- [Constructing the CanonicalizedResource Element](#) (p. 690)
- [Constructing the CanonicalizedAmzHeaders Element](#) (p. 691)
- [Positional versus Named HTTP Header StringToSign Elements](#) (p. 691)
- [Time Stamp Requirement](#) (p. 691)
- [Authentication Examples](#) (p. 692)
- [REST Request Signing Problems](#) (p. 695)
- [Query String Request Authentication Alternative](#) (p. 695)

Note

This topic explains authenticating requests using Signature Version 2. Amazon S3 now supports the latest Signature Version 4. This latest signature version is supported in all regions and any new regions after January 30, 2014 will support only Signature Version 4. For more information, go to [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

Authentication is the process of proving your identity to the system. Identity is an important factor in Amazon S3 access control decisions. Requests are allowed or denied in part based on the identity of the requester. For example, the right to create buckets is reserved for registered developers and (by default) the right to create objects in a bucket is reserved for the owner of the bucket in question. As a developer, you'll be making requests that invoke these privileges, so you'll need to prove your identity to the system by authenticating your requests. This section shows you how.

Note

The content in this section does not apply to HTTP POST. For more information, see [Browser-Based Uploads Using POST \(AWS Signature Version 2\)](#) (p. 697).

The Amazon S3 REST API uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. To authenticate a request, you first concatenate selected elements of the request to form a string. You then use your AWS secret access key to calculate the HMAC of that string. Informally, we call this process "signing the request," and we call the output of the HMAC algorithm the signature, because it simulates the security properties of a real signature. Finally, you add this signature as a parameter of the request by using the syntax described in this section.

When the system receives an authenticated request, it fetches the AWS secret access key that you claim to have and uses it in the same way to compute a signature for the message it received. It then compares the signature it calculated against the signature presented by the requester. If the two signatures match, the system concludes that the requester must have access to the AWS secret access key and therefore acts with the authority of the principal to whom the key was issued. If the two signatures do not match, the request is dropped and the system responds with an error message.

Example Authenticated Amazon S3 REST Request

```
GET /photos/puppy.jpg HTTP/1.1
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000
```

```
Authorization: AWS AKIAIOSFODNN7EXAMPLE:frJIUN8DYpKDtOLCwo//yllqDzg=
```

Using Temporary Security Credentials

If you are signing your request using temporary security credentials (see [Making Requests \(p. 10\)](#)), you must include the corresponding security token in your request by adding the `x-amz-security-token` header.

When you obtain temporary security credentials using the AWS Security Token Service API, the response includes temporary security credentials and a session token. You provide the session token value in the `x-amz-security-token` header when you send requests to Amazon S3. For information about the AWS Security Token Service API provided by IAM, go to [Action](#) in the *AWS Security Token Service API Reference Guide*.

The Authentication Header

The Amazon S3 REST API uses the standard HTTP `Authorization` header to pass authentication information. (The name of the standard header is unfortunate because it carries authentication information, not authorization.) Under the Amazon S3 authentication scheme, the `Authorization` header has the following form:

```
Authorization: AWS AWSAccessKeyId:Signature
```

Developers are issued an AWS access key ID and AWS secret access key when they register. For request authentication, the `AWSAccessKeyId` element identifies the access key ID that was used to compute the signature and, indirectly, the developer making the request.

The `Signature` element is the RFC 2104 HMAC-SHA1 of selected elements from the request, and so the `Signature` part of the `Authorization` header will vary from request to request. If the request signature calculated by the system matches the `Signature` included with the request, the requester will have demonstrated possession of the AWS secret access key. The request will then be processed under the identity, and with the authority, of the developer to whom the key was issued.

Following is pseudogrammar that illustrates the construction of the `Authorization` request header. (In the example, `\n` means the Unicode code point U+000A, commonly called newline).

```
Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;

Signature = Base64( HMAC-SHA1( YourSecretAccessKey, UTF-8-Encoding-Of( StringToSign ) ) );

StringToSign = HTTP-Verb + "\n" +
  Content-MD5 + "\n" +
  Content-Type + "\n" +
  Date + "\n" +
  CanonicalizedAmzHeaders +
  CanonicalizedResource;

CanonicalizedResource = [ "/" + Bucket ] +
  <HTTP-Request-URI, from the protocol name up to the query string> +
  [ subresource, if present. For example "?acl", "?location", "?logging", or "?torrent" ];

CanonicalizedAmzHeaders = <described below>
```

HMAC-SHA1 is an algorithm defined by [RFC 2104 - Keyed-Hashing for Message Authentication](#). The algorithm takes as input two byte-strings, a key and a message. For Amazon S3 request authentication,

use your AWS secret access key (`YourSecretAccessKey`) as the key, and the UTF-8 encoding of the `StringToSign` as the message. The output of HMAC-SHA1 is also a byte string, called the digest. The `Signature` request parameter is constructed by Base64 encoding this digest.

Request Canonicalization for Signing

Recall that when the system receives an authenticated request, it compares the computed request signature with the signature provided in the request in `StringToSign`. For that reason, you must compute the signature by using the same method used by Amazon S3. We call the process of putting a request in an agreed-upon form for signing *canonicalization*.

Constructing the CanonicalizedResource Element

`CanonicalizedResource` represents the Amazon S3 resource targeted by the request. Construct it for a REST request as follows:

Launch Process

1	Start with an empty string ("").
2	<p>If the request specifies a bucket using the HTTP Host header (virtual hosted-style), append the bucket name preceded by a "/" (e.g., <code>"/bucketname"</code>). For path-style requests and requests that don't address a bucket, do nothing. For more information about virtual hosted-style requests, see Virtual Hosting of Buckets (p. 45).</p> <p>For a virtual hosted-style request <code>"https://johnsmith.s3.amazonaws.com/photos/puppy.jpg"</code>, the <code>CanonicalizedResource</code> is <code>"/johnsmith"</code>.</p> <p>For the path-style request, <code>"https://s3.amazonaws.com/johnsmith/photos/puppy.jpg"</code>, the <code>CanonicalizedResource</code> is <code>""</code>.</p>
3	<p>Append the path part of the un-decoded HTTP Request-URI, up-to but not including the query string.</p> <p>For a virtual hosted-style request <code>"https://johnsmith.s3.amazonaws.com/photos/puppy.jpg"</code>, the <code>CanonicalizedResource</code> is <code>"/johnsmith/photos/puppy.jpg"</code>.</p> <p>For a path-style request, <code>"https://s3.amazonaws.com/johnsmith/photos/puppy.jpg"</code>, the <code>CanonicalizedResource</code> is <code>"/johnsmith/photos/puppy.jpg"</code>. At this point, the <code>CanonicalizedResource</code> is the same for both the virtual hosted-style and path-style request.</p> <p>For a request that does not address a bucket, such as GET Service, append <code>"/"</code>.</p>
4	<p>If the request addresses a subresource, such as <code>?versioning</code>, <code>?location</code>, <code>?acl</code>, <code>?torrent</code>, <code>?lifecycle</code>, or <code>?versionid</code>, append the subresource, its value if it has one, and the question mark. Note that in case of multiple subresources, subresources must be lexicographically sorted by subresource name and separated by '&', e.g., <code>?acl&versionId=value</code>.</p> <p>The subresources that must be included when constructing the <code>CanonicalizedResource</code> Element are <code>acl</code>, <code>lifecycle</code>, <code>location</code>, <code>logging</code>, <code>notification</code>, <code>partNumber</code>, <code>policy</code>, <code>requestPayment</code>, <code>torrent</code>, <code>uploadId</code>, <code>uploads</code>, <code>versionId</code>, <code>versioning</code>, <code>versions</code>, and <code>website</code>.</p> <p>If the request specifies query string parameters overriding the response header values (see Get Object), append the query string parameters and their values. When signing, you do not encode these values; however, when making the request, you must encode these parameter values. The query string parameters in a GET request include <code>response-content-type</code>, <code>response-content-language</code>, <code>response-expires</code>, <code>response-cache-control</code>, <code>response-content-disposition</code>, and <code>response-content-encoding</code>.</p>

The delete query string parameter must be included when you create the CanonicalizedResource for a multi-object Delete request.

Elements of the CanonicalizedResource that come from the HTTP Request-URI should be signed literally as they appear in the HTTP request, including URL-Encoding meta characters.

The CanonicalizedResource might be different than the HTTP Request-URI. In particular, if your request uses the HTTP Host header to specify a bucket, the bucket does not appear in the HTTP Request-URI. However, the CanonicalizedResource continues to include the bucket. Query string parameters might also appear in the Request-URI but are not included in CanonicalizedResource. For more information, see [Virtual Hosting of Buckets \(p. 45\)](#).

Constructing the CanonicalizedAmzHeaders Element

To construct the CanonicalizedAmzHeaders part of StringToSign, select all HTTP request headers that start with 'x-amz-' (using a case-insensitive comparison), and use the following process.

CanonicalizedAmzHeaders Process

1	Convert each HTTP header name to lowercase. For example, 'X-Amz-Date' becomes 'x-amz-date'.
2	Sort the collection of headers lexicographically by header name.
3	Combine header fields with the same name into one "header-name:comma-separated-value-list" pair as prescribed by RFC 2616, section 4.2, without any whitespace between values. For example, the two metadata headers 'x-amz-meta-username: fred' and 'x-amz-meta-username: barney' would be combined into the single header 'x-amz-meta-username: fred,barney'.
4	"Unfold" long headers that span multiple lines (as allowed by RFC 2616, section 4.2) by replacing the folding whitespace (including new-line) by a single space.
5	Trim any whitespace around the colon in the header. For example, the header 'x-amz-meta-username: fred,barney' would become 'x-amz-meta-username:fred,barney'
6	Finally, append a newline character (U+000A) to each canonicalized header in the resulting list. Construct the CanonicalizedResource element by concatenating all headers in this list into a single string.

Positional versus Named HTTP Header StringToSign Elements

The first few header elements of StringToSign (Content-Type, Date, and Content-MD5) are positional in nature. StringToSign does not include the names of these headers, only their values from the request. In contrast, the 'x-amz-' elements are named. Both the header names and the header values appear in StringToSign.

If a positional header called for in the definition of StringToSign is not present in your request (for example, Content-Type or Content-MD5 are optional for PUT requests and meaningless for GET requests), substitute the empty string ("") for that position.

Time Stamp Requirement

A valid time stamp (using either the HTTP Date header or an x-amz-date alternative) is mandatory for authenticated requests. Furthermore, the client timestamp included with an authenticated request must be within 15 minutes of the Amazon S3 system time when the request is received. If not, the request will fail with the RequestTimeTooSkewed error code. The intention of these restrictions is to limit the

possibility that intercepted requests could be replayed by an adversary. For stronger protection against eavesdropping, use the HTTPS transport for authenticated requests.

Note

The validation constraint on request date applies only to authenticated requests that do not use query string authentication. For more information, see [Query String Request Authentication Alternative](#) (p. 695).

Some HTTP client libraries do not expose the ability to set the `Date` header for a request. If you have trouble including the value of the 'Date' header in the canonicalized headers, you can set the timestamp for the request by using an 'x-amz-date' header instead. The value of the `x-amz-date` header must be in one of the RFC 2616 formats (<http://www.ietf.org/rfc/rfc2616.txt>). When an `x-amz-date` header is present in a request, the system will ignore any `Date` header when computing the request signature. Therefore, if you include the `x-amz-date` header, use the empty string for the `Date` when constructing the `StringToSign`. See the next section for an example.

Authentication Examples

The examples in this section use the (non-working) credentials in the following table.

Parameter	Value
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE
AWSSecretAccessKey	wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY

In the example `StringToSigns`, formatting is not significant, and `\n` means the Unicode code point `U+000A`, commonly called newline. Also, the examples use "+0000" to designate the time zone. You can use "GMT" to designate timezone instead, but the signatures shown in the examples will be different.

Object GET

This example gets an object from the johnsmith bucket.

Request	StringToSign
<pre>GET /photos/puppy.jpg HTTP/1.1 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:36:42 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE: bWq2s1WEIj+Ydj0vQ697zp+IXMU=</pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:36:42 +0000\n /johnsmith/photos/puppy.jpg</pre>

Note that the `CanonicalizedResource` includes the bucket name, but the HTTP Request-URI does not. (The bucket is specified by the `Host` header.)

Object PUT

This example puts an object into the johnsmith bucket.

Request	StringToSign
<pre>PUT /photos/puppy.jpg HTTP/1.1</pre>	<pre>PUT\n</pre>

Request	StringToSign
Content-Type: image/jpeg Content-Length: 94328 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 21:15:45 +0000 <i>Authorization: AWS AKIAIOSFODNN7EXAMPLE: MyyxERY7whkBe+bq8fHCL/2kKUG=</i>	\n image/jpeg\n Tue, 27 Mar 2007 21:15:45 +0000\n /johnsmith/photos/puppy.jpg

Note the Content-Type header in the request and in the StringToSign. Also note that the Content-MD5 is left blank in the StringToSign, because it is not present in the request.

List

This example lists the content of the johnsmith bucket.

Request	StringToSign
GET /?prefix=photos&max-keys=50&marker=puppy HTTP/1.1 User-Agent: Mozilla/5.0 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:42:41 +0000 <i>Authorization: AWS AKIAIOSFODNN7EXAMPLE: htDYFYduRNeN8P9ZfE/s9SuKy0U=</i>	GET\n \n \n Tue, 27 Mar 2007 19:42:41 +0000\n /johnsmith/

Note the trailing slash on the CanonicalizedResource and the absence of query string parameters.

Fetch

This example fetches the access control policy subresource for the 'johnsmith' bucket.

