Amazon Simple Storage Service

Developer Guide

API Version 2006-03-01

Amazon Web Services, LLC

Amazon Simple Storage Service: Developer Guide

Amazon Web Services, LLC

Welcome to Amazon S3

This is the *Amazon Simple Storage Service (Amazon S3) Developer Guide*. It explains Amazon S3 core concepts of buckets, objects and how to work with these resources using the available APIs. It describes how you send requests to create buckets, store and retrieve your objects, and manage permissions to your resources. The guide describes both the access control and the authentication process. The access control defines who can access objects and buckets within Amazon S3, and the type of access (e.g., READ, WRITE, and so on). The authentication process verifies identity of a user trying to access Amazon Web Services (AWS).

Amazon Simple Storage Service (Amazon S3) is a web service that enables you to store data in the cloud. You can then download the data or use the data with other AWS services, such as Amazon Elastic Cloud Computer (see Amazon Elastic Compute Cloud (Amazon EC2)).

How Do I...?

Information	Relevant Sections
General product overview and pricing	Amazon Simple Storage Service (Amazon S3)
Get a quick hands-on introduction to Amazon S3	Amazon Simple Storage Service (Amazon S3) *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. 81)
How do I work with objects	Working with Amazon S3 Objects (p. 100)
How do I make requests	Making Requests (p. 11)
How do I manage access to my resources	Access Control (p. 277)

Introduction to Amazon S3

Topics

This introduction to Amazon S3 is intended to give you 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.

Overview of Amazon S3

Amazon S3 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 can be used 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 to developers.

Advantages to Amazon S3

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

- **Create Buckets—**Create and name a bucket that stores data
  Buckets are the fundamental container in Amazon S3 for data storage.
- **Store data in Buckets—**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.
- **Download data—**Download your data or enable others to

Download your data any time 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 to ensure that data is kept secure from unauthorized access.
- **Standard interfaces—**Use standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.

Amazon S3 Concepts

Topics

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

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, and they serve as the unit of aggregation for usage reporting.

You can configure buckets so that they are created in a specific Region. For more information, see Buckets and Regions (p. 83). 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 Versioning (p. 355).

For more information about buckets, see Working with Amazon S3 Buckets (p. 81).

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. The developer 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 Versioning (p. 355).

Keys

A key is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. Because the combination of a bucket, key, and version ID uniquely identify each object, Amazon S3 can be thought of 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.wsdl, "doc" is the name of the bucket and "2006-03-01/AmazonS3.wsdl" is the key.

Regions

You can choose the geographical Region where Amazon S3 will store the buckets you create. You might choose a Region to optimize latency, minimize costs, or address regulatory requirements. Amazon S3 currently supports the following Regions:

- **US Standard—**Uses Amazon S3 servers in the United States
  This is the default Region. The US Standard Region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps. To use this region, select US Standard as the region when creating a bucket in the console. The US Standard Region provides eventual consistency for all requests.
- **US West (Oregon) Region—**Uses Amazon S3 servers in Oregon
  To use this Region, choose Oregon as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the US West (Oregon) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **US West (Northern California) Region—**Uses Amazon S3 servers in Northern California
  To use this Region, choose Northern California as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the US West (Northern California) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **EU (Ireland) Region—**Uses Amazon S3 servers in Ireland
  To use this Region, choose Ireland as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the EU (Ireland) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **Asia Pacific (Singapore) Region—**Uses Amazon S3 servers in Singapore
  To use this Region, choose Singapore as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the Asia Pacific (Singapore) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **Asia Pacific (Sydney) Region**—Uses Amazon S3 servers in Sydney
  To use this Region, choose Sydney as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the Asia Pacific (Sydney) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **Asia Pacific (Tokyo) Region—**Uses Amazon S3 servers in Tokyo
  To use this Region, choose Tokyo as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the Asia Pacific (Tokyo) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **South America (Sao Paulo) Region—**Uses Amazon S3 servers in Sao Paulo
  To use this Region, choose Sao Paulo as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the South America (Sao Paulo) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.

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.

Amazon S3 Data Consistency Model

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 will never write corrupted or partial data.

Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. After a "success" is returned, your data is safely stored. However, information about the changes might not immediately replicate across Amazon S3 and you might observe the following behaviors:

- A process writes a new object to Amazon S3 and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might report "key does not exist."
- 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 attempts to read it. Until the change is fully propagated, Amazon S3 might return the prior data.
- A process deletes an existing object and immediately attempts 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.

The US Standard Region provides eventual consistency for all requests. All other regions provide read-after-write consistency for PUTS of new objects and eventual consistency for overwrite PUTS and DELETES.

Note
Amazon S3 does not currently support object locking. If two puts are simultaneously made to the same key, the put with the latest time stamp 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.

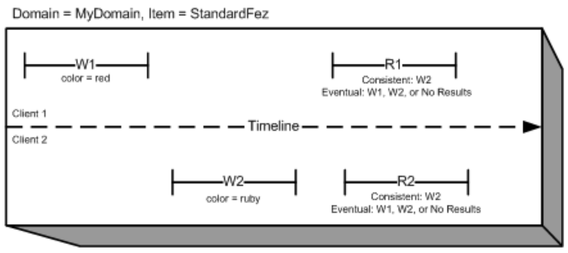
The following table describes the characteristics of eventually consistent read and consistent read.

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

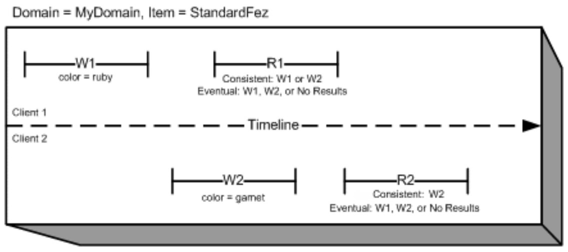
This section provides examples of eventually consistent and consistent read requests when multiple clients are writing to the same items.

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`, `color = ruby`, or no results, depending on the amount of time that has elapsed.

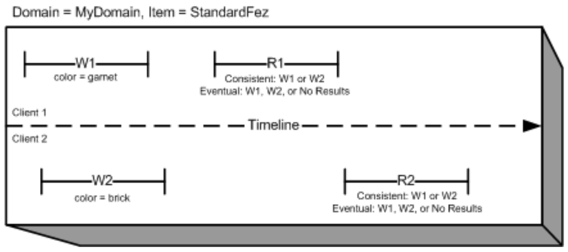

Domain = MyDomain, Item = StandardFez
W1
color = red
R1
Consistent: W2
Eventual: W1, W2, or No Results
Client 1
Timeline
Client 2
W2
color = ruby
R2
Consistent: W2
Eventual: W1, W2, or No Results

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`, `color = garnet`, or no results depending on the amount of time that has elapsed.


Domain = MyDomain, Item = StandardFez
W1
color = ruby
R1
Consistent: W1 or W2
Eventual: W1, W2, or No Results
Client 1
Timeline
Client 2
W2
color = garnet
R2
Consistent: W2
Eventual: W1, W2, or No Results

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.


Domain = MyDomain, Item = StandardFez
W1
color = garnet
R1
Consistent: W1 or W2
Eventual: W1, W2, or No Results
Client 1
Timeline
Client 2
W2
color = brick
R2
Consistent: W1 or W2
Eventual: W1, W2, or No Results

Features

Topics

This section describes important Amazon S3 features.

Reduced Redundancy Storage

Customers can store their data using the Amazon S3 Reduced Redundancy Storage (RRS) option. RRS enables customers to reduce their costs by storing non-critical, reproducible data at lower levels of redundancy than Amazon S3's standard storage. RRS provides a cost-effective, highly available solution for distributing or sharing content that is durably stored elsewhere, or for storing thumbnails, transcoded media, or other processed data that can be easily reproduced. The RRS option stores objects on multiple devices across multiple facilities, providing 400 times the durability of a typical disk drive, but does not replicate objects as many times as standard Amazon S3 storage, and thus is even more cost effective.

RRS provides 99.99% durability of objects over a given year. This durability level corresponds to an average expected loss of 0.01% of objects annually.

We charge less for using RRS than for standard Amazon S3 storage. For pricing information, go to the Amazon S3 detail page.

For more information, see Using Reduced Redundancy Storage (p. 353).

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 (e.g., IP address). The policies are expressed in our *access policy language* and enable centralized management of permissions. The permissions attached to a bucket apply to all of the objects in that bucket..

Individuals as well as 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 Amazon resources 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
- From an account's custom application (as identified by a user agent string)

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

Unlike ACLs (described below) that can only add (grant) permissions 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, so that an account can 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:

- 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 (p. 306).

AWS Identity and Access Management

Amazon S3 integrates with AWS Identity and Access Management (IAM), a service that lets your organization do the following:

- Create users and groups under your organization's AWS account
- Easily share your AWS account resources between the users in the account
- Assign unique security credentials to each user
- Granularly control users access to services and resources
- Get a single AWS bill for all users under the AWS account

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 general information about IAM, go to:

- Identity and Access Management (IAM)
- AWS Identity and Access Management Getting Started Guide
- Using AWS Identity and Access Management

For specific information about how you can control User access to Amazon S3, go to Integrating with Other AWS Products in *Using AWS Identity and Access Management*.

Access Control Lists

For more information, see Using ACLs (p. 323)

Versioning

For more information, see Object Versioning (p. 103)

Operations

Amazon S3 offers APIs in REST and SOAP. Following are the most common operations 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 choose to download the data via HTTP or BitTorrent.
- **Deleting an Object—**Delete some of your data.
- **Listing Keys—**List the keys contained in one of your buckets.
  You can filter the key list based on a prefix.

Details on this and all other functionality are described in detail later in this guide.

Amazon S3 Application Programming Interfaces (API)

The Amazon S3 architecture is designed to be programming language-neutral, using our 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.

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

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 (go to 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 Amazon's infrastructure.

Before storing anything in Amazon S3, you need to register with the service and provide a payment instrument that will be charged at the end of each month. There are no set-up fees to begin using the service. At the end of the month, your payment instrument is automatically charged for that month's usage.

For information about paying for Amazon S3 storage, go to the AWS Resource Center.

Related Amazon Web Services

Once we load your data into AWS you can use it with all AWS services. The following services are the ones you might use most frequently:

- **Amazon ElasticCompute Cloud—**This web service provides virtual compute resources in the cloud. For more information, go to Amazon ElasticCompute Cloud.
- **Amazon Elastic MapReduce—**This web service enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.
  It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3). For more information, go to Amazon Elastic MapReduce.
- **Amazon Import/Export—**This service enables you to mail a storage device, such as a RAID drive, to Amazon so that we can upload your (terabytes) of data onto Amazon S3. For more information, go to *AWS Import/Export Developer Guide*.

Making Requests

Topics

Amazon S3 is a REST service. You can send requests to Amazon S3 using the REST API or the AWS SDK (see STS 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 identify of the requester trying to access 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 security credentials.

If you are using the AWS SDK, the libraries compute the signature from the credentials 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.

Types of Security Credentials

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

AWS Account Security Credentials

When you create an AWS Account, AWS assigns the following credentials to you:

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

The Access Key ID uniquely identifies an AWS Account. You can use these AWS Account credentials to send authenticated requests to Amazon S3.

IAM User Security Credentials

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 credentials reduces security, and creating individual AWS Accounts for each employee might not be practical. Also, you cannot easily share the 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) service to create users under your AWS Account with their own security credentials (Access Key ID and Secret Access Key) 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 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 AWS Identity and Access Management (IAM).

Temporary Security Credentials

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

IAM provides the AWS Security Token Service (STS) 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 at most for the duration you specify when you request them. You use the Access Key ID and Secret Access Key credentials the same way you use them when sending requests using your AWS Account or IAM user credentials. 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. 48). For information about sending requests using AWS SDKs, see Making Requests using AWS SDKs (p. 14).

For more information about IAM support for temporary security credentials, go to Granting Temporary Access to Your AWS Resources.

For added security, you can require MFA authentication when accessing your Amazon S3 resources. You can do this by configuring a bucket policy (see Adding Bucket Policy to Require MFA

Authentication (p. 311)). After you require MFA authentication to access your S3 resources, the only way you can access these resources is by providing temp credentials that are created with an MFA key. For more information, go to AWS Multi-Factor Authentication detail page and Configuring MFA-Protected API Access in the *AWS Identity and Access Management Using IAM* guide.

Viewing Your AWS Security Credentials

AWS Accounts and IAM users have Access Key ID and Secret Access Key security credentials.

Viewing Your AWS Account Security Credentials

Your Access Key ID and Secret Access Key are displayed when you create your AWS account. They are not e-mailed to you. If you need to see them again, you can view them at any time from your AWS account.

1. Go to the Amazon Web Services web site at http://aws.amazon.com.
2. Click the **Account** tab.
3. Click **Security Credentials.**
   If you have not already signed in, Sign In or Create an AWS Account page displays. You must sign in before you can see your AWS account security credentials.

Your access credentials, Access Key ID and Secret Access Key, are displayed on the resulting Security Credentials page.