Request	StringToSign
GET /?acl HTTP/1.1 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:44:46 +0000 <i>Authorization: AWS AKIAIOSFODNN7EXAMPLE: c2WLPFtWHVgbEmeEG93a4cG37dM=</i>	GET\n \n \n Tue, 27 Mar 2007 19:44:46 +0000\n /johnsmith/?acl

Notice how the subresource query string parameter is included in the CanonicalizedResource.

Delete

This example deletes an object from the 'johnsmith' bucket using the path-style and Date alternative.

Request	StringToSign
DELETE /johnsmith/photos/puppy.jpg HTTP/1.1 User-Agent: dotnet	DELETE\n \n

Request	StringToSign
Host: s3.amazonaws.com Date: Tue, 27 Mar 2007 21:20:27 +0000 x-amz-date: Tue, 27 Mar 2007 21:20:26 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE:lx3byBScXR6KzyMaiFNkardMwNk=	\n Tue, 27 Mar 2007 21:20:26 +0000\n /johnsmith/photos/puppy.jpg

Note how we used the alternate 'x-amz-date' method of specifying the date (because our client library prevented us from setting the date, say). In this case, the x-amz-date takes precedence over the Date header. Therefore, date entry in the signature must contain the value of the x-amz-date header.

Upload

This example uploads an object to a CNAME style virtual hosted bucket with metadata.

Request	StringToSign
PUT /db-backup.dat.gz HTTP/1.1 User-Agent: curl/7.15.5 Host: static.johnsmith.net:8080 Date: Tue, 27 Mar 2007 21:06:08 +0000 x-amz-acl: public-read content-type: application/x-download Content-MD5: 4gJE4saaMU4BqNR0kLY+lw== X-Amz-Meta-ReviewedBy: joe@johnsmith.net X-Amz-Meta-ReviewedBy: jane@johnsmith.net X-Amz-Meta-FileChecksum: 0x02661779 X-Amz-Meta-ChecksumAlgorithm: crc32 Content-Disposition: attachment; filename=database.dat Content-Encoding: gzip Content-Length: 5913339 Authorization: AWS AKIAIOSFODNN7EXAMPLE: ilyl83RwaSoYIEdixDQcA4OnAnc=	PUT\n 4gJE4saaMU4BqNR0kLY+lw==\n application/x-download\n Tue, 27 Mar 2007 21:06:08 +0000\n x-amz-acl:public-read\n x-amz-meta-checksumalgorithm:crc32\n x-amz-meta-filechecksum:0x02661779\n x-amz-meta-reviewedby: joe@johnsmith.net,jane@johnsmith.net\n /static.johnsmith.net/db-backup.dat.gz

Notice how the 'x-amz-' headers are sorted, trimmed of whitespace, and converted to lowercase. Note also that multiple headers with the same name have been joined using commas to separate values.

Note how only the Content-Type and Content-MD5 HTTP entity headers appear in the StringToSign. The other Content-* entity headers do not.

Again, note that the CanonicalizedResource includes the bucket name, but the HTTP Request-URI does not. (The bucket is specified by the Host header.)

List All My Buckets

Request	StringToSign
GET / HTTP/1.1 Host: s3.amazonaws.com Date: Wed, 28 Mar 2007 01:29:59 +0000	GET\n \n \n

Request	StringToSign
Authorization: <i>AWS</i> <i>AKIAIOSFODNN7EXAMPLE:qGdzdERIC03wnaRNKh6OqZehG9s=</i>	Wed, 28 Mar 2007 01:29:59 +0000\n /

Unicode Keys

Request	StringToSign
GET /dictionary/fran%C3%A7ais/pr%C3%a9f %C3%a8re HTTP/1.1 Host: s3.amazonaws.com Date: Wed, 28 Mar 2007 01:49:49 +0000 Authorization: <i>AWS</i> <i>AKIAIOSFODNN7EXAMPLE:DNEZGsoieTZ92F3bUfSPQcbGm%3%a8re</i>	GET\n \n \n Wed, 28 Mar 2007 01:49:49 +0000\n /dictionary/fran%C3%A7ais/pr%C3%a9f %C3%a8re

Note

The elements in `StringToSign` that were derived from the Request-URI are taken literally, including URL-Encoding and capitalization.

REST Request Signing Problems

When REST request authentication fails, the system responds to the request with an XML error document. The information contained in this error document is meant to help developers diagnose the problem. In particular, the `StringToSign` element of the `SignatureDoesNotMatch` error document tells you exactly what request canonicalization the system is using.

Some toolkits silently insert headers that you do not know about beforehand, such as adding the header `Content-Type` during a PUT. In most of these cases, the value of the inserted header remains constant, allowing you to discover the missing headers by using tools such as `Ethereal` or `tcpmon`.

Query String Request Authentication Alternative

You can authenticate certain types of requests by passing the required information as query-string parameters instead of using the `Authorization` HTTP header. This is useful for enabling direct third-party browser access to your private Amazon S3 data without proxying the request. The idea is to construct a "presigned" request and encode it as a URL that an end-user's browser can retrieve. Additionally, you can limit a presigned request by specifying an expiration time.

Note

For examples of using the AWS SDKs to generating presigned URLs, see [Share an Object with Others](#) (p. 167).

Creating a Signature

Following is an example query string authenticated Amazon S3 REST request.

```
GET /photos/puppy.jpg
?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1141889120&Signature=vjbyPxybdZaNmGa%2ByT272YEAiv4%3D HTTP/1.1
Host: johnsmith.s3.amazonaws.com
```

Date: Mon, 26 Mar 2007 19:37:58 +0000

The query string request authentication method doesn't require any special HTTP headers. Instead, the required authentication elements are specified as query string parameters:

Query String Parameter Name	Example Value	Description
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE	Your AWS access key ID. Specifies the AWS secret access key used to sign the request and, indirectly, the identity of the developer making the request.
Expires	1141889120	The time when the signature expires, specified as the number of seconds since the epoch (00:00:00 UTC on January 1, 1970). A request received after this time (according to the server) will be rejected.
Signature	vjbyPxybdZaNmGa %2ByT272YEAiv4%3D	The URL encoding of the Base64 encoding of the HMAC-SHA1 of StringToSign.

The query string request authentication method differs slightly from the ordinary method but only in the format of the `Signature` request parameter and the `StringToSign` element. Following is pseudo-grammar that illustrates the query string request authentication method.

```
Signature = URL-Encode( Base64( HMAC-SHA1( YourSecretAccessKey, UTF-8-Encoding-Of( StringToSign ) ) ) );

StringToSign = HTTP-VERB + "\n" +
    Content-MD5 + "\n" +
    Content-Type + "\n" +
    Expires + "\n" +
    CanonicalizedAmzHeaders +
    CanonicalizedResource;
```

`YourSecretAccessKey` is the AWS secret access key ID that Amazon assigns to you when you sign up to be an Amazon Web Service developer. Notice how the `Signature` is URL-Encoded to make it suitable for placement in the query string. Note also that in `StringToSign`, the HTTP `Date` positional element has been replaced with `Expires`. The `CanonicalizedAmzHeaders` and `CanonicalizedResource` are the same.

Note

In the query string authentication method, you do not use the `Date` or the `x-amz-date` request header when calculating the string to sign.

Query String Request Authentication

Request	StringToSign
GET /photos/puppy.jpg? AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&	GET\n \n \n

Request	StringToSign
Signature=NpgCjnDzrM %2BWFzoENXmpNDUsSn8%3D& Expires=1175139620 HTTP/1.1 Host: johnsmith.s3.amazonaws.com	1175139620\n /johnsmith/photos/puppy.jpg

We assume that when a browser makes the GET request, it won't provide a Content-MD5 or a Content-Type header, nor will it set any x-amz- headers, so those parts of the `StringToSign` are left blank.

Using Base64 Encoding

HMAC request signatures must be Base64 encoded. Base64 encoding converts the signature into a simple ASCII string that can be attached to the request. Characters that could appear in the signature string like plus (+), forward slash (/), and equals (=) must be encoded if used in a URI. For example, if the authentication code includes a plus (+) sign, encode it as %2B in the request. Encode a forward slash as %2F and equals as %3D.

For examples of Base64 encoding, refer to the Amazon S3 [Authentication Examples](#) (p. 692).

Browser-Based Uploads Using POST (AWS Signature Version 2)

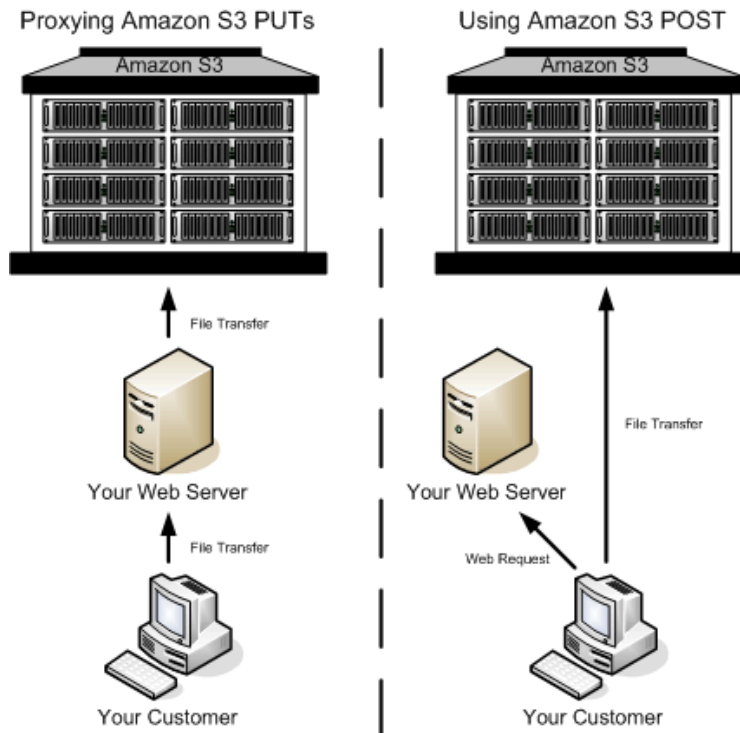
Amazon S3 supports POST, which allows your users to upload content directly to Amazon S3. POST is designed to simplify uploads, reduce upload latency, and save you money on applications where users upload data to store in Amazon S3.

Note

The request authentication discussed in this section is based on AWS Signature Version 2, a protocol for authenticating inbound API requests to AWS services.

Amazon S3 now supports Signature Version 4, a protocol for authenticating inbound API requests to AWS services, in all AWS regions. At this time, AWS regions created before January 30, 2014 will continue to support the previous protocol, Signature Version 2. Any new regions after January 30, 2014 will support only Signature Version 4 and therefore all requests to those regions must be made with Signature Version 4. For more information, see [Authenticating Requests in Browser-Based Uploads Using POST \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

The following figure shows an upload using Amazon S3 POST.



Uploading Using POST

1	The user opens a web browser and accesses your web page.
2	Your web page contains an HTTP form that contains all the information necessary for the user to upload content to Amazon S3.
3	The user uploads content directly to Amazon S3.

Note

Query string authentication is not supported for POST.

HTML Forms (AWS Signature Version 2)

Topics

- [HTML Form Encoding \(p. 699\)](#)
- [HTML Form Declaration \(p. 699\)](#)
- [HTML Form Fields \(p. 700\)](#)
- [Policy Construction \(p. 702\)](#)
- [Constructing a Signature \(p. 705\)](#)
- [Redirection \(p. 705\)](#)

When you communicate with Amazon S3, you normally use the REST or SOAP API to perform put, get, delete, and other operations. With POST, users upload data directly to Amazon S3 through their browsers, which cannot process the SOAP API or create a REST PUT request.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

To allow users to upload content to Amazon S3 by using their browsers, you use HTML forms. HTML forms consist of a form declaration and form fields. The form declaration contains high-level information about the request. The form fields contain detailed information about the request, as well as the policy that is used to authenticate it and ensure that it meets the conditions that you specify.

Note

The form data and boundaries (excluding the contents of the file) cannot exceed 20 KB.

This section explains how to use HTML forms.

HTML Form Encoding

The form and policy must be UTF-8 encoded. You can apply UTF-8 encoding to the form by specifying it in the HTML heading or as a request header.

Note

The HTML form declaration does not accept query string authentication parameters.

The following is an example of UTF-8 encoding in the HTML heading:

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
```

The following is an example of UTF-8 encoding in a request header:

```
Content-Type: text/html; charset=UTF-8
```

HTML Form Declaration

The form declaration has three components: the action, the method, and the enclosure type. If any of these values is improperly set, the request fails.

The action specifies the URL that processes the request, which must be set to the URL of the bucket. For example, if the name of your bucket is "johnsmith", the URL is "http://johnsmith.s3.amazonaws.com/".

Note

The key name is specified in a form field.

The method must be POST.

The enclosure type (enctype) must be specified and must be set to multipart/form-data for both file uploads and text area uploads. For more information, go to [RFC 1867](#).

Example

The following example is a form declaration for the bucket "johnsmith".