Viewing Your IAM User Security Credentials

An AWS Account owner creates IAM users under its account. When the owner creates an IAM user under its account, the account owner gets to download the IAM user credentials (Access Key ID and Secret Access Key). The account owner can then provide these credentials for others to use. The IAM user can then use these credentials to send requests to Amazon S3.

Each IAM user needs a password to connect to the AWS Management Console. For more information, go to How does a user sign in?.

Viewing Your Temporary Security Credentials

You request temporary security credentials programmatically. You can view them only when they are created.

Request Endpoints

You send REST requests to the service's predefined endpoint. The *Amazon Web Services General Reference* lists AWS Services and the corresponding endpoints. For information about Amazon S3 regions and endpoints, go to Regions and Endpoints.

Making Requests using AWS SDKs

Topics

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.

Making Requests Using AWS Account or IAM User Credentials

Topics

You can use an 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, .NET, and PHP.

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

The following tasks guide you through using the Java classes to send authenticated requests using your AWS Account credentials or IAM user credentials.

Making Requests Using Your AWS Account or IAM user Credentials

1	Create an instance of the `AmazonS3Client` class by providing your AWS Account or IAM user credentials (Access Key ID, Secret Access Key).
2	Execute one of the `AmazonS3Client` methods to send requests to Amazon S3. The client generates the necessary signature value from your credentials and includes it in the request it sends to Amazon S3.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                 myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);

// Send sample request (list objects in a given bucket).
ObjectListing objectListing = s3client.listObjects(new
     ListObjectsRequest().withBucketName(bucketName));
```

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. 100) and Working with Amazon S3 Buckets (p. 81). You can test these examples using your AWS Account or IAM user credentials.

For example, to list all the object keys in your bucket, see Listing Keys Using the AWS SDK for Java (p. 228).

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

The following tasks guide you through using the .NET classes to send authenticated requests using your AWS account or IAM user credentials.

Making Requests Using Your AWS Account or IAM user Credentials

1	Create an instance of the `AmazonS3Client` class by providing your AWS Account or IAM user credentials (Access Key ID, Secret Access Key).
2	Execute one of the `AmazonS3Client` methods to send requests to Amazon S3. The client generates the necessary signature from your credentials and includes it in the request it sends to Amazon S3.

The following C# code sample demonstrates the preceding tasks.

```
BasicAWSCredentials basicCredentials =
               new BasicAWSCredentials("*** Access Key ID ***",
                                       "*** Security Access Key ***");
AmazonS3Client s3Client = new AmazonS3Client(basicCredentials);

// Send sample request (for example, list buckets).
var response = s3Client.ListBuckets();
```

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. 100) and Working with Amazon S3 Buckets (p. 81). 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. 231).

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

The following tasks guide you through using the PHP classes to send authenticated requests using your AWS account or IAM user credentials.

Making Requests Using Your AWS Account or IAM user Credentials

1	Create an instance of the `AmazonS3` class by providing your AWS Account or IAM user credentials.

2	Execute one of the `AmazonS3` methods to send requests to Amazon S3. The client API generates the necessary signature from your credentials and include it in the request it sends to Amazon S3.

The following PHP code sample demonstrates the preceding tasks and illustrate how the `AmazonS3` client sends a request to list all object keys in a bucket.

```
// Instantiate the class.
$s3 = new AmazonS3();
// Use the high-level API.
$response = $s3->get_object_list($bucket);
```

Note
You can create the `AmazonS3` 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. 100) and Working with Amazon S3 Buckets (p. 81). You can test these examples using your AWS Account or IAM user credentials.

For example, to list all the object keys in your bucket, see Listing Keys Using the AWS SDK for PHP (p. 234).

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

The following tasks guide you through using the AWS SDK for Ruby to send authenticated requests using your AWS Account credentials or IAM user credentials.

Making Requests Using Your AWS Account or IAM user Credentials

1	Create an instance of the `AWS::S3` class by providing your AWS Account or IAM user credentials (Access Key ID, Secret Access Key).
2	Make a request to Amazon S3 by enumerating objects in a bucket using the `buckets` method of `AWS::S3`. The client generates the necessary signature value from your credentials and includes it in the request it sends to Amazon S3.

The following Ruby code sample demonstrates the preceding tasks.

```
# Get an instance of the S3 interface using the specified credentials configur
ation.
s3 = AWS::S3.new(
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
puts bucket
```

Note
You can create the `AWS:S3` 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. 100) and Working with Amazon S3 Buckets (p. 81). You can test these examples using your AWS Account or IAM user credentials.

Making Requests Using IAM User Temporary Credentials

Topics

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. 11)) using AWS SDK for Java 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 duration, between 1 and 36 hours, when requesting the temporary security credentials.

Making Requests Using IAM User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client `AWSSecurityTokenServiceClient` by providing your credentials.
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 `BasicSessionCredentials` object so you can provide the credentials to your Amazon S3 client.
4	Create an instance of the `AmazonS3Client` class by passing in the temporary security credentials. You send the requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following Java code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AWSCredentials credentials =
                 new BasicAWSCredentials("*** Access Key ID ***",
                                         "*** Secret Key ***");
AWSSecurityTokenServiceClient stsClient =
                        new AWSSecurityTokenServiceClient(credentials);
```

```
//
// Manually start a session.
GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();
// Following duration can be set only if temporary credentials are requested
by an IAM user.
getSessionTokenRequest.setDurationSeconds(7200);

GetSessionTokenResult sessionTokenResult =
                           stsClient.getSessionToken(getSessionTokenRequest);
Credentials sessionCredentials = sessionTokenResult.getCredentials();

// 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());

// The following will be part of your less trusted code. You provide temporary
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

// Test. For example, get object keys in a bucket.
ObjectListing objects = s3.listObjects(bucketName);
```

Example

Note

If you obtain temporary security credentials using your AWS account 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 Java code example lists the object keys in the specified bucket. For illustration, the code example obtains temporary security credentials for a default one 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, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

```
import java.io.IOException;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetSessionTokenRequest;
import com.amazonaws.services.securitytoken.model.GetSessionTokenResult;
import com.amazonaws.services.s3.model.ObjectListing;

public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";

    public static void main(String[] args) throws IOException {
          PropertiesCredentials credentials = new PropertiesCredentials(
             S3Sample.class.getResourceAsStream("AwsCredentials.properties"));


        AWSSecurityTokenServiceClient stsClient = 
                              new AWSSecurityTokenServiceClient(credentials);


        //
        // Start a session.
        GetSessionTokenRequest getSessionTokenRequest = 
                                             new GetSessionTokenRequest();

        GetSessionTokenResult sessionTokenResult = 
                           stsClient.getSessionToken(getSessionTokenRequest);

        Credentials sessionCredentials = sessionTokenResult.getCredentials();
        System.out.println("Session Credentials: " 
                                             + sessionCredentials.toString());



        // Package the session credentials as a BasicSessionCredentials
        // object for an S3 client object to use.
        BasicSessionCredentials basicSessionCredentials = 
             new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),

                                   sessionCredentials.getSecretAccessKey(),
                                   sessionCredentials.getSessionToken());
        AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

        // Test. For example, get object keys for a given bucket.
        ObjectListing objects = s3.listObjects(bucketName);
        System.out.println("No. of Objects = " +
                                        objects.getObjectSummaries().size());

    }
}
```

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

An IAM user or an AWS Account can request temporary security credentials (see Making Requests (p. 11)) using AWS SDK for .NET 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 duration, between 1 and 36 hours, when requesting the temporary security credentials.

Making Requests Using IAM User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client `AmazonSecurityTokenServiceClient` by providing your credentials.
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 you temporary security credentials.
3	Package up 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.

The following C# code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenServiceConfig();

 AmazonSecurityTokenServiceClient stsClient =
           new AmazonSecurityTokenServiceClient("*** Access Key ID ***",
                                                "*** Secret Access Key ***",
                                                config);
```

```
GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();
// Following duration can be set only if temporary credentials are requested
by an IAM user.
getSessionTokenRequest.DurationSeconds = 7200; // seconds.
Credentials credentials =
    stsClient.GetSessionToken(getSessionTokenRequest).GetSessionTokenResult.Cre
dentials;

SessionAWSCredentials sessionCredentials =
                          new SessionAWSCredentials(credentials.AccessKeyId,
                                                  credentials.SecretAccessKey,

                                                    credentials.SessionToken);

// The following will be part of your less trusted code. You provide temporary
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3Client = new AmazonS3Client(sessionCredentials);

// Test. For example, send request to list object key in a bucket.
var response = s3Client.ListObjects(bucketName);
```

Example

Note

If you obtain temporary security credentials using your AWS account 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 C# code example lists object keys in the specified bucket. For illustration, the code 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 will need to create an IAM user under your AWS Account. For more information about how to create an IAM user, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using Amazon.Runtime;
using Amazon.S3.Model;
using System.Collections.Generic;

namespace s3.amazon.com.docsamples.listingkeys
{
    class TempCred_ExplicitSessionStart
    {
        static string bucketName = "*** Provide bucket name ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];
            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials =
                     GetTemporaryCredentials(accessKeyID, secretAccessKeyID);

                // Create client by providing temporary security credentials.
                AmazonS3Client s3Client = new AmazonS3Client(tempCredentials);


                ListObjectsRequest listObjectRequest =
                                  new ListObjectsRequest();
                listObjectRequest.BucketName = bucketName;

                // Send request to Amazon S3.
             ListObjectsResponse response = s3Client.ListObjects(listObjectRe
quest);
                List<S3Object> objects = response.S3Objects;
                Console.WriteLine("Object count = {0}", objects.Count);

                Console.WriteLine("Press any key to continue...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
            catch (AmazonSecurityTokenServiceException stsException)
            {
                Console.WriteLine(stsException.Message,
                                 stsException.InnerException);
            }
        }

        private static SessionAWSCredentials GetTemporaryCredentials(
                         string accessKeyId, string secretAccessKeyId)
        {
            AmazonSecurityTokenServiceClient stsClient =
                new AmazonSecurityTokenServiceClient(accessKeyId,
                                                     secretAccessKeyId);

            GetSessionTokenRequest getSessionTokenRequest =
                                     new GetSessionTokenRequest();
            getSessionTokenRequest.DurationSeconds = 7200; // seconds

            GetSessionTokenResponse sessionTokenResponse =
                          stsClient.GetSessionToken(getSessionTokenRequest);
            GetSessionTokenResult sessionTokenResult =
                                   sessionTokenResponse.GetSessionTokenResult;

            Credentials credentials = sessionTokenResult.Credentials;

            SessionAWSCredentials sessionCredentials =
        SessionAWSCredentials.CreateStaticCredentials(credentials.AccessKeyId,
                                                  credentials.SecretAccessKey,

                                                    credentials.SessionToken);


            return sessionCredentials;
        }
    }
}
```

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

An IAM user or an AWS Account can request temporary security credentials (see Making Requests (p. 11)) using the AWS SDK for PHP 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, between 1 and 36 hours, when requesting the temporary security credentials.

Making Requests Using IAM User Temporary Security Credentials

1	Create an instance of the `AmazonSTS` class by providing your credentials.
2	Execute the `AmazonSTS::get_session_token()` method to start a session. The method returns you temporary security credentials.
3	Create an instance of the `AmazonS3` class by providing the temporary security credentials you obtained in the preceding step. Any methods in the `AmazonS3` class that you call use the temporary security credentials to send authenticated requests to Amazon S3.

The following PHP code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It 
has 
// your security credentials that you use to obtain temporary security creden
tials.
$token = new AmazonSTS();
$response = $token->get_session_token();

$AccessKeyId = (string)$response->body->GetSessionTokenResult->Credentials-
>AccessKeyId;
$SecretAccessKey = (string)$response->body->GetSessionTokenResult->Credentials-
>SecretAccessKey;
$SessionToken = (string)$response->body->GetSessionTokenResult->Credentials-
>SessionToken;

// The following will be part of your less trusted code. You provide temporary
 security 
// credentials so it can send authenticated requests to Amazon S3.
// Create an AmazonS3 using temporary security credentials.
$s3 = new AmazonS3($AccessKeyId, $SecretAccessKey, $SessionToken);
// Send requests to Amazon S3.
```

Note
If you obtain temporary security credentials using your AWS account 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 code example lists object keys in the specified bucket. For illustration, the code 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 will need to create an IAM user under your AWS Account. For more information about how to create an IAM user, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';
header('Content-Type: text/plain; charset=utf-8');

$bucket = '*** Provide bucket name ***';

$token = new AmazonSTS();
$response1 = $token->get_session_token();

$AccessKeyId = (string)$response1->body->GetSessionTokenResult->Credentials-
>AccessKeyId;
$SecretAccessKey = (string)$response1->body->GetSessionTokenResult->Credentials-
>SecretAccessKey;
$SessionToken = (string)$response1->body->GetSessionTokenResult->Credentials-
>SessionToken;

// Instantiate the class.
$s3 = new AmazonS3($AccessKeyId, $SecretAccessKey, $SessionToken);

// Send list object request.

$response = $s3->list_objects($bucket);

// Success?
print_r(gettype($response) === 'array');

if($response)
{
   echo "Keys retrieved!" . PHP_EOL;

 foreach ($response as $key)
    {
        print_r($key);
    }
}
else
{
    print_r($response);
}
```

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

An IAM user or an AWS Account can request temporary security credentials (see Making Requests (p. 11)) 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, between 1 and 36 hours, when requesting the temporary security credentials.