```
<form action="http://johnsmith.s3.amazonaws.com/" method="post"
```

```
enctype="multipart/form-data">
```

HTML Form Fields

The following table describes fields that can be used within an HTML form.

Note

The variable `${filename}` is automatically replaced with the name of the file provided by the user and is recognized by all form fields. If the browser or client provides a full or partial path to the file, only the text following the last slash (/) or backslash (\) will be used. For example, "C:\Program Files\directory1\file.txt" will be interpreted as "file.txt". If no file or file name is provided, the variable is replaced with an empty string.

Field Name	Description	Required
<code>AWSSecretKeyId</code>	The AWS Access Key ID of the owner of the bucket who grants an anonymous user access for a request that satisfies the set of constraints in the policy. This field is required if the request includes a policy document.	Conditional
<code>acl</code>	An Amazon S3 access control list (ACL). If an invalid access control list is specified, an error is generated. For more information on ACLs, see Access Control Lists (p. 7) . Type: String Default: private Valid Values: <code>private</code> <code>public-read</code> <code>public-read-write</code> <code>aws-exec-read</code> <code>authenticated-read</code> <code>bucket-owner-read</code> <code>bucket-owner-full-control</code>	No
<code>Cache-Control</code> , <code>Content-Type</code> , <code>Content-Disposition</code> , <code>Content-Encoding</code> , <code>Expires</code>	REST-specific headers. For more information, see PUT Object .	No
<code>key</code>	The name of the uploaded key. To use the filename provided by the user, use the <code>\${filename}</code> variable. For example, if user Betty uploads the file <code>lolcatz.jpg</code> and you specify <code>/user/betty/\${filename}</code> , the file is stored as <code>/user/betty/lolcatz.jpg</code> . For more information, see Object Key and Metadata (p. 99) .	Yes
<code>policy</code>	Security policy describing what is permitted in the request. Requests without a security policy are considered anonymous and will succeed only on publicly writable buckets.	No
<code>success_action_redirect</code> , <code>redirect</code>	The URL to which the client is redirected upon successful upload. Amazon S3 appends the	No

Field Name	Description	Required
	<p>bucket, key, and etag values as query string parameters to the URL.</p> <p>If success_action_redirect is not specified, Amazon S3 returns the empty document type specified in the success_action_status field.</p> <p>If Amazon S3 cannot interpret the URL, it ignores the field.</p> <p>If the upload fails, Amazon S3 displays an error and does not redirect the user to a URL.</p> <p>For more information, see Redirection (p. 705).</p> <p>Note The redirect field name is deprecated and support for the redirect field name will be removed in the future.</p>	
success_action_status	<p>The status code returned to the client upon successful upload if success_action_redirect is not specified.</p> <p>Valid values are 200, 201, or 204 (default).</p> <p>If the value is set to 200 or 204, Amazon S3 returns an empty document with a 200 or 204 status code.</p> <p>If the value is set to 201, Amazon S3 returns an XML document with a 201 status code. For information about the content of the XML document, see POST Object.</p> <p>If the value is not set or if it is set to an invalid value, Amazon S3 returns an empty document with a 204 status code.</p> <p>Note Some versions of the Adobe Flash player do not properly handle HTTP responses with an empty body. To support uploads through Adobe Flash, we recommend setting success_action_status to 201.</p>	No
signature	<p>The HMAC signature constructed by using the secret access key that corresponds to the provided AWSSecretAccessKey. This field is required if a policy document is included with the request.</p> <p>For more information, see Using Auth Access.</p>	Conditional

Field Name	Description	Required
x-amz-security-token	A security token used by session credentials If the request is using Amazon DevPay then it requires two x-amz-security-token form fields: one for the product token and one for the user token. If the request is using session credentials, then it requires one x-amz-security-token form. For more information, see Temporary Security Credentials in the <i>IAM User Guide</i> .	No
Other field names prefixed with x-amz-meta-	User-specified metadata. Amazon S3 does not validate or use this data. For more information, see PUT Object .	No
file	File or text content. The file or content must be the last field in the form. Any fields below it are ignored. You cannot upload more than one file at a time.	Yes

Policy Construction

Topics

- [Expiration \(p. 703\)](#)
- [Conditions \(p. 703\)](#)
- [Condition Matching \(p. 704\)](#)
- [Character Escaping \(p. 704\)](#)

The policy is a UTF-8 and Base64-encoded JSON document that specifies conditions that the request must meet and is used to authenticate the content. Depending on how you design your policy documents, you can use them per upload, per user, for all uploads, or according to other designs that meet your needs.

Note

Although the policy document is optional, we highly recommend it over making a bucket publicly writable.

The following is an example of a policy document:

```
{ "expiration": "2007-12-01T12:00:00.000Z",  
  "conditions": [  
    {"acl": "public-read" },  
    {"bucket": "johnsmith" },  
    ["starts-with", "$key", "user/eric/"],  
  ]  
}
```



```
}
```

The policy document contains the expiration and conditions.

Expiration

The expiration element specifies the expiration date of the policy in ISO 8601 UTC date format. For example, "2007-12-01T12:00:00.000Z" specifies that the policy is not valid after midnight UTC on 2007-12-01. Expiration is required in a policy.

Conditions

The conditions in the policy document validate the contents of the uploaded object. Each form field that you specify in the form (except `AWSAccessKeyId`, signature, file, policy, and field names that have an `x-` ignore- prefix) must be included in the list of conditions.

Note

If you have multiple fields with the same name, the values must be separated by commas. For example, if you have two fields named "x-amz-meta-tag" and the first one has a value of "Ninja" and second has a value of "Stallman", you would set the policy document to `Ninja,Stallman`. All variables within the form are expanded before the policy is validated. Therefore, all condition matching should be performed against the expanded fields. For example, if you set the key field to `user/betty/${filename}`, your policy might be `["starts-with", "$key", "user/betty/"]`. Do not enter `["starts-with", "$key", "user/betty/${filename}"]`. For more information, see [Condition Matching \(p. 704\)](#).

The following table describes policy document conditions.

Element Name	Description
acl	Specifies conditions that the ACL must meet. Supports exact matching and <code>starts-with</code> .
content-length-range	Specifies the minimum and maximum allowable size for the uploaded content. Supports range matching.
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	REST-specific headers. Supports exact matching and <code>starts-with</code> .
key	The name of the uploaded key. Supports exact matching and <code>starts-with</code> .
success_action_redirect, redirect	The URL to which the client is redirected upon successful upload. Supports exact matching and <code>starts-with</code> .
success_action_status	The status code returned to the client upon successful upload if <code>success_action_redirect</code> is not specified. Supports exact matching.
x-amz-security-token	Amazon DevPay security token. Each request that uses Amazon DevPay requires two <code>x-amz-security-token</code> form fields: one for the product token

Element Name	Description
	and one for the user token. As a result, the values must be separated by commas. For example, if the user token is <code>eW91dHVizQ==</code> and the product token is <code>b0hnNVNKWVJIQTA=</code> , you set the policy entry to: { "x-amz-security-token": "eW91dHVizQ==,b0hnNVNKWVJIQTA=" }. Other field names prefixed with <code>x-amz-meta-</code>
	User-specified metadata. Supports exact matching and <code>starts-with</code> .

Note

If your toolkit adds additional fields (e.g., Flash adds filename), you must add them to the policy document. If you can control this functionality, prefix `x-ignore-` to the field so Amazon S3 ignores the feature and it won't affect future versions of this feature.

Condition Matching

The following table describes condition matching types. Although you must specify one condition for each form field that you specify in the form, you can create more complex matching criteria by specifying multiple conditions for a form field.

Condition	Description
Exact Matches	Exact matches verify that fields match specific values. This example indicates that the ACL must be set to public-read: <pre>{"acl": "public-read" }</pre> This example is an alternate way to indicate that the ACL must be set to public-read: <pre>["eq", "\$acl", "public-read"]</pre>
Starts With	If the value must start with a certain value, use <code>starts-with</code> . This example indicates that the key must start with <code>user/betty/</code> : <pre>["starts-with", "\$key", "user/betty/"]</pre>
Matching Any Content	To configure the policy to allow any content within a field, use <code>starts-with</code> with an empty value. This example allows any <code>success_action_redirect</code> : <pre>["starts-with", "\$success_action_redirect", ""]</pre>
Specifying Ranges	For fields that accept ranges, separate the upper and lower ranges with a comma. This example allows a file size from 1 to 10 megabytes: <pre>["content-length-range", 1048579, 10485760]</pre>

Character Escaping

The following table describes characters that must be escaped within a policy document.

Escape Sequence	Description
\\	Backslash
\\$	Dollar sign
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\uxxxx	All Unicode characters

Constructing a Signature

Step	Description
1	Encode the policy by using UTF-8.
2	Encode those UTF-8 bytes by using Base64.
3	Sign the policy with your secret access key by using HMAC SHA-1.
4	Encode the SHA-1 signature by using Base64.

For general information about authentication, see [Using Auth Access](#) .

Redirection

This section describes how to handle redirects.

General Redirection

On completion of the POST request, the user is redirected to the location that you specified in the `success_action_redirect` field. If Amazon S3 cannot interpret the URL, it ignores the `success_action_redirect` field.

If `success_action_redirect` is not specified, Amazon S3 returns the empty document type specified in the `success_action_status` field.

If the POST request fails, Amazon S3 displays an error and does not provide a redirect.

Pre-Upload Redirection

If your bucket was created using `<CreateBucketConfiguration>`, your end users might require a redirect. If this occurs, some browsers might handle the redirect incorrectly. This is relatively rare but is most likely to occur right after a bucket is created.

Upload Examples (AWS Signature Version 2)

Topics

- [File Upload \(p. 706\)](#)
- [Text Area Upload \(p. 708\)](#)

Note

The request authentication discussed in this section is based on AWS Signature Version 2, a protocol for authenticating inbound API requests to AWS services.

Amazon S3 now supports Signature Version 4, a protocol for authenticating inbound API requests to AWS services, in all AWS regions. At this time, AWS regions created before January 30, 2014 will continue to support the previous protocol, Signature Version 2. Any new regions after January 30, 2014 will support only Signature Version 4 and therefore all requests to those regions must be made with Signature Version 4. For more information, see [Examples: Browser-Based Upload using HTTP POST \(Using AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

File Upload

This example shows the complete process for constructing a policy and form that can be used to upload a file attachment.

Policy and Form Construction

The following policy supports uploads to Amazon S3 for the johnsmith bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "johnsmith"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
    {"success_action_redirect": "http://johnsmith.s3.amazonaws.com/successful_upload.html"},
    ["starts-with", "$Content-Type", "image/"],
    {"x-amz-meta-uuid": "14365123651274"},
    ["starts-with", "$x-amz-meta-tag", ""]
  ]
}
```

This policy requires the following:

- The upload must occur before 12:00 UTC on December 1, 2007.
- The content must be uploaded to the johnsmith bucket.
- The key must start with "user/eric/".
- The ACL is set to public-read.
- The success_action_redirect is set to http://johnsmith.s3.amazonaws.com/successful_upload.html.
- The object is an image file.
- The x-amz-meta-uuid tag must be set to 14365123651274.
- The x-amz-meta-tag can contain any value.

The following is a Base64-encoded version of this policy.

```
eyJhZiZxhwaXJhdGlvbiI6IClYMDA3LTExYLTExVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zIjogWwogICAgZyJidWNrZXQiOiA
```

Using your credentials create a signature, for example 0RavWzkygo6QX9caELEqKi9kDbU= is the signature for the preceding policy document.

The following form supports a POST request to the johnsmith.net bucket that uses this policy.

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
    ...
    <form action="http://johnsmith.s3.amazonaws.com/" method="post" enctype="multipart/form-
data">
      Key to upload: <input type="input" name="key" value="user/eric/" /><br />
      <input type="hidden" name="acl" value="public-read" />
      <input type="hidden" name="success_action_redirect" value="http://
johnsmith.s3.amazonaws.com/successful_upload.html" />
      Content-Type: <input type="input" name="Content-Type" value="image/jpeg" /><br />
      <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
      Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />
      <input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />
      <input type="hidden" name="Policy" value="POLICY" />
      <input type="hidden" name="Signature" value="SIGNATURE" />
      File: <input type="file" name="file" /> <br />
      <!-- The elements after this will be ignored -->
      <input type="submit" name="submit" value="Upload to Amazon S3" />
    </form>
    ...
  </html>
```

Sample Request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10) Gecko/20071115
Firefox/2.0.0.10
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=9431149156168
Content-Length: 118698

--9431149156168
Content-Disposition: form-data; name="key"

user/eric/MyPicture.jpg
--9431149156168
Content-Disposition: form-data; name="acl"

public-read
--9431149156168
Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/successful_upload.html
--9431149156168
```

```
Content-Disposition: form-data; name="Content-Type"

image/jpeg
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-tag"

Some,Tag,For,Picture
--9431149156168
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIOSFODNN7EXAMPLE
--9431149156168
Content-Disposition: form-data; name="Policy"

eyJhZXBwaXJhdGlvbiI6IClYMDA3LTExLTExVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zIjogWwogICAgeyJidWNrZXQiOiA
--9431149156168
Content-Disposition: form-data; name="Signature"

ORavWzkygo6QX9caELEqKi9kDbU=
--9431149156168
Content-Disposition: form-data; name="file"; filename="MyFilename.jpg"
Content-Type: image/jpeg

...file content...
--9431149156168
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--9431149156168--
```

Sample Response

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtdFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: http://johnsmith.s3.amazonaws.com/successful_upload.html?
bucket=johnsmith&key=user/eric/
MyPicture.jpg&etag="39d459dfbc0faabbb5e179358dfb94c3&quot;
Server: AmazonS3
```

Text Area Upload

Topics

- [Policy and Form Construction \(p. 708\)](#)
- [Sample Request \(p. 710\)](#)
- [Sample Response \(p. 711\)](#)

The following example shows the complete process for constructing a policy and form to upload a text area. Uploading a text area is useful for submitting user-created content, such as blog postings.

Policy and Form Construction

The following policy supports text area uploads to Amazon S3 for the johnsmith bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "johnsmith"},
    [{"starts-with", "$key", "user/eric/"}],
    {"acl": "public-read"},
    {"success_action_redirect": "http://johnsmith.s3.amazonaws.com/new_post.html"},
    [{"eq", "$Content-Type", "text/html"}],
    {"x-amz-meta-uuid": "14365123651274"},
    [{"starts-with", "$x-amz-meta-tag", ""}]
  ]
}
```

This policy requires the following:

- The upload must occur before 12:00 GMT on 2007-12-01.
- The content must be uploaded to the johnsmith bucket.
- The key must start with "user/eric/".
- The ACL is set to public-read.
- The success_action_redirect is set to http://johnsmith.s3.amazonaws.com/new_post.html.
- The object is HTML text.
- The x-amz-meta-uuid tag must be set to 14365123651274.
- The x-amz-meta-tag can contain any value.