Making Requests Using IAM User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client `AWS::STS::Session` by providing your credentials.
2	Start a session by calling the `new_session` method of the STS client that 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	Use the temporary credentials in a new instance of the `AWS::S3` class by passing in the temporary security credentials. You send the requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following Ruby code sample demonstrates the preceding tasks.

```
# Start a session.
# In real applications, the following code is part of your trusted code. It has

# your security credentials that you use to obtain temporary security creden
tials.

sts = AWS::STS.new(
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key)

session = sts.new_session()
puts "Session expires at: #{session.expires_at.to_s}"

# Get an instance of the S3 interface using the session credentials.
s3 = AWS::S3.new(session.credentials)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
```

Example

Note

If you obtain temporary security credentials using your AWS account 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 code example lists the object keys in the specified bucket. For illustration, the code example obtains temporary security credentials for a default one 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, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

```
require 'rubygems'
require 'aws-sdk'

# In real applications, the following code is part of your trusted code. It has

# your security credentials you use to obtain temporary security credentials.

my_access_key_id = '***Provide Access Key ID***'
my_secret_key = '***Provide Secret Key***'
bucket_name = '*** Provide bucket name ***'

AWS.config({
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key
})

# Start a session.
sts = AWS::STS.new()
session = sts.new_session()
puts "Session expires at: #{session.expires_at.to_s}"

# get an instance of the S3 interface using the session credentials
s3 = AWS::S3.new(session.credentials)

# get a list of all object keys in a bucket
bucket = s3.buckets[bucket_name].objects.collect(&:key)
puts bucket
```

Making Requests Using Federated User Temporary Credentials

Topics

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.

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 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 (see Making Requests (p. 11)) so they can send authenticated requests to access your AWS resources. When requesting these temporary credentials from the IAM service, you must provide a user name and an IAM policy describing the resource permissions 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.

Note

To request temporary security credentials for federated users and applications, for added security, 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, go to AWS Identity and Access Management FAQs

Making Requests Using Federated User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client `AWSSecurityTokenServiceClient` by providing your security credentials.
2	Start a session by calling the `getFederationToken` method of the STS client you created in the preceding step. You will need to provide session information including the user name and an IAM policy that you want to attach to the temporary credentials. This method returns your temporary security credentials.
3	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.

4	Create an instance of the `AmazonS3Client` class by passing 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.

The following Java code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AWSCredentials credentials =
          new BasicAWSCredentials("*** Access Key ID ***",
                                  "*** Secret Key ***");
AWSSecurityTokenServiceClient stsClient =
                        new AWSSecurityTokenServiceClient(credentials);

GetFederationTokenRequest getFederationTokenRequest =
                        new GetFederationTokenRequest();
getFederationTokenRequest.setDurationSeconds(7200);
getFederationTokenRequest.setName("User1");

// Define the policy and add to the request.
Policy policy = new Policy();
// Define the policy here.
// Add the policy to the request.
getFederationTokenRequest.setPolicy(policy.toJson());

GetFederationTokenResult federationTokenResult =
                     stsClient.getFederationToken(getFederationTokenRequest);
Credentials sessionCredentials = federationTokenResult.getCredentials();

// Package the session credentials as a BasicSessionCredentials object
// for an S3 client object to use.
BasicSessionCredentials basicSessionCredentials = new BasicSessionCredentials(

      sessionCredentials.getAccessKeyId(),
      sessionCredentials.getSecretAccessKey(),
      sessionCredentials.getSessionToken());

// The following will be part of your less trusted code. You provide temporary
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create an Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

// Test. For example, send list object keys in a bucket.
ObjectListing objects = s3.listObjects(bucketName);
```

Example

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

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application specific permissions when requesting temporary security credentials. This sample only lists objects in a specific bucket. Therefore, first 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 to only list your AWS resources. For more information about how to create an IAM user, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

You can now 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 list 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"
    }
  ]
}
```

You must update the following sample and provide the bucket name that you specified in the preceding federated user access policy.

```
import java.io.IOException;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.PropertiesCredentials;
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.services.s3.AmazonS3Client;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;
import com.amazonaws.services.s3.model.ObjectListing;

public class S3Sample {
 private static String bucketName = "*** Specify bucket name ***";
    public static void main(String[] args) throws IOException {
        PropertiesCredentials credentials = new PropertiesCredentials(
              S3Sample.class.getResourceAsStream("AwsCredentials.properties"));


        AWSSecurityTokenServiceClient stsClient =
                        new AWSSecurityTokenServiceClient(credentials);

        GetFederationTokenRequest getFederationTokenRequest =
                                         new GetFederationTokenRequest();
        getFederationTokenRequest.setDurationSeconds(7200);
        getFederationTokenRequest.setName("User1");

        // Define the policy and add to the request.
        Policy policy = new Policy();
        policy.withStatements(new Statement(Effect.Allow)
            .withActions(S3Actions.ListObjects)
            .withResources(new Resource("arn:aws:s3:::ExampleBucket")));

        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 S3 client object to use.
        BasicSessionCredentials basicSessionCredentials =
              new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),

                                   sessionCredentials.getSecretAccessKey(),
                                   sessionCredentials.getSessionToken());
        AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);


       // Test. For example, send ListBucket request using the temporary secur
ity credentials.
        ObjectListing objects = s3.listObjects(bucketName);
        System.out.println("No. of Objects = " + objects.getObjectSummar
ies().size());
    }
}
```

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

You can provide temporary security credentials for your federated users and applications (see Making Requests (p. 11)) 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 describing the resource permissions 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

To request temporary security credentials for federated users and applications, for added security, 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, go to AWS Identity and Access Management FAQs

Making Requests Using Federated User Temporary Credentials

1	Create an instance of the AWS Security Token Service client, `AmazonSecurityTokenServiceClient` class, by providing your credentials.
2	Start a session by calling the `GetFederationToken` method of the STS client. You will need to 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.
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 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.

The following C# code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It 
has 
// your security credentials you use to obtain temporary security credentials.
AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenServiceConfig();
AmazonSecurityTokenServiceClient stsClient = 
       new AmazonSecurityTokenServiceClient(
                        accessKeyId, secretAccessKeyId, config);

GetFederationTokenRequest federationTokenRequest = 
                                     new GetFederationTokenRequest();
federationTokenRequest.Name            = "User1";
federationTokenRequest.Policy          =  "*** Specify policy ***";
federationTokenRequest.DurationSeconds = 7200;

GetFederationTokenResponse federationTokenResponse = 
                  stsClient.GetFederationToken(federationTokenRequest);
```

```
GetFederationTokenResult federationTokenResult =
                  federationTokenResponse.GetFederationTokenResult;
Credentials credentials = federationTokenResult.Credentials;


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

// The following will be part of your less trusted code. You provide temporary
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3Client = new AmazonS3Client(sessionCredentials);
// Test. For example, send list object keys in a bucket.
ListObjectsRequest listObjectRequest =  new ListObjectsRequest();
listObjectRequest.BucketName = bucketName;
ListObjectsResponse response = s3Client.ListObjects(listObjectRequest);
```

Example

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

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application specific permissions to the federated user. This sample only lists objects in a specific bucket. Therefore, first 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 to only list your AWS resources. For more information about how to create an IAM user, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

You can now 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 list 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"
    }
  ]
}
```

You must update the following sample and provide the bucket name that you specified in the preceding federated user access policy.

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using Amazon.Runtime;
using Amazon.S3.Model;
using System.Collections.Generic;
```

```
namespace s3.amazon.com.docsamples.listingkeys
{
    class TempCred_FederatedCredentials
    {
        static string bucketName = "*** Provide bucket name ***";

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];
            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials =
                     GetTemporaryFederatedCredentials(accessKeyID,
                                                      secretAccessKeyID);

                AmazonS3Client s3Client = new AmazonS3Client(tempCredentials);


                ListObjectsRequest listObjectRequest =
                                  new ListObjectsRequest();
                listObjectRequest.BucketName = bucketName;

              ListObjectsResponse response = s3Client.ListObjects(listObjectRe
quest);
                List<S3Object> objects = response.S3Objects;
                Console.WriteLine("Object count = {0}", objects.Count);

                Console.WriteLine("Press any key to continue...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
            catch (AmazonSecurityTokenServiceException stsException)
            {
                Console.WriteLine(stsException.Message,
                                 stsException.InnerException);
            }
        }

     private static SessionAWSCredentials GetTemporaryFederatedCredentials(
                                string accessKeyId, string secretAccessKeyId)
        {
          AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenSer
viceConfig();
            AmazonSecurityTokenServiceClient stsClient =
                new AmazonSecurityTokenServiceClient(
                                             accessKeyId, secretAccessKeyId,
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:::ExampleBucket""
    }
  ]
}

";


            GetFederationTokenResponse federationTokenResponse =
                         stsClient.GetFederationToken(federationTokenRequest);

            GetFederationTokenResult federationTokenResult =
                             federationTokenResponse.GetFederationTokenResult;

            Credentials credentials = federationTokenResult.Credentials;


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

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

You can provide temporary security credentials to your federated users and applications (see Making Requests (p. 11)) 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 describing the resource permissions 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
To request temporary security credentials for federated users and applications, for added security, 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, go to AWS Identity and Access Management FAQs.

Making Requests Using Federated User Temporary Credentials

1	Create an instance of the `AmazonSTS` class by providing your credentials.
2	Create an instance of the `CFPolicy` by providing the IAM policy that you wish to attach to the temporary security credentials.
3	Execute the `AmazonSTS::get_federation_token()` method by providing the user name, the policy, and the optional session duration. The method returns you temporary security credentials. You can provide these credentials to your federated users.
4	Any federated user who has the temporary security credentials can send requests to Amazon S3 by creating an instance of the `AmazonS3` class by providing the temporary security credentials. Any methods in the `AmazonS3` class that you call use the temporary security credentials to send authenticated requests to Amazon S3.

The following PHP code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It 
has 
// your security credentials that you used to obtain temporary security creden
tials.
$token = new AmazonSTS();

$policy = new CFPolicy($token, array(
    'Statement' => array(
        array(
            'Sid' => 'randomstatementid' . time(),
            'Action' => array('s3:ListBucket'),
            'Effect' => 'Allow',
            'Resource' => 'arn:aws:s3:::YourBucketName'
        )
    )
));

// Fetch the session credentials.
$response1 = $token->get_federation_token(
                      'User1',
                       array(
                          'Policy' => $policy->get_json(),
                          'DurationSeconds' => 3600
));

$AccessKeyId = (string)$response1->
                body->GetFederationTokenResult->Credentials->AccessKeyId;
$SecretAccessKey = (string)$response1->
                body->GetFederationTokenResult->Credentials->SecretAccessKey;
$SessionToken = (string)$response1->
                body->GetFederationTokenResult->Credentials->SessionToken;

// The following will be part of your less trusted code. You provide temporary
 security 
// credentials so it can send authenticated requests to Amazon S3.
$s3 = new AmazonS3(array(
 'key' => $AccessKeyId,
 'secret' => $SecretAccessKey,
 'token' => $SessionToken
 ));
// Send requests to Amazon S3.
```

Example

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

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application specific permissions to the federated user. This sample only lists objects in a specific bucket. Therefore, first 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 to only list your AWS resources. For more information about how to create an IAM user, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

You can now 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 list objects in a specific bucket. You must update the policy and provide your own existing bucket name.

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

You must update the following sample and provide the bucket name that you specified in the preceding federated user access policy.

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';
header('Content-Type: text/plain; charset=utf-8');

$bucket = 'ExampleBucket';

$token = new AmazonSTS();

$policy = new CFPolicy($token, array(
    'Statement' => array(
        array(
            'Sid' => 'randomstatementid' . time(),
            'Action' => array('s3:ListBucket'),
            'Effect' => 'Allow',
            'Resource' => 'arn:aws:s3:::ExampleBucket'
        )
    )
));

// Fetch the session credentials.
$response1 = $token->get_federation_token(
                      'User1',
                       array(
                          'Policy' => $policy->get_json(),
                          'DurationSeconds' => 3600
));

$AccessKeyId = (string)$response1->
                body->GetFederationTokenResult->Credentials->AccessKeyId;
$SecretAccessKey = (string)$response1->
                body->GetFederationTokenResult->Credentials->SecretAccessKey;
$SessionToken = (string)$response1->
                body->GetFederationTokenResult->Credentials->SessionToken;

// Instantiate the class.
$s3 = new AmazonS3(array(
 'key' => $AccessKeyId,
 'secret' => $SecretAccessKey,
 'token' => $SessionToken
 ));
// Get object list using temporary credentials.

//$response = $s3->get_object_list($bucket);
$response = $s3->list_objects($bucket);

// Success?
print_r($response);
```

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

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

You can provide temporary security credentials for your federated users and applications (see Making Requests (p. 11)) so that they can send authenticated requests to access your AWS resources. When requesting these temporary credentials from the IAM service, you must provide a user name and an IAM policy describing the resource permissions 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.