Following is a Base64-encoded version of this policy.

```
eyJhZiZxhwaXJhdGlvbiI6ICImMDA3LDEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXR  
pb25zIjogWwoGICAgeyJidWNRZXQiOiAiam9obnNtaXR0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiLCaidXNlci9lcm1jLy  
LAogICAgeyJhY2wiOiAicHVibG1jLXJlYWQifSwKICAgIHSic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl  
C5zMy5hbWF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlbnQtVHlwZSIsICJ0ZXh0L2h0bWwiXSww  
CAGIHSieC1hbXotbWV0YS1ldWlkIjogIjE0MTIzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiLCaidXNlci9lcm1jLy  
IsICJlXQogIF0KfQo=
```

Using your credentials, create a signature. For example, qA7FWXKq6VvU681I9KdveT1cWgF= is the signature for the preceding policy document.

The following form supports a POST request to the johnsmith.net bucket that uses this policy.

```
<html>
<head>
  ...
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  ...
</head>
<body>
  ...
  <form action="http://johnsmith.s3.amazonaws.com/" method="post" enctype="multipart/form-
data">
    Key to upload: <input type="input" name="key" value="user/eric/" /><br />
    <input type="hidden" name="acl" value="public-read" />
    <input type="hidden" name="success_action_redirect" value="http://
johnsmith.s3.amazonaws.com/new_post.html" />
    <input type="hidden" name="Content-Type" value="text/html" />
    <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
    Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />
    <input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />
    <input type="hidden" name="Policy" value="POLICY" />
    <input type="hidden" name="Signature" value="SIGNATURE" />
    Entry: <textarea name="file" cols="60" rows="10">
```

Your blog post goes here.

```
</textarea><br />
<!-- The elements after this will be ignored -->
<input type="submit" name="submit" value="Upload to Amazon S3" />
</form>
...
</html>
```

Sample Request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10) Gecko/20071115
  Firefox/2.0.0.10
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=178521717625888
Content-Length: 118635

-178521717625888
Content-Disposition: form-data; name="key"

ser/eric/NewEntry.html
--178521717625888
Content-Disposition: form-data; name="acl"

public-read
--178521717625888
Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/new_post.html
--178521717625888
Content-Disposition: form-data; name="Content-Type"

text/html
--178521717625888
Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274
--178521717625888
Content-Disposition: form-data; name="x-amz-meta-tag"

Interesting Post
--178521717625888
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIOSFODNN7EXAMPLE
--178521717625888
Content-Disposition: form-data; name="Policy"
eyJhZXBwaXJhdGlvbiI6ICl0MDA3LTExLTExVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zIjogWwogICAgeyJidWNrZXQiOiA
--178521717625888
Content-Disposition: form-data; name="Signature"

qA7FWXKq6VvU681I9KdveT1cWgF=
```



```
--178521717625888
Content-Disposition: form-data; name="file"

...content goes here...
--178521717625888
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--178521717625888--
```

Sample Response

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: http://johnsmith.s3.amazonaws.com/new_post.html?bucket=johnsmith&key=user/eric/
NewEntry.html&etag=40c3271af26b7f1672e41b8a274d28d4
Server: AmazonS3
```

POST with Adobe Flash

This section describes how to use POST with Adobe Flash.

Adobe Flash Player Security

By default, the Adobe Flash Player security model prohibits Adobe Flash Players from making network connections to servers outside the domain that serves the SWF file.

To override the default, you must upload a publicly readable `crossdomain.xml` file to the bucket that will accept POST uploads. The following is a sample `crossdomain.xml` file.

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="*" secure="false" />
</cross-domain-policy>
```

Note

For more information about the Adobe Flash security model, go to the Adobe website. Adding the `crossdomain.xml` file to your bucket allows any Adobe Flash Player to connect to the `crossdomain.xml` file within your bucket; however, it does not grant access to the actual Amazon S3 bucket.

Adobe Flash Considerations

The FileReference API in Adobe Flash adds the `Filename` form field to the POST request. When you build Adobe Flash applications that upload to Amazon S3 by using the FileReference API action, include the following condition in your policy:

```
['starts-with', '$Filename', '']
```

Some versions of the Adobe Flash Player do not properly handle HTTP responses that have an empty body. To configure POST to return a response that does not have an empty body, set `success_action_status` to 201. Amazon S3 will then return an XML document with a 201 status

code. For information about the content of the XML document, see [POST Object](#). For information about form fields, see [HTML Form Fields \(p. 700\)](#).

Amazon S3 Resources

Following is a table that lists related resources that you'll find useful as you work with this service.

Resource	Description
Amazon Simple Storage Service Getting Started Guide	The Getting Started Guide provides a quick tutorial of the service based on a simple use case.
Amazon Simple Storage Service API Reference	The API Reference describes Amazon S3 operations in detail.
Amazon S3 Technical FAQ	The FAQ covers the top questions developers have asked about this product.
AWS Developer Resource Center	A central starting point to find documentation, code samples, release notes, and other information to help you build innovative applications with AWS.
AWS Management Console	The console allows you to perform most of the functions of Amazon S3 without programming.
https://forums.aws.amazon.com/	A community-based forum for developers to discuss technical questions related to AWS.
AWS Support Center	The home page for AWS Technical Support, including access to our Developer Forums, Technical FAQs, Service Status page, and Premium Support.
AWS Premium Support	The primary web page for information about AWS Premium Support, a one-on-one, fast-response support channel to help you build and run applications on AWS Infrastructure Services.
Amazon S3 product information	The primary web page for information about Amazon S3.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse, etc.
Conditions of Use	Detailed information about the copyright and trademark usage at Amazon.com and other topics.

SQL Reference for Amazon S3 Select and Glacier Select

This reference contains a description of the structured query language (SQL) elements that are supported by Amazon S3 Select and Glacier Select.

Topics

- [SELECT Command](#) (p. 714)
- [Data Types](#) (p. 721)
- [Operators](#) (p. 721)
- [Reserved Keywords](#) (p. 723)
- [SQL Functions](#) (p. 727)

SELECT Command

Amazon S3 Select and Glacier Select support only the `SELECT` SQL command. The following ANSI standard clauses are supported for `SELECT`:

- `SELECT` list
- `FROM` clause
- `WHERE` clause
- `LIMIT` clause (Amazon S3 Select only)

Note

Amazon S3 Select and Glacier Select queries currently do not support subqueries or joins.

SELECT List

The `SELECT` list names the columns, functions, and expressions that you want the query to return. The list represents the output of the query.

```
SELECT *  
SELECT projection [ AS column_alias | column_alias ] [, ...]
```

The first form with `*` (asterisk) returns every row that passed the `WHERE` clause, as-is. The second form creates a row with user-defined output scalar expressions ***projection*** for each column.

FROM Clause

Amazon S3 Select and Glacier Select support the following forms of the `FROM` clause:

```
FROM table_name  
FROM table_name alias  
FROM table_name AS alias
```

Where `table_name` is one of `S3Object` (for Amazon S3 Select) or `ARCHIVE` or `OBJECT` (for Glacier Select) referring to the archive being queried over. Users coming from traditional relational databases can think of this as a database schema that contains multiple views over a table.

Following standard SQL, the `FROM` clause creates rows that are filtered in the `WHERE` clause and projected in the `SELECT` list.

For JSON objects that are stored in Amazon S3 Select, you can also use the following forms of the `FROM` clause:

```
FROM S3Object[*].path
FROM S3Object[*].path alias
FROM S3Object[*].path AS alias
```

Using this form of the `FROM` clause, you can select from arrays or objects within a JSON object. You can specify `path` using one of the following forms:

- By name (in an object): `.name` or `['name']`
- By index (in an array): `[index]`
- By wildcard (in an object): `.*`
- By wildcard (in an array): `[*]`

Note

- This form of the `FROM` clause works only with JSON objects.
- Wildcards always emit at least one record. If no record matches, then Amazon S3 Select emits the value `MISSING`. During output serialization (after the query is complete), Amazon S3 Select replaces `MISSING` values with empty records.
- Aggregate functions (`AVG`, `COUNT`, `MAX`, `MIN`, and `SUM`) skip `MISSING` values.
- If you don't provide an alias when using a wildcard, you can refer to the row using the last element in the path. For example, you could select all prices from a list of books using the query `SELECT price FROM S3Object[*].books[*].price`. If the path ends in a wildcard rather than a name, then you can use the value `_1` to refer to the row. For example, instead of `SELECT price FROM S3Object[*].books[*].price`, you could use the query `SELECT _1.price FROM S3Object[*].books[*]`.
- Amazon S3 Select always treats a JSON document as an array of root-level values. Thus, even if the JSON object that you are querying has only one root element, the `FROM` clause must begin with `S3Object[*]`. However, for compatibility reasons, Amazon S3 Select allows you to omit the wildcard if you don't include a path. Thus, the complete clause `FROM S3Object` is equivalent to `FROM S3Object[*] as S3Object`. If you include a path, you must also use the wildcard. So `FROM S3Object` and `FROM S3Object[*].path` are both valid clauses, but `FROM S3Object.path` is not.

Example

Examples:

Example #1

This example shows results using the following dataset and query:

```
{
  "Rules": [
    {"id": "id-1", "condition": "x < 20"},
    {"condition": "y > x"},
    {"id": "id-2", "condition": "z = DEBUG"}
  ]
}
```

```
    ]
  },
  {
    "created": "June 27",
    "modified": "July 6"
  }
}
```

```
SELECT id FROM S3Object[*].Rules[*].id
```

```
{ "id": "id-1" },
{ },
{ "id": "id-2" },
{ }
```

Amazon S3 Select produces each result for the following reasons:

- `{"id": "id-1"}` — `S3Object[0].Rules[0].id` produced a match.
- `{ }` — `S3Object[0].Rules[1].id` did not match a record, so Amazon S3 Select emitted `MISSING`, which was then changed to an empty record during output serialization and returned.
- `{"id": "id-2"}` — `S3Object[0].Rules[2].id` produced a match.
- `{ }` — `S3Object[1]` did not match on `Rules`, so Amazon S3 Select emitted `MISSING`, which was then changed to an empty record during output serialization and returned.

If you don't want Amazon S3 Select to return empty records when it doesn't find a match, you can test for the value `MISSING`. The following query returns the same results as the previous query, but with the empty values omitted:

```
SELECT id FROM S3Object[*].Rules[*].id WHERE id IS NOT MISSING
```

```
{ "id": "id-1" },
{ "id": "id-2" }
```

Example #2

This example shows results using the following dataset and queries:

```
{
  "created": "936864000",
  "dir_name": "important_docs",
  "files": [
    {
      "name": "."
    },
    {
      "name": ".."
    },
    {
      "name": ".aws"
    },
    {
      "name": "downloads"
    }
  ],
  "owner": "AWS S3"
},
{
  "created": "936864000",
```

```
"dir_name": "other_docs",
"files": [
  {
    "name": "."
  },
  {
    "name": ".."
  },
  {
    "name": "my stuff"
  },
  {
    "name": "backup"
  }
],
"owner": "User"
}
```

```
SELECT d.dir_name, d.files FROM S3Object[*] d
```

```
{
  "dir_name": "important_docs",
  "files": [
    {
      "name": "."
    },
    {
      "name": ".."
    },
    {
      "name": ".aws"
    },
    {
      "name": "downloads"
    }
  ]
},
{
  "dir_name": "other_docs",
  "files": [
    {
      "name": "."
    },
    {
      "name": ".."
    },
    {
      "name": "my stuff"
    },
    {
      "name": "backup"
    }
  ]
}
```

```
SELECT _1.dir_name, _1.owner FROM S3Object[*]
```

```
{
  "dir_name": "important_docs",
  "owner": "AWS S3"
},
```

```
{
  "dir_name": "other_docs",
  "owner": "User"
}
```

WHERE Clause

The `WHERE` clause follows this syntax:

```
WHERE condition
```

The `WHERE` clause filters rows based on the *condition*. A condition is an expression that has a Boolean result. Only rows for which the condition evaluates to `TRUE` are returned in the result.

LIMIT Clause (Amazon S3 Select only)

The `LIMIT` clause follows this syntax:

```
LIMIT number
```

The `LIMIT` clause limits the number of records that you want the query to return based on *number*.

Note

Glacier Select does not support the `LIMIT` clause.

Attribute Access

The `SELECT` and `WHERE` clauses can refer to record data using one of the methods in the following sections, depending on whether the file that is being queried is in CSV or JSON format.

CSV

- **Column Numbers** – You can refer to the *Nth* column of a row with the column name `_N`, where *N* is the column position. The position count starts at 1. For example, the first column is named `_1` and the second column is named `_2`.

You can refer to a column as `_N` or `alias._N`. For example, `_2` and `myAlias._2` are both valid ways to refer to a column in the `SELECT` list and `WHERE` clause.

- **Column Headers** – For objects in CSV format that have a header row, the headers are available to the `SELECT` list and `WHERE` clause. In particular, as in traditional SQL, within `SELECT` and `WHERE` clause expressions, you can refer to the columns by `alias.column_name` or `column_name`.

JSON (Amazon S3 Select only)

- **Document** – You can access JSON document fields as `alias.name`. Nested fields can also be accessed; for example, `alias.name1.name2.name3`.
- **List** – You can access elements in a JSON list using zero-based indexes with the `[]` operator. For example, you can access the second element of a list as `alias[1]`. Accessing list elements can be combined with fields as `alias.name1.name2[1].name3`.
- **Examples:** Consider this JSON object as a sample dataset:

```
{"name": "Susan Smith",
"org": "engineering",
"projects":
```



```
[
  {"project_name":"project1", "completed":false},
  {"project_name":"project2", "completed":true}
]
```

Example #1

The following query returns these results:

```
Select s.name from S3Object s
```

```
{"name":"Susan Smith"}
```

Example #2

The following query returns these results:

```
Select s.projects[0].project_name from S3Object s
```

```
{"project_name":"project1"}
```

Case Sensitivity of Header/Attribute Names

With Amazon S3 Select and Glacier Select, you can use double quotation marks to indicate that column headers (for CSV objects) and attributes (for JSON objects) are case sensitive. Without double quotation marks, object headers/attributes are case insensitive. An error is thrown in cases of ambiguity.

The following examples are either 1) Amazon S3 or Glacier objects in CSV format with the specified column header(s), and with `FileHeaderInfo` set to "Use" for the query request; or 2) Amazon S3 objects in JSON format with the specified attributes.

Example #1: The object being queried has header/attribute "NAME".

- The following expression successfully returns values from the object (no quotation marks: case insensitive):

```
SELECT s.name from S3Object s
```

- The following expression results in a 400 error `MissingHeaderName` (quotation marks: case sensitive):

```
SELECT s."name" from S3Object s
```

Example #2: The Amazon S3 object being queried has one header/attribute with "NAME" and another header/attribute with "name".

- The following expression results in a 400 error `AmbiguousFieldName` (no quotation marks: case insensitive, but there are two matches):

```
SELECT s.name from S3Object s
```

- The following expression successfully returns values from the object (quotation marks: case sensitive, so it resolves the ambiguity).