Note
To request temporary security credentials for federated users and applications, for added security, 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, go to AWS Identity and Access Management FAQs

Making Requests Using Federated User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client `AWS::STS::Session` by providing your security credentials.
2	Start a session by calling the `new_federated_session` method of the STS client you created in the preceding step. You will need to provide session information including the user name and an IAM policy that you want to attach to the temporary credentials. This method returns your temporary security credentials.
3	Create an instance of the `AWS::S3` class by passing 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.

The following Ruby code sample demonstrates the preceding tasks.

```
# Start a session with restricted permissions.
sts = AWS::STS.new()
policy = AWS::STS::Policy.new
policy.allow(
  :actions => ["s3:ListBucket"],
  :resources => "arn:aws:s3:::#{bucket_name}")

session = sts.new_federated_session(
  'User1',
  :policy => policy,
  :duration => 2*60*60)

puts "Policy: #{policy.to_json}"

# Get an instance of the S3 interface using the session credentials.
s3 = AWS::S3.new(session.credentials)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
```

Example

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

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application specific permissions when requesting temporary security credentials. This sample only lists objects in a specific bucket. Therefore, first 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 to only list your AWS resources. For more information about how to create an IAM user, go to Set Up a Group, Grant Permissions, and Add Users in the *AWS Identity and Access Management Getting Started Guide*.

You can now 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 list 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"
    }
  ]
}
```

You must update the following sample and provide the bucket name that you specified in the preceding federated user access policy.

```
require 'rubygems'
require 'aws-sdk'

# In real applications, the following code is part of your trusted code. It has

# your security credentials that you use to obtain temporary security creden
tials.

my_access_key_id = '***Provide Access Key ID***'
```

```
my_secret_key = '***Provide Secret Key***'
bucket_name = '*** Provide bucket name ***'

AWS.config({
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key
})

# Start a session with restricted permissions.
sts = AWS::STS.new()
policy = AWS::STS::Policy.new
policy.allow(
  :actions => ["s3:ListBucket"],
  :resources => "arn:aws:s3:::#{bucket_name}")

session = sts.new_federated_session(
  'User1',
  :policy => policy,
  :duration => 2*60*60)

puts "Policy: #{policy.to_json}"

# Get an instance of the S3 interface using the session credentials.
s3 = AWS::S3.new(session.credentials)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
puts "No. of Objects = #{bucket.count.to_s}"
puts bucket
```

Making Requests using the REST API

Topics

This section contains information specific to the Amazon S3 REST API. The examples in this guide use the newer, virtual, hosted-style method for accessing buckets instead of the path-style. For more information, see Working with Amazon S3 Buckets (p. 81)

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

```
DELETE /puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: mybucket.s3.amazonaws.com
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

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

```
DELETE /mybucket/puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: s3.amazonaws.com
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

Amazon S3 supports virtual-hosted-style and path-style access in all Regions. The path-style syntax, however, 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, 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 a “PermanentRedirect” error, an HTTP response code 301, and a message indicating what the correct URI is for your resource if you try to access a non US Standard bucket with path-style syntax using:

- `http://s3.amazonaws.com`
- A different Region endpoint than where the bucket resides, for example, `http://s3-euwest-1.amazonaws.com` and the bucket was created with the location constraint of Northern-California

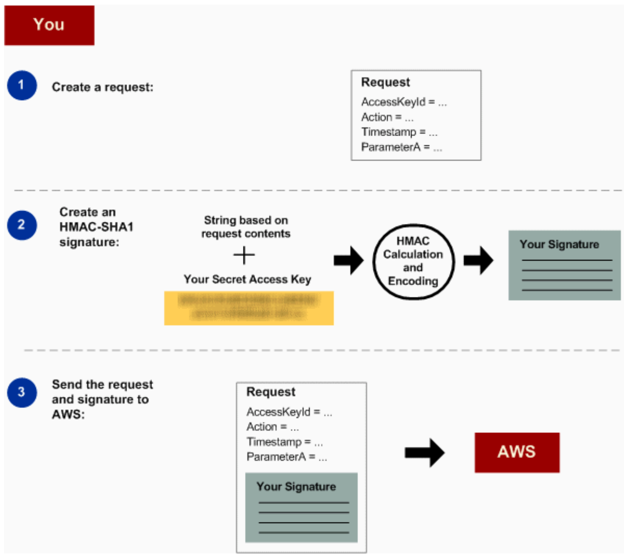
Authenticating Requests Using the REST API

When accessing Amazon S3 using REST and SOAP, you must provide the following items in your request so the request can be authenticated:

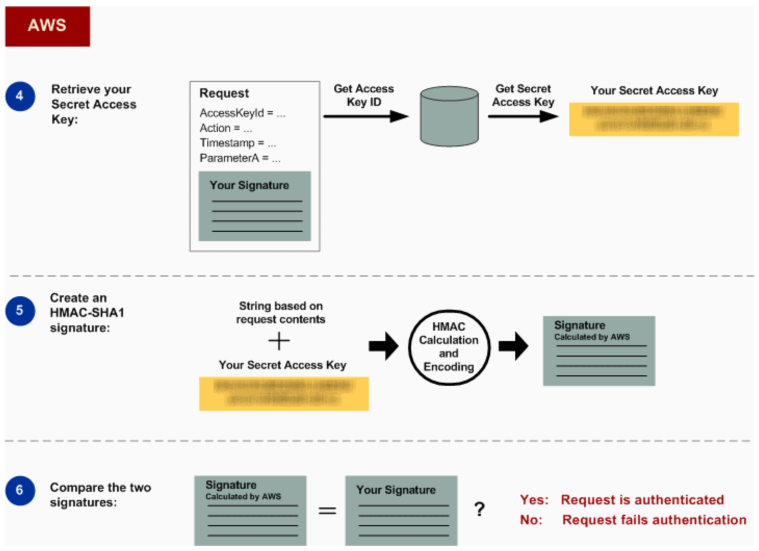
Request Elements

- **AWS Access Key Id—**It is 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 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 API to determine what the API 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 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.


AWS
4 Retrieve your Secret Access Key:
Request
AccessKeyId = ...
Action = ...
Timestamp = ...
ParameterA = ...
Your Signature
Get Access Key ID
Get Secret Access Key
Your Secret Access Key
5 Create an HMAC-SHA1 signature:
String based on request contents
Your Secret Access Key
HMAC Calculation and Encoding
Signature Calculated by AWS
6 Compare the two signatures:
Signature Calculated by AWS
Your Signature
?
Yes: Request is authenticated
No: Request fails authentication

4	Amazon S3 uses the Access Key ID to look up your Secret Access Key.
5	Amazon S3 calculate a signature from the request data and the Secret Access Key using the same algorithm 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 and SOAP authentication, see Signing and Authenticating REST Requests (p. 48) and Authenticating SOAP Requests (p. 459).

Signing and Authenticating REST Requests

Topics

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 (p. 65).

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, 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, then 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. 11)), 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 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 SecretAccess Key when they register. For request authentication, the*`AWSAccessKeyId`* element identifies the secret key that was used to compute the signature, and (indirectly) the developer making the request.

The *`Signature`* element is the RFC 2104HMAC-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, then the requester will have demonstrated possession to the AWSSecret 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 pseudo-grammar that illustrates the construction of the *Authorization* request header (`\n`means the Unicode code point `U+000A` commonly called newline).