```
SELECT s."NAME" from S3Object s
```

Using Reserved Keywords as User-Defined Terms

Amazon S3 Select and Glacier Select have a set of reserved keywords that are needed to execute the SQL expressions used to query object content. Reserved keywords include function names, data types, operators, and so on. In some cases, user-defined terms like the column headers (for CSV files) or attributes (for JSON object) may clash with a reserved keyword. When this happens, you must use double quotation marks to indicate that you are intentionally using a user-defined term that clashes with a reserved keyword. Otherwise a 400 parse error will result.

For the full list of reserved keywords see [Reserved Keywords \(p. 723\)](#).

The following example is either 1) an Amazon S3 or Glacier object in CSV format with the specified column headers, with `FileHeaderInfo` set to "Use" for the query request, or 2) an Amazon S3 object in JSON format with the specified attributes.

Example: The object being queried has header/attribute named "CAST", which is a reserved keyword.

- The following expression successfully returns values from the object (quotation marks: use user-defined header/attribute):

```
SELECT s."CAST" from S3Object s
```

- The following expression results in a 400 parse error (no quotation marks: clash with reserved keyword):

```
SELECT s.CAST from S3Object s
```

Scalar Expressions

Within the `WHERE` clause and the `SELECT` list, you can have SQL *scalar expressions*, which are expressions that return scalar values. They have the following form:

- ***literal***

An SQL literal.

- ***column_reference***

A reference to a column in the form *column_name* or *alias.column_name*.

- ***unary_op expression***

Where ***unary_op*** unary is an SQL unary operator.

- ***expression binary_op expression***

Where ***binary_op*** is an SQL binary operator.

- ***func_name***

Where ***func_name*** is the name of a scalar function to invoke.

- ***expression*** [`NOT`] `BETWEEN` ***expression*** `AND` ***expression***
- ***expression*** `LIKE` ***expression*** [`ESCAPE` ***expression***]

Data Types

Amazon S3 Select and Glacier Select support several primitive data types.

Data Type Conversions

The general rule is to follow the `CAST` function if defined. If `CAST` is not defined, then all input data is treated as a string. It must be cast into the relevant data types when necessary.

For more information about the `CAST` function, see [CAST \(p. 729\)](#).

Supported Data Types

Amazon S3 Select and Glacier Select support the following set of primitive data types.

Name	Description	Examples
bool	TRUE or FALSE	FALSE
int, integer	8-byte signed integer in the range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.	100000
string	UTF8-encoded variable-length string. The default limit is one character. The maximum character limit is 2,147,483,647.	'xyz '
float	8-byte floating point number.	CAST(0.456 AS FLOAT)
decimal, numeric	Base-10 number, with maximum precision of 38 (that is, the maximum number of significant digits), and with scale within the range of -2^{31} to $2^{31}-1$ (that is, the base-10 exponent).	123.456
timestamp	<p>Time stamps represent a specific moment in time, always include a local offset, and are capable of arbitrary precision.</p> <p>In the text format, time stamps follow the W3C note on date and time formats, but they must end with the literal "T" if not at least whole-day precision. Fractional seconds are allowed, with at least one digit of precision, and an unlimited maximum. Local-time offsets can be represented as either hour:minute offsets from UTC, or as the literal "Z" to denote a local time of UTC. They are required on time stamps with time and are not allowed on date values.</p>	CAST('2007-04-05T14:30Z' AS TIMESTAMP)

Operators

Amazon S3 Select and Glacier Select support the following operators.

Logical Operators

- AND
- NOT
- OR

Comparison Operators

- <
- >
- <=
- >=
- =
- <>
- !=
- BETWEEN
- IN – For example: IN ('a' , 'b' , 'c')

Pattern Matching Operators

- LIKE

Math Operators

Addition, subtraction, multiplication, division, and modulo are supported.

- +
- -
- *
- %

Operator Precedence

The following table shows the operators' precedence in decreasing order.

Operator/ Element	Associativity	Required
-	right	unary minus
*, /, %	left	multiplication, division, modulo
+, -	left	addition, subtraction
IN		set membership
BETWEEN		range containment
LIKE		string pattern matching
<>		less than, greater than

Operator/ Element	Associativity	Required
=	right	equality, assignment
NOT	right	logical negation
AND	left	logical conjunction
OR	left	logical disjunction

Reserved Keywords

Below is the list of reserved keywords for Amazon S3 Select and Glacier Select. These include function names, data types, operators, etc., that needed to execute the SQL expressions used to query object content.

```
absolute
action
add
all
allocate
alter
and
any
are
as
asc
assertion
at
authorization
avg
bag
begin
between
bit
bit_length
blob
bool
boolean
both
by
cascade
cascaded
case
cast
catalog
char
char_length
character
character_length
check
clob
close
coalesce
collate
collation
```

column
commit
connect
connection
constraint
constraints
continue
convert
corresponding
count
create
cross
current
current_date
current_time
current_timestamp
current_user
cursor
date
day
deallocate
dec
decimal
declare
default
deferrable
deferred
delete
desc
describe
descriptor
diagnostics
disconnect
distinct
domain
double
drop
else
end
end-exec
escape
except
exception
exec
execute
exists
external
extract
false
fetch
first
float
for
foreign
found
from
full
get
global
go
goto
grant
group
having
hour
identity

immediate
in
indicator
initially
inner
input
insensitive
insert
int
integer
intersect
interval
into
is
isolation
join
key
language
last
leading
left
level
like
limit
list
local
lower
match
max
min
minute
missing
module
month
names
national
natural
nchar
next
no
not
null
nullif
numeric
octet_length
of
on
only
open
option
or
order
outer
output
overlaps
pad
partial
pivot
position
precision
prepare
preserve
primary
prior
privileges
procedure

public
read
real
references
relative
restrict
revoke
right
rollback
rows
schema
scroll
second
section
select
session
session_user
set
sexp
size
smallint
some
space
sql
sqlcode
sqlerror
sqlstate
string
struct
substring
sum
symbol
system_user
table
temporary
then
time
timestamp
timezone_hour
timezone_minute
to
trailing
transaction
translate
translation
trim
true
tuple
union
unique
unknown
unpivot
update
upper
usage
user
using
value
values
varchar
varying
view
when
whenever
where
with


```
work  
write  
year  
zone
```

SQL Functions

Amazon S3 Select and Glacier Select support several SQL functions.

Topics

- [Aggregate Functions \(Amazon S3 Select only\) \(p. 727\)](#)
- [Conditional Functions \(p. 728\)](#)
- [Conversion Functions \(p. 729\)](#)
- [Date Functions \(p. 729\)](#)
- [String Functions \(p. 735\)](#)

Aggregate Functions (Amazon S3 Select only)

Amazon S3 Select supports the following aggregate functions.

Note

Glacier Select does not support aggregate functions.

Function	Argument Type	Return Type
AVG(expression)	INT, FLOAT, DECIMAL	DECIMAL for an INT argument, FLOAT for a floating-point argument; otherwise the same as the argument data type.
COUNT	-	INT
MAX(expression)	INT, DECIMAL	Same as the argument type.
MIN(expression)	INT, DECIMAL	Same as the argument type.
SUM(expression)	INT, FLOAT, DOUBLE, DECIMAL	INT for INT argument, FLOAT for a floating-point argument; otherwise, the same as the argument data type.

Conditional Functions

Amazon S3 Select and Glacier Select support the following conditional functions.

Topics

- [COALESCE \(p. 728\)](#)
- [NULLIF \(p. 728\)](#)

COALESCE

Evaluates the arguments in order and returns the first non-unknown, that is, the first non-null or non-missing. This function does not propagate null and missing.

Syntax

```
COALESCE ( expression, expression, ... )
```

Parameters

expression

The target expression that the function operates on.

Examples

```
COALESCE(1)                -- 1
COALESCE(null)             -- null
COALESCE(null, null)       -- null
COALESCE(missing)          -- null
COALESCE(missing, missing) -- null
COALESCE(1, null)          -- 1
COALESCE(null, null, 1)    -- 1
COALESCE(null, 'string')   -- 'string'
COALESCE(missing, 1)       -- 1
```

NULLIF

Given two expressions, returns NULL if the two expressions evaluate to the same value; otherwise, returns the result of evaluating the first expression.

Syntax

```
NULLIF ( expression1, expression2 )
```

Parameters

expression1, expression2

The target expressions that the function operates on.

Examples

```
NULLIF(1, 1)                -- null
```

```
NULLIF(1, 2)           -- 1
NULLIF(1.0, 1)         -- null
NULLIF(1, '1')         -- 1
NULLIF([1], [1])       -- null
NULLIF(1, NULL)        -- 1
NULLIF(NULL, 1)        -- null
NULLIF(null, null)     -- null
NULLIF(missing, null)  -- null
NULLIF(missing, missing) -- null
```

Conversion Functions

Amazon S3 Select and Glacier Select support the following conversion functions.

Topics

- [CAST \(p. 729\)](#)

CAST

The CAST function converts an entity, such as an expression that evaluates to a single value, from one type to another.

Syntax

```
CAST ( expression AS data_type )
```

Parameters

expression

A combination of one or more values, operators, and SQL functions that evaluate to a value.

data_type

The target data type, such as INT, to cast the expression to. For a list of supported data types, see [Data Types \(p. 721\)](#).

Examples

```
CAST('2007-04-05T14:30Z' AS TIMESTAMP)
CAST(0.456 AS FLOAT)
```

Date Functions

Amazon S3 Select and Glacier Select support the following date functions.

Topics

- [DATE_ADD \(p. 730\)](#)
- [DATE_DIFF \(p. 730\)](#)
- [EXTRACT \(p. 731\)](#)
- [TO_STRING \(p. 732\)](#)
- [TO_TIMESTAMP \(p. 734\)](#)

- [UTCNOW \(p. 735\)](#)

DATE_ADD

Given a date part, a quantity, and a time stamp, returns an updated time stamp by altering the date part by the quantity.

Syntax

```
DATE_ADD( date_part, quantity, timestamp )
```

Parameters

date_part

Specifies which part of the date to modify. This can be one of the following:

- year
- month
- day
- hour
- minute
- second

quantity

The value to apply to the updated time stamp. Positive values for quantity add to the time stamp's date_part, and negative values subtract.

timestamp

The target time stamp that the function operates on.

Examples

```
DATE_ADD(year, 5, `2010-01-01T`)           -- 2015-01-01 (equivalent to 2015-01-01T)
DATE_ADD(month, 1, `2010T`)                 -- 2010-02T (result will add precision as
necessary)
DATE_ADD(month, 13, `2010T`)                 -- 2011-02T
DATE_ADD(day, -1, `2017-01-10T`)             -- 2017-01-09 (equivalent to 2017-01-09T)
DATE_ADD(hour, 1, `2017T`)                  -- 2017-01-01T01:00-00:00
DATE_ADD(hour, 1, `2017-01-02T03:04Z`)      -- 2017-01-02T04:04Z
DATE_ADD(minute, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:05:05.006Z
DATE_ADD(second, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:04:06.006Z
```

DATE_DIFF

Given a date part and two valid time stamps, returns the difference in date parts. The return value is a negative integer when the date_part value of timestamp1 is greater than the date_part value of timestamp2. The return value is a positive integer when the date_part value of timestamp1 is less than the date_part value of timestamp2.

Syntax

```
DATE_DIFF( date_part, timestamp1, timestamp2 )
```

Parameters

date_part

Specifies which part of the time stamps to compare. For the definition of *date_part*, see [DATE_ADD \(p. 730\)](#).

timestamp1

The first time stamp to compare.

timestamp2

The second time stamp to compare.

Examples

```
DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`)           -- 1
DATE_DIFF(year, `2010T`, `2010-05T`)                   -- 4 (2010T is equivalent to
  2010-01-01T00:00:00.000Z)
DATE_DIFF(month, `2010T`, `2011T`)                     -- 12
DATE_DIFF(month, `2011T`, `2010T`)                     -- -12
DATE_DIFF(day, `2010-01-01T23:00T`, `2010-01-02T01:00T`) -- 0 (need to be at least 24h
  apart to be 1 day apart)
```

EXTRACT

Given a date part and a time stamp, returns the time stamp's date part value.

Syntax

```
EXTRACT( date_part FROM timestamp )
```

Parameters

date_part

Specifies which part of the time stamps to extract. This can be one of the following:

- year
- month
- day
- hour
- minute
- second
- timezone_hour
- timezone_minute

timestamp

The target time stamp that the function operates on.

Examples

```
EXTRACT(YEAR FROM `2010-01-01T`)                       -- 2010
EXTRACT(MONTH FROM `2010T`)                             -- 1 (equivalent to
  2010-01-01T00:00:00.000Z)
```

```
EXTRACT(MONTH FROM `2010-10T`) -- 10
EXTRACT(HOUR FROM `2017-01-02T03:04:05+07:08`) -- 3
EXTRACT(MINUTE FROM `2017-01-02T03:04:05+07:08`) -- 4
EXTRACT(TIMEZONE_HOUR FROM `2017-01-02T03:04:05+07:08`) -- 7
EXTRACT(TIMEZONE_MINUTE FROM `2017-01-02T03:04:05+07:08`) -- 8
```

TO_STRING

Given a time stamp and a format pattern, returns a string representation of the time stamp in the given format.

Syntax

```
TO_STRING ( timestamp time_format_pattern )
```

Parameters

timestamp

The target time stamp that the function operates on.

time_format_pattern

A string that has the following special character interpretations.

Format	Example	Description
yy	69	2-digit year
y	1969	4-digit year
yyyy	1969	Zero-padded 4-digit year
M	1	Month of year
MM	01	Zero-padded month of year
MMM	Jan	Abbreviated month year name
MMMM	January	Full month of year name
MMMMM	J	Month of year first letter (NOTE: not valid for use with to_timestamp function)
d	2	Day of month (1-31)
dd	02	Zero-padded day of month (01-31)

Format	Example	Description
a	AM	AM or PM of day
h	3	Hour of day (1-12)
hh	03	Zero-padded hour of day (01-12)
H	3	Hour of day (0-23)
HH	03	Zero-padded hour of day (00-23)
m	4	Minute of hour (0-59)
mm	04	Zero-padded minute of hour (00-59)
s	5	Second of minute (0-59)
ss	05	Zero-padded second of minute (00-59)
S	0	Fraction of second (precision: 0.1, range: 0.0-0.9)
SS	6	Fraction of second (precision: 0.01, range: 0.0-0.99)
SSS	60	Fraction of second (precision: 0.001, range: 0.0-0.999)
...
SSSSSSSSS	60000000	Fraction of second (maximum precision: 1 nanosecond, range: 0.0-0.999999999)
n	600000000	Nano of second

Format	Example	Description
X	+07 or Z	Offset in hours or "Z" if the offset is 0
XX or XXXX	+0700 or Z	Offset in hours and minutes or "Z" if the offset is 0
XXX or XXXXX	+07:00 or Z	Offset in hours and minutes or "Z" if the offset is 0
x	7	Offset in hours
xx or xxxx	700	Offset in hours and minutes
xxx or xxxxx	+07:00	Offset in hours and minutes

Examples

```
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y')           -- "July 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMM d, yyyy')         -- "Jul 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'M-d-yy')              -- "7-20-69"
TO_STRING(`1969-07-20T20:18Z`, 'MM-d-y')              -- "07-20-1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y h:m a')     -- "July 20, 1969 8:18 PM"
TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd'T' 'H:m:ssX') -- "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00Z`, 'y-MM-dd'T' 'H:m:ssX') -- "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd'T' 'H:m:ssXXXX') --
  "1969-07-20T20:18:00+0800"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd'T' 'H:m:ssXXXXX') --
  "1969-07-20T20:18:00+08:00"
```

TO_TIMESTAMP

Given a string, converts it to a time stamp. This is the inverse operation of TO_STRING.

Syntax

```
TO_TIMESTAMP ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
TO_TIMESTAMP('2007T')           -- `2007T`
```



```
TO_TIMESTAMP('2007-02-23T12:14:33.079-08:00') -- `2007-02-23T12:14:33.079-08:00`
```

UTCNOW

Returns the current time in UTC as a time stamp.

Syntax

```
UTCNOW()
```

Parameters

none

Examples

```
UTCNOW() -- 2017-10-13T16:02:11.123Z
```

String Functions

Amazon S3 Select and Glacier Select support the following string functions.

Topics

- [CHAR_LENGTH, CHARACTER_LENGTH \(p. 735\)](#)
- [LOWER \(p. 736\)](#)
- [SUBSTRING \(p. 736\)](#)
- [TRIM \(p. 737\)](#)
- [UPPER \(p. 737\)](#)

CHAR_LENGTH, CHARACTER_LENGTH

Counts the number of characters in the specified string.

Note

CHAR_LENGTH and CHARACTER_LENGTH are synonyms.

Syntax

```
CHAR_LENGTH ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
CHAR_LENGTH('') -- 0
```

```
CHAR_LENGTH( 'abcdefg' )    -- 7
```

LOWER

Given a string, converts all uppercase characters to lowercase characters. Any non-uppercased characters remain unchanged.

Syntax

```
LOWER ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
LOWER( 'AbCdEfG!@#$$' ) -- 'abcdefg!@#$$'
```

SUBSTRING

Given a string, a start index, and optionally a length, returns the substring from the start index up to the end of the string, or up to the length provided.

Note

The first character of the input string has index 1. If *start* is < 1, it is set to 1.

Syntax

```
SUBSTRING( string FROM start [ FOR length ] )
```

Parameters

string

The target string that the function operates on.

start

The start position of the string.

length

The length of the substring to return. If not present, proceed to the end of the string.

Examples

```
SUBSTRING("123456789", 0)      -- "123456789"
SUBSTRING("123456789", 1)      -- "123456789"
SUBSTRING("123456789", 2)      -- "23456789"
SUBSTRING("123456789", -4)     -- "123456789"
SUBSTRING("123456789", 0, 999) -- "123456789"
SUBSTRING("123456789", 1, 5)   -- "12345"
```

TRIM

Trims leading or trailing characters from a string. The default character to remove is ' '.

Syntax

```
TRIM ( [[LEADING | TRAILING | BOTH remove_chars] FROM] string )
```

Parameters

string

The target string that the function operates on.

LEADING | TRAILING | BOTH

Whether to trim leading or trailing characters, or both leading and trailing characters.

remove_chars

The set of characters to remove. Note that *remove_chars* can be a string with length > 1. This function returns the string with any character from *remove_chars* found at the beginning or end of the string that was removed.

Examples

```
TRIM('      foobar      ')      -- 'foobar'
TRIM('      \tfoobar\t      ')  -- '\tfoobar\t'
TRIM(LEADING FROM '      foobar      ') -- 'foobar'
TRIM(TRAILING FROM '      foobar      ') -- '      foobar'
TRIM(BOTH FROM '      foobar      ')  -- 'foobar'
TRIM(BOTH '12' FROM '1112211foobar22211122') -- 'foobar'
```

UPPER

Given a string, converts all lowercase characters to uppercase characters. Any non-lowercased characters remain unchanged.

Syntax

```
UPPER ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
UPPER('AbCdEfG!@#') -- 'ABCDEFG!@#'
```

Document History

- **Latest documentation update:** September 18, 2019
- **Current API version:** 2006-03-01

The following table describes the important changes in each release of the *Amazon Simple Storage Service Developer Guide* from June 19, 2018, onward. For notification about updates to this documentation, you can subscribe to an RSS feed.

update-history-change	update-history-description	update-history-date
Same-Region replication (p. 738)	Same-Region replication (SRR) is used to copy objects across Amazon S3 buckets in the same AWS Region. For information about both cross-Region and same-Region replication, see Replication .	September 18, 2019
Cross-Region replication support for object lock (p. 738)	Cross-Region replication now supports Amazon S3 object lock. For more information, see Cross-Region Replication and What Does Amazon S3 Replicate? .	May 28, 2019
Amazon S3 batch operations (p. 738)	Using Amazon S3 batch operations you can perform large-scale batch operations on Amazon S3 objects. Amazon S3 batch operations can execute a single operation on lists of objects that you specify. A single job can perform the specified operation on billions of objects containing exabytes of data. For more information, see Performing Batch Operations .	April 30, 2019
Asia Pacific (Hong Kong) Region (p. 738)	Amazon S3 is now available in the Asia Pacific (Hong Kong) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	April 24, 2019
Added a new field to the server access logs (p. 738)	Amazon S3 added the following new field to the server access logs: Transport Layer Security (TLS) version. For more information, see Amazon S3 Server Access Log Format .	March 28, 2019

New archive storage class (p. 738)	Amazon S3 now offers a new archive storage class, DEEP_ARCHIVE, for storing rarely accessed objects. For more information, see Storage Classes .	March 27, 2019
Added new fields to the server access logs (p. 738)	Amazon S3 added the following new fields to the server access logs: Host Id, Signature Version, Cipher Suite, Authentication Type, and Host Header. For more information, see Amazon S3 Server Access Log Format .	March 5, 2019
Support for Parquet-formatted Amazon S3 inventory files (p. 738)	Amazon S3 now supports the Apache Parquet (Parquet) format in addition to the Apache optimized row columnar (ORC) and comma-separated values (CSV) file formats for inventory output files. For more information, see Amazon S3 Inventory .	December 4, 2018
Restore speed upgrade (p. 738)	Using Amazon S3 restore speed upgrade you can change the speed of a restoration from the GLACIER storage class to a faster speed while the restoration is in progress. For more information, see Restoring Archived Objects .	November 26, 2018
Restore event notifications (p. 738)	Amazon S3 event notifications now supports initiation and completion events when restoring objects from the GLACIER storage class. For more information, see Event Notifications .	November 26, 2018

PUT directly to the GLACIER storage class (p. 738)	The Amazon S3 PUT operation now supports specifying GLACIER as the storage class when creating objects. Previously, you had to transition objects to the GLACIER storage class from another Amazon S3 storage class. Also, when using cross-Region replication (CRR), you can now specify GLACIER as the storage class for replicated objects. For more information about the GLACIER storage class, see Storage Classes . For more information about specifying the storage class for replicated objects, see Replication Configuration Overview . For more information about the direct PUT to GLACIER REST API changes, see Document History: PUT directly to GLACIER .	November 26, 2018
New storage class (p. 738)	Amazon S3 now offers a new storage class named INTELLIGENT_TIERING that is designed for long-lived data with changing or unknown access patterns. For more information, see Storage Classes .	November 26, 2018
Amazon S3 Object Lock (p. 738)	Amazon S3 now offers Object Lock functionality that provides Write Once Read Many protections for Amazon S3 objects. For more information, see Locking Objects .	November 26, 2018
Amazon S3 Block Public Access (p. 738)	Amazon S3 now includes the ability to block public access to buckets and objects on a per-bucket or account-wide basis. For more information, see Using Amazon S3 Block Public Access .	November 15, 2018
Filtering enhancements in cross-Region replication (CRR) rules (p. 738)	In a CRR rule configuration, you can specify an object filter to choose a subset of objects to apply the rule to. Previously, you could filter only on an object key prefix. In this release, you can filter on an object key prefix, one or more object tags, or both. For more information, see CRR Setup: Replication Configuration Overview .	September 19, 2018

New Amazon S3 Select features (p. 738)	Amazon S3 Select now supports Apache Parquet input, queries on nested JSON objects, and two new Amazon CloudWatch monitoring metrics (<code>SelectScannedBytes</code> and <code>SelectReturnedBytes</code>).	September 5, 2018
Updates now available over RSS (p. 738)	You can now subscribe to an RSS feed to receive notifications about updates to the Amazon Simple Storage Service Developer Guide.	June 19, 2018

Earlier Updates

The following table describes the important changes in each release of the *Amazon Simple Storage Service Developer Guide* before June 19, 2018.

Change	Description	Date
Code examples update	Code examples updated: <ul style="list-style-type: none">• C#—Updated all of the examples to use the task-based asynchronous pattern. For more information, see Amazon Web Services Asynchronous APIs for .NET in the <i>AWS SDK for .NET Developer Guide</i>. Code examples are now compliant with version 3 of the AWS SDK for .NET.• Java—Updated all of the examples to use the client builder model. For more information about the client builder model, see Creating Service Clients.• PHP—Updated all of the examples to use the AWS SDK for PHP 3.0. For more information about the AWS SDK for PHP 3.0, see AWS SDK for PHP.• Ruby—Updated example code so that the examples work with the AWS SDK for Ruby version 3.	April 30, 2018
Amazon S3 now reports GLACIER and ONEZONE_IA storage classes to Amazon CloudWatch Logs storage metrics	In addition to reporting actual bytes, these storage metrics include per-object overhead bytes for applicable storage classes (ONEZONE_IA, STANDARD_IA, and GLACIER): <ul style="list-style-type: none">• For ONEZONE_IA and STANDARD_IA storage class objects, Amazon S3 reports objects smaller than 128 KB as 128 KB. For more information, see Amazon S3 Storage Classes (p. 103).• For GLACIER storage class objects, the storage metrics report the following overheads:<ul style="list-style-type: none">• A 32 KB per-object overhead, charged at GLACIER storage class pricing• An 8 KB per-object overhead, charged at STANDARD storage class pricing For more information, see Transitioning Objects Using Amazon S3 Lifecycle (p. 121) .	April 30, 2018

Change	Description	Date
	For more information about storage metrics, see Monitoring Metrics with Amazon CloudWatch (p. 611) .	
New storage class	Amazon S3 now offers a new storage class, ONEZONE_IA (IA, for infrequent access) for storing objects. For more information, see Amazon S3 Storage Classes (p. 103) .	April 4, 2018
Amazon S3 Select	Amazon S3 now supports retrieving object content based on an SQL expression. For more information, see Selecting Content from Objects (p. 245) .	April 4, 2018
Asia Pacific (Osaka-Local) Region	Amazon S3 is now available in the Asia Pacific (Osaka-Local) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> . Important You can use the Asia Pacific (Osaka-Local) Region only in conjunction with the Asia Pacific (Tokyo) Region. To request access to Asia Pacific (Osaka-Local) Region, contact your sales representative.	February 12, 2018
Amazon S3 inventory creation timestamp	Amazon S3 inventory now includes a timestamp of the date and start time of the creation of the Amazon S3 inventory report. You can use the timestamp to determine changes in your Amazon S3 storage from the start time of when the inventory report was generated.	January 16, 2018
EU (Paris) Region	Amazon S3 is now available in the EU (Paris) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	December 18, 2017
China (Ningxia) Region	Amazon S3 is now available in the China (Ningxia) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	December 11, 2017
Querying archives with SQL	Amazon S3 now supports querying Glacier data archives with SQL. For more information, see Querying Archived Objects (p. 253) .	November 29, 2017
Support for ORC-formatted Amazon S3 inventory files	Amazon S3 now supports the Apache optimized row columnar (ORC) format in addition to comma-separated values (CSV) file format for inventory output files. Also, you can now query Amazon S3 inventory using standard SQL by using Amazon Athena, Amazon Redshift Spectrum, and other tools such as Presto , Apache Hive , and Apache Spark . For more information, see Amazon S3 Inventory (p. 422) .	November 17, 2017
Default encryption for S3 buckets	Amazon S3 default encryption provides a way to set the default encryption behavior for an S3 bucket. You can set default encryption on a bucket so that all objects are encrypted when they are stored in the bucket. The objects are encrypted using server-side encryption with either Amazon S3-managed keys (SSE-S3) or AWS KMS-managed keys (SSE-KMS). For more information, see Amazon S3 Default Encryption for S3 Buckets (p. 66) .	November 06, 2017

Change	Description	Date
Encryption status in Amazon S3 inventory	Amazon S3 now supports including encryption status in Amazon S3 inventory so you can see how your objects are encrypted at rest for compliance auditing or other purposes. You can also configure to encrypt S3 inventory with server-side encryption (SSE) or SSE-KMS so that all inventory files are encrypted accordingly. For more information, see Amazon S3 Inventory (p. 422).	November 06, 2017
Cross-Region replication (CRR) enhancements	Cross-Region replication now supports the following: <ul style="list-style-type: none"> In a cross-account scenario, you can add a CRR configuration to change replica ownership to the AWS account that owns the destination bucket. For more information, see Additional Replication Configuration: Changing the Replica Owner (p. 568). By default, Amazon S3 does not replicate objects in your source bucket that are created using server-side encryption using keys stored in AWS KMS. In your CRR configuration, you can now direct Amazon S3 to replicate these objects. For more information, see Additional Replication Configuration: Replicating Objects Created with Server-Side Encryption (SSE) Using Encryption Keys stored in AWS KMS (p. 570). 	November 06, 2017
EU (London) Region	Amazon S3 is now available in the EU (London) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	December 13, 2016
Canada (Central) Region	Amazon S3 is now available in the Canada (Central) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	December 8, 2016
Object tagging	Amazon S3 now supports object tagging. Object tagging enables you to categorize storage. Object key name prefixes also enable you to categorize storage, object tagging adds another dimension to it. <p>There are added benefits tagging offers. These include:</p> <ul style="list-style-type: none"> Object tags enable fine-grained access control of permissions (for example, you could grant an IAM user permissions to read-only objects with specific tags). Fine-grained control in specifying lifecycle configuration. You can specify tags to select a subset of objects to which lifecycle rule applies. If you have cross-Region replication (CRR) configured, Amazon S3 can replicate the tags. You must grant necessary permission to the IAM role created for Amazon S3 to assume to replicate objects on your behalf. You can also customize CloudWatch metrics and CloudTrail events to display information by specific tag filters. <p>For more information, see Object Tagging (p. 110).</p>	November 29, 2016

Change	Description	Date
Amazon S3 lifecycle now supports tag-based filters	Amazon S3 now supports tag-based filtering in lifecycle configuration. You can now specify lifecycle rules in which you can specify a key prefix, one or more object tags, or a combination of both to select a subset of objects to which the lifecycle rule applies. For more information, see Object Lifecycle Management (p. 119) .	November 29, 2016
CloudWatch request metrics for buckets	Amazon S3 now supports CloudWatch metrics for requests made on buckets. When you enable these metrics for a bucket, the metrics report at 1-minute intervals. You can also configure which objects in a bucket will report these request metrics. For more information, see Monitoring Metrics with Amazon CloudWatch (p. 611) .	November 29, 2016
Amazon S3 Inventory	Amazon S3 now supports storage inventory. Amazon S3 inventory provides a flat-file output of your objects and their corresponding metadata on a daily or weekly basis for an S3 bucket or a shared prefix (that is, objects that have names that begin with a common string). For more information, see Amazon S3 Inventory (p. 422) .	November 29, 2016
Amazon S3 Analytics – Storage Class Analysis	The new Amazon S3 analytics – storage class analysis feature observes data access patterns to help you determine when to transition less frequently accessed STANDARD storage to the STANDARD_IA (IA, for infrequent access) storage class. After storage class analysis observes the infrequent access patterns of a filtered set of data over a period of time, you can use the analysis results to help you improve your lifecycle policies. This feature also includes a detailed daily analysis of your storage usage at the specified bucket, prefix, or tag level that you can export to an S3 bucket. For more information, see Amazon S3 Analytics – Storage Class Analysis (p. 257) in the <i>Amazon Simple Storage Service Developer Guide</i> .	November 29, 2016
New Expedited and Bulk data retrievals when restoring archived objects from Glacier	Amazon S3 now supports Expedited and Bulk data retrievals in addition to Standard retrievals when restoring objects archived to Glacier. For more information, see Restoring Archived Objects (p. 248) .	November 21, 2016
CloudTrail object logging	CloudTrail supports logging Amazon S3 object level API operations such as <code>GetObject</code> , <code>PutObject</code> , and <code>DeleteObject</code> . You can configure your event selectors to log object level API operations. For more information, see Logging Amazon S3 API Calls by Using AWS CloudTrail (p. 621) .	November 21, 2016
US East (Ohio) Region	Amazon S3 is now available in the US East (Ohio) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	October 17, 2016

Change	Description	Date
IPv6 support for Amazon S3 Transfer Acceleration	Amazon S3 now supports Internet Protocol version 6 (IPv6) for Amazon S3 Transfer Acceleration. You can connect to Amazon S3 over IPv6 by using the new dual-stack for Transfer Acceleration endpoint. For more information, see Getting Started with Amazon S3 Transfer Acceleration (p. 74) .	October 6, 2016
IPv6 support	Amazon S3 now supports Internet Protocol version 6 (IPv6). You can access Amazon S3 over IPv6 by using dual-stack endpoints. For more information, see Making Requests to Amazon S3 over IPv6 (p. 12) .	August 11, 2016
Asia Pacific (Mumbai) Region	Amazon S3 is now available in the Asia Pacific (Mumbai) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	June 27, 2016
Amazon S3 Transfer Acceleration	Amazon S3 Transfer Acceleration enables fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration takes advantage of Amazon CloudFront's globally distributed edge locations. For more information, see Amazon S3 Transfer Acceleration (p. 73) .	April 19, 2016
Lifecycle support to remove expired object delete markers	Lifecycle configuration <code>Expiration</code> action now allows you to direct Amazon S3 to remove expired object delete markers in a versioned bucket. For more information, see Elements to Describe Lifecycle Actions (p. 130) .	March 16, 2016

Change	Description	Date
Bucket lifecycle configuration now supports action to abort incomplete multipart uploads	<p>Bucket lifecycle configuration now supports the <code>AbortIncompleteMultipartUpload</code> action that you can use to direct Amazon S3 to abort multipart uploads that don't complete within a specified number of days after being initiated. When a multipart upload becomes eligible for an abort operation, Amazon S3 deletes any uploaded parts and aborts the multipart upload.</p> <p>For conceptual information, see the following topics in the <i>Amazon Simple Storage Service Developer Guide</i>:</p> <ul style="list-style-type: none"> Aborting Incomplete Multipart Uploads Using a Bucket Lifecycle Policy (p. 177) Elements to Describe Lifecycle Actions (p. 130) <p>The following API operations have been updated to support the new action:</p> <ul style="list-style-type: none"> PUT Bucket lifecycle – The XML configuration now allows you to specify the <code>AbortIncompleteMultipartUpload</code> action in a lifecycle configuration rule. List Parts and Initiate Multipart Upload – Both of these API operations now return two additional response headers (<code>x-amz-abort-date</code>, and <code>x-amz-abort-rule-id</code>) if the bucket has a lifecycle rule that specifies the <code>AbortIncompleteMultipartUpload</code> action. These headers in the response indicate when the initiated multipart upload will become eligible for abort operation and which lifecycle rule is applicable. 	March 16, 2016
Asia Pacific (Seoul) Region	Amazon S3 is now available in the Asia Pacific (Seoul) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	January 6, 2016
New condition key and a Multipart Upload change	<p>IAM policies now support an Amazon S3 <code>s3:x-amz-storage-class</code> condition key. For more information, see Specifying Conditions in a Policy (p. 350).</p> <p>You no longer need to be the initiator of a multipart upload to upload parts and complete the upload. For more information, see Multipart Upload API and Permissions (p. 179).</p>	December 14, 2015
Renamed the US Standard Region	Changed the Region name string from "US Standard" to "US East (N. Virginia)." This is only a Region name update, there is no change in the functionality.	December 11, 2015

Change	Description	Date
New storage class	<p>Amazon S3 now offers a new storage class, STANDARD_IA (IA, for infrequent access) for storing objects. This storage class is optimized for long-lived and less frequently accessed data. For more information, see Amazon S3 Storage Classes (p. 103).</p> <p>Lifecycle configuration feature updates now allow you to transition objects to the STANDARD_IA storage class. For more information, see Object Lifecycle Management (p. 119).</p> <p>Previously, the cross-Region replication feature used the storage class of the source object for object replicas. Now, when you configure cross-Region replication you can specify a storage class for the object replica created in the destination bucket. For more information, see Replication (p. 551).</p>	September 16, 2015
AWS CloudTrail integration	New AWS CloudTrail integration allows you to record Amazon S3 API activity in your S3 bucket. You can use CloudTrail to track S3 bucket creations or deletions, access control modifications, or lifecycle policy changes. For more information, see Logging Amazon S3 API Calls by Using AWS CloudTrail (p. 621) .	September 1, 2015
Bucket limit increase	Amazon S3 now supports bucket limit increases. By default, customers can create up to 100 buckets in their AWS account. Customers who need additional buckets can increase that limit by submitting a service limit increase. For information about how to increase your bucket limit, go to AWS Service Limits in the <i>AWS General Reference</i> . For more information, see Creating a Bucket (p. 53) and Bucket Restrictions and Limitations (p. 58) .	August 4, 2015
Consistency model update	Amazon S3 now supports read-after-write consistency for new objects added to Amazon S3 in the US East (N. Virginia) Region. Prior to this update, all Regions except US East (N. Virginia) Region supported read-after-write consistency for new objects uploaded to Amazon S3. With this enhancement, Amazon S3 now supports read-after-write consistency in all Regions for new objects added to Amazon S3. Read-after-write consistency allows you to retrieve objects immediately after creation in Amazon S3. For more information, see Regions (p. 4) .	August 4, 2015
Event notifications	Amazon S3 event notifications have been updated to add notifications when objects are deleted and to add filtering on object names with prefix and suffix matching. For more information, see Configuring Amazon S3 Event Notifications (p. 530) .	July 28, 2015
Amazon CloudWatch integration	New Amazon CloudWatch integration allows you to monitor and set alarms on your Amazon S3 usage through CloudWatch metrics for Amazon S3. Supported metrics include total bytes for standard storage, total bytes for reduced-redundancy storage, and total number of objects for a given S3 bucket. For more information, see Monitoring Metrics with Amazon CloudWatch (p. 611) .	July 28, 2015

Change	Description	Date
Support for deleting and emptying non-empty buckets	Amazon S3 now supports deleting and emptying non-empty buckets. For more information, see Deleting or Emptying a Bucket (p. 62).	July 16, 2015
Bucket policies for Amazon VPC endpoints	Amazon S3 has added support for bucket policies for Amazon Virtual Private Cloud (Amazon VPC) endpoints. You can use S3 bucket policies to control access to buckets from specific Amazon VPC endpoints, or specific VPCs. VPC endpoints are easy to configure, are highly reliable, and provide a secure connection to Amazon S3 without requiring a gateway or a NAT instance. For more information, see Example Bucket Policies for VPC Endpoints for Amazon S3 (p. 378).	April 29, 2015
Event notifications	Amazon S3 event notifications have been updated to support the switch to resource-based permissions for AWS Lambda functions. For more information, see Configuring Amazon S3 Event Notifications (p. 530).	April 9, 2015
Cross-Region replication	Amazon S3 now supports cross-Region replication. Cross-Region replication is the automatic, asynchronous copying of objects across buckets in different AWS Regions. For more information, see Replication (p. 551).	March 24, 2015
Event notifications	Amazon S3 now supports new event types and destinations in a bucket notification configuration. Prior to this release, Amazon S3 supported only the <i>s3:ReducedRedundancyLostObject</i> event type and an Amazon SNS topic as the destination. For more information about the new event types, see Configuring Amazon S3 Event Notifications (p. 530).	November 13, 2014
Server-side encryption with customer-provided encryption keys	<p>Server-side encryption with AWS Key Management Service (AWS KMS)</p> <p>Amazon S3 now supports server-side encryption using AWS Key Management Service. This feature allows you to manage the envelope key through AWS KMS, and Amazon S3 calls AWS KMS to access the envelope key within the permissions you set.</p> <p>For more information about server-side encryption with AWS KMS, see Protecting Data Using Server-Side Encryption with AWS Key Management Service.</p>	November 12, 2014
EU (Frankfurt) Region	Amazon S3 is now available in the EU (Frankfurt) Region.	October 23, 2014

Change	Description	Date
Server-side encryption with customer-provided encryption keys	<p>Amazon S3 now supports server-side encryption using customer-provided encryption keys (SSE-C). Server-side encryption enables you to request Amazon S3 to encrypt your data at rest. When using SSE-C, Amazon S3 encrypts your objects with the custom encryption keys that you provide. Since Amazon S3 performs the encryption for you, you get the benefits of using your own encryption keys without the cost of writing or executing your own encryption code.</p> <p>For more information about SSE-C, see Server-Side Encryption (Using Customer-Provided Encryption Keys).</p>	June 12, 2014
Lifecycle support for versioning	<p>Prior to this release, lifecycle configuration was supported only on nonversioned buckets. Now you can configure lifecycle on both nonversioned and versioning-enabled buckets. For more information, see Object Lifecycle Management (p. 119).</p>	May 20, 2014
Access control topics revised	<p>Revised Amazon S3 access control documentation. For more information, see Identity and Access Management in Amazon S3 (p. 301).</p>	April 15, 2014
Server access logging topic revised	<p>Revised server access logging documentation. For more information, see Amazon S3 Server Access Logging (p. 647).</p>	November 26, 2013
.NET SDK samples updated to version 2.0	<p>.NET SDK samples in this guide are now compliant to version 2.0.</p>	November 26, 2013
SOAP Support Over HTTP Deprecated	<p>SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.</p>	September 20, 2013
IAM policy variable support	<p>The IAM access policy language now supports variables. When a policy is evaluated, any policy variables are replaced with values that are supplied by context-based information from the authenticated user's session. You can use policy variables to define general purpose policies without explicitly listing all the components of the policy. For more information about policy variables, see IAM Policy Variables Overview in the <i>IAM User Guide</i>.</p> <p>For examples of policy variables in Amazon S3, see User Policy Examples (p. 380).</p>	April 3, 2013
Console support for Requester Pays	<p>You can now configure your bucket for Requester Pays by using the Amazon S3 console. For more information, see Configure Requester Pays by Using the Amazon S3 Console (p. 81).</p>	December 31, 2012

Change	Description	Date
Root domain support for website hosting	Amazon S3 now supports hosting static websites at the root domain. Visitors to your website can access your site from their browser without specifying "www" in the web address (e.g., "example.com"). Many customers already host static websites on Amazon S3 that are accessible from a "www" subdomain (e.g., "www.example.com"). Previously, to support root domain access, you needed to run your own web server to proxy root domain requests from browsers to your website on Amazon S3. Running a web server to proxy requests introduces additional costs, operational burden, and another potential point of failure. Now, you can take advantage of the high availability and durability of Amazon S3 for both "www" and root domain addresses. For more information, see Hosting a Static Website on Amazon S3 (p. 503).	December 27, 2012
Console revision	Amazon S3 console has been updated. The documentation topics that refer to the console have been revised accordingly.	December 14, 2012
Support for Archiving Data to Glacier	Amazon S3 now supports a storage option that enables you to utilize Glacier's low-cost storage service for data archival. To archive objects, you define archival rules identifying objects and a timeline when you want Amazon S3 to archive these objects to Glacier. You can easily set the rules on a bucket using the Amazon S3 console or programmatically using the Amazon S3 API or AWS SDKs. For more information, see Object Lifecycle Management (p. 119).	November 13, 2012
Support for Website Page Redirects	For a bucket that is configured as a website, Amazon S3 now supports redirecting a request for an object to another object in the same bucket or to an external URL. For more information, see (Optional) Configuring a Webpage Redirect (p. 510). For information about hosting websites, see Hosting a Static Website on Amazon S3 (p. 503).	October 4, 2012
Support for Cross-Origin Resource Sharing (CORS)	Amazon S3 now supports Cross-Origin Resource Sharing (CORS). CORS defines a way in which client web applications that are loaded in one domain can interact with or access resources in a different domain. With CORS support in Amazon S3, you can build rich client-side web applications on top of Amazon S3 and selectively allow cross-domain access to your Amazon S3 resources. For more information, see Cross-Origin Resource Sharing (CORS) (p. 151).	August 31, 2012
Support for Cost Allocation Tags	Amazon S3 now supports cost allocation tagging, which allows you to label S3 buckets so you can more easily track their cost against projects or other criteria. For more information about using tagging for buckets, see Using Cost Allocation S3 Bucket Tags (p. 95).	August 21, 2012

Change	Description	Date
Support for MFA-protected API access in bucket policies	<p>Amazon S3 now supports MFA-protected API access, a feature that can enforce AWS Multi-Factor Authentication for an extra level of security when accessing your Amazon S3 resources. It is a security feature that requires users to prove physical possession of an MFA device by providing a valid MFA code. For more information, go to AWS Multi-Factor Authentication. You can now require MFA authentication for any requests to access your Amazon S3 resources.</p> <p>To enforce MFA authentication, Amazon S3 now supports the <code>aws:MultiFactorAuthAge</code> key in a bucket policy. For an example bucket policy, see Adding a Bucket Policy to Require MFA (p. 375).</p>	July 10, 2012
Object Expiration support	You can use Object Expiration to schedule automatic removal of data after a configured time period. You set object expiration by adding lifecycle configuration to a bucket.	27 December 2011
New Region supported	Amazon S3 now supports the South America (São Paulo) Region. For more information, see Accessing a Bucket (p. 55).	December 14, 2011
Multi-Object Delete	Amazon S3 now supports Multi-Object Delete API that enables you to delete multiple objects in a single request. With this feature, you can remove large numbers of objects from Amazon S3 more quickly than using multiple individual DELETE requests. For more information, see Deleting Objects (p. 227).	December 7, 2011
New Region supported	Amazon S3 now supports the US West (Oregon) Region. For more information, see Buckets and Regions (p. 55).	November 8, 2011
Documentation Update	Documentation bug fixes.	November 8, 2011
Documentation Update	<p>In addition to documentation bug fixes, this release includes the following enhancements:</p> <ul style="list-style-type: none"> • New server-side encryption sections using the AWS SDK for PHP (see Specifying Server-Side Encryption Using the AWS SDK for PHP (p. 275)) and the AWS SDK for Ruby (see Specifying Server-Side Encryption Using the AWS SDK for Ruby (p. 277)). • New section on creating and testing Ruby samples (see Using the AWS SDK for Ruby - Version 3 (p. 679)). 	October 17, 2011
Server-side encryption support	Amazon S3 now supports server-side encryption. It enables you to request Amazon S3 to encrypt your data at rest, that is, encrypt your object data when Amazon S3 writes your data to disks in its data centers. In addition to REST API updates, the AWS SDK for Java and .NET provide necessary functionality to request server-side encryption. You can also request server-side encryption when uploading objects using AWS Management Console. To learn more about data encryption, go to Using Data Encryption .	October 4, 2011

Change	Description	Date
Documentation Update	<p>In addition to documentation bug fixes, this release includes the following enhancements:</p> <ul style="list-style-type: none"> Added Ruby and PHP samples to the Making Requests (p. 10) section. Added sections describing how to generate and use presigned URLs. For more information, see Share an Object with Others (p. 167) and Uploading Objects Using Presigned URLs (p. 206). Updated an existing section to introduce AWS Explorers for Eclipse and Visual Studio. For more information, see Using the AWS SDKs, CLI, and Explorers (p. 669). 	September 22, 2011
Support for sending requests using temporary security credentials	<p>In addition to using your AWS account and IAM user security credentials to send authenticated requests to Amazon S3, you can now send requests using temporary security credentials you obtain from AWS Identity and Access Management (IAM). You can use the AWS Security Token Service API or the AWS SDK wrapper libraries to request these temporary security credentials from IAM. You can request these temporary security credentials for your own use or hand them out to federated users and applications. This feature enables you to manage your users outside AWS and provide them with temporary security credentials to access your AWS resources.</p> <p>For more information, see Making Requests (p. 10).</p> <p>For more information about IAM support for temporary security credentials, see Temporary Security Credentials in the <i>IAM User Guide</i>.</p>	August 3, 2011
Multipart Upload API extended to enable copying objects up to 5 TB	<p>Prior to this release, Amazon S3 API supported copying objects of up to 5 GB in size. To enable copying objects larger than 5 GB, Amazon S3 now extends the multipart upload API with a new operation, <code>UploadPart (Copy)</code>. You can use this multipart upload operation to copy objects up to 5 TB in size. For more information, see Copying Objects (p. 210).</p> <p>For conceptual information about multipart upload API, see Uploading Objects Using Multipart Upload API (p. 175).</p>	June 21, 2011
SOAP API calls over HTTP disabled	<p>To increase security, SOAP API calls over HTTP are disabled. Authenticated and anonymous SOAP requests must be sent to Amazon S3 using SSL.</p>	June 6, 2011

Change	Description	Date
IAM enables cross-account delegation	<p>Previously, to access an Amazon S3 resource, an IAM user needed permissions from both the parent AWS account and the Amazon S3 resource owner. With cross-account access, the IAM user now only needs permission from the owner account. That is, If a resource owner grants access to an AWS account, the AWS account can now grant its IAM users access to these resources.</p> <p>For more information, see Creating a Role to Delegate Permissions to an IAM User in the <i>IAM User Guide</i>.</p> <p>For more information on specifying principals in a bucket policy, see Specifying a Principal in a Policy (p. 343).</p>	June 6, 2011
New link	This service's endpoint information is now located in the AWS General Reference. For more information, go to Regions and Endpoints in AWS General Reference .	March 1, 2011
Support for hosting static websites in Amazon S3	Amazon S3 introduces enhanced support for hosting static websites. This includes support for index documents and custom error documents. When using these features, requests to the root of your bucket or a subfolder (e.g., http://mywebsite.com/subfolder) returns your index document instead of the list of objects in your bucket. If an error is encountered, Amazon S3 returns your custom error message instead of an Amazon S3 error message. For more information, see Hosting a Static Website on Amazon S3 (p. 503) .	February 17, 2011
Response Header API Support	The GET Object REST API now allows you to change the response headers of the REST GET Object request for each request. That is, you can alter object metadata in the response, without altering the object itself. For more information, see Getting Objects (p. 161) .	January 14, 2011
Large object support	Amazon S3 has increased the maximum size of an object you can store in an S3 bucket from 5 GB to 5 TB. If you are using the REST API you can upload objects of up to 5 GB size in a single PUT operation. For larger objects, you must use the Multipart Upload REST API to upload objects in parts. For more information, see Uploading Objects Using Multipart Upload API (p. 175) .	December 9, 2010
Multipart upload	Multipart upload enables faster, more flexible uploads into Amazon S3. It allows you to upload a single object as a set of parts. For more information, see Uploading Objects Using Multipart Upload API (p. 175) .	November 10, 2010
Canonical ID support in bucket policies	You can now specify canonical IDs in bucket policies. For more information, see Access Policy Language Overview (p. 341)	September 17, 2010
Amazon S3 works with IAM	This service now integrates with AWS Identity and Access Management (IAM). For more information, go to AWS Services That Work with IAM in the <i>IAM User Guide</i> .	September 2, 2010

Change	Description	Date
Notifications	The Amazon S3 notifications feature enables you to configure a bucket so that Amazon S3 publishes a message to an Amazon Simple Notification Service (Amazon SNS) topic when Amazon S3 detects a key event on a bucket. For more information, see Setting Up Notification of Bucket Events (p. 530) .	July 14, 2010
Bucket policies	Bucket policies is an access management system you use to set access permissions across buckets, objects, and sets of objects. This functionality supplements and in many cases replaces access control lists. For more information, see Using Bucket Policies and User Policies (p. 341) .	July 6, 2010
Path-style syntax available in all Regions	Amazon S3 now supports the path-style syntax for any bucket in the US Classic Region, or if the bucket is in the same Region as the endpoint of the request. For more information, see Virtual Hosting (p. 45) .	June 9, 2010
New endpoint for EU (Ireland)	Amazon S3 now provides an endpoint for EU (Ireland): <code>http://s3-eu-west-1.amazonaws.com</code> .	June 9, 2010
Console	You can now use Amazon S3 through the AWS Management Console. You can read about all of the Amazon S3 functionality in the console in the Amazon Simple Storage Service Console User Guide.	June 9, 2010
Reduced Redundancy	Amazon S3 now enables you to reduce your storage costs by storing objects in Amazon S3 with reduced redundancy. For more information, see Reduced Redundancy Storage (p. 6) .	May 12, 2010
New Region supported	Amazon S3 now supports the Asia Pacific (Singapore) Region. For more information, see Buckets and Regions (p. 55) .	April 28, 2010
Object Versioning	This release introduces object versioning. All objects now can have a key and a version. If you enable versioning for a bucket, Amazon S3 gives all objects added to a bucket a unique version ID. This feature enables you to recover from unintended overwrites and deletions. For more information, see Versioning (p. 7) and Using Versioning (p. 432) .	February 8, 2010
New Region supported	Amazon S3 now supports the US West (N. California) Region. The new endpoint for requests to this Region is <code>s3-us-west-1.amazonaws.com</code> . For more information, see Buckets and Regions (p. 55) .	December 2, 2009
AWS SDK for .NET	AWS now provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using .NET language-specific API operations instead of REST or SOAP. These libraries provide basic functions (not included in the REST or SOAP APIs), such as request authentication, request retries, and error handling so that it's easier to get started. For more information about language-specific libraries and resources, see Using the AWS SDKs, CLI, and Explorers (p. 669) .	November 11, 2009

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.