```
Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;

Signature = Base64( HMAC-SHA1( YourSecretAccessKeyID, 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> +
 [ sub-resource, if present. For example "?acl", "?location", "?logging", or
"?torrent"];

CanonicalizedAmzHeaders = <described below>
```

HMAC-SHA1 is an algorithm defined by RFC 2104 (go to 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 (*YourSecretAccessKeyID*) 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 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	If the request specifies a bucket using the HTTP Host header (virtual hosted-style), append the bucket name preceded by a `"/"` (e.g., "/bucketname"). For path-style requests and requests that don't address a bucket, do nothing. For more information on virtual hosted-style requests, see Virtual Hosting of Buckets (p. 59). For a virtual hosted-style request "https://johnsmith.s3.amazonaws.com/photos/puppy.jpg", the `CanonicalizedResource` is "/johnsmith". For the path-style request, "https://s3.amazonaws.com/johnsmith/photos/puppy.jpg", the `CanonicalizedResource` is "".
3	Append the path part of the un-decoded HTTP Request-URI, up-to but not including the query string. For a virtual hosted-style request "https://johnsmith.s3.amazonaws.com/photos/puppy.jpg", the `CanonicalizedResource` is "/johnsmith/photos/puppy.jpg". For a path-style request, "https://s3.amazonaws.com/johnsmith/photos/puppy.jpg", the `CanonicalizedResource` is "/johnsmith/photos/puppy.jpg". At this point, the `CanonicalizedResource` is the same for both the virtual hosted-style and path-style request. For a request that does not address a bucket, such as GET Service, append "/".
4	If the request addresses a sub-resource, like `?versioning`, `?location`, `?acl`, `?torrent`, `?lifecycle`, or `?versionid` append the sub-resource, its value if it has one, and the question mark. Note that in case of multiple sub-resources, sub-resources must be lexicographically sorted by sub-resource name and separated by '&'. e.g. ?acl&versionId=*value*. The list of sub-resources that must be included when constructing the CanonicalizedResource Element are: acl, lifecycle, location, logging, notification, partNumber, policy, requestPayment, torrent, uploadId, uploads, versionId, versioning, versions and website. If the request specifies query string parameters overriding the response header values (see Get Object), append the query string parameters, and its 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 `response-content-type`, `response-content-language`, `response-expires`, `response-cache-control`, `response-content-disposition`, `response-content-encoding`. The `delete` query string parameter must be including when creating 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 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. 59).

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 lower-case. 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 white-space 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 white-space (including new-line) by a single space.
5	Trim any white-space 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 new-line (`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, (`Content-Type` or `Content-MD5`, for example, are optional for PUT requests, and meaningless for GET requests), substitute the empty string ("") in 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 time-stamp 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 status 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 only applies to authenticated requests that do not use query string authentication. For more information, see Query String Request Authentication Alternative (p. 57).

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 time-stamp for the request 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 *`StringToSign`*s, formatting is not significant and `\n` means the Unicode code point `U+000A` commonly called newline. Also, the examples use "+0000" to designate timezone. You can use "GMT" to designate timezone instead, but the signatures shown in the examples will be different.

Example Object GET

This example gets an object from the johnsmith bucket.

Request	StringToSign

`GET /photos/puppy.jpg HTTP/1.1`
`Host: johnsmith.s3.amazonaws.com`
`Date: Tue, 27 Mar 2007 19:36:42`
`+0000`
 	
`Authorization: AWS AKIAIOSFODNN7EX`
`AMPLE:`
`bWq2s1WEIj+Ydj0vQ697zp+IXMU=`
	`GET\n`
	`\n`
	`\n`
	`Tue, 27 Mar 2007 19:36:42 +0000\n`
	`/johnsmith/photos/puppy.jpg`

Note that the CanonicalizedResource includes the bucket name, but the HTTP Request-URI does not (it is specified by the Host header)

Example Object PUT

This example puts an object into the johnsmith bucket.

Request	StringToSign

`PUT /photos/puppy.jpg HTTP/1.1`
`Content-Type: image/jpeg`
`Content-Length: 94328`
`Host: johnsmith.s3.amazonaws.com`
`Date: Tue, 27 Mar 2007 21:15:45 +0000`
 	
`Authorization: AWS AKIAIOSFODNN7EX`
`AMPLE:`
`MyyxeRY7whkBe+bq8fHCL/2kKUg=`
	`PUT\n`
	`\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 since it is not present in the request.

Example 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`
 	
`Authorization: AWS AKIAIOSFODNN7EXAMPLE:`
`htDYFYduRNen8P9ZfE/s9SuKy0U=`
	`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.

Example Fetch

This example fetches the access control policy sub-resource 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`
 	
`Authorization: AWS AKIAIOSFODNN7EXAMPLE:`
`c2WLPFtWHVgbEmeEG93a4cG37dM=`
	`GET\n`
	`\n`
	`\n`
	`Tue, 27 Mar 2007 19:44:46 +0000\n`
	`/johnsmith/?acl`

Notice how the sub-resource query string parameter is included in the CanonicalizedResource.

Example 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`
`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:9b2sXq0KfxsxHtdZkzx/9Ngqyh8=`
	`DELETE\n`
	`\n`
	`\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 field for the actual 'Date' header is left blank in the StringToSign.

Example Upload

This example uploads an object to a CNAME style virtual hosted bucket with metadata.

Request

```
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; file
name=database.dat
Content-Encoding: gzip
Content-Length: 5913339

Authorization: AWS AKIAIOSFODNN7EXAMPLE:
ilyl83RwaSoYIEdixDQcA4OnAnc=
```

StringToSign

```
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@johns
mith.net\n
/static.johnsmith.net/db-
backup.dat.gz
```

Notice how the 'x-amz-' headers are sorted, white-space trimmed, converted to lowercase, and multiple headers with the same name have been joined using a comma 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).

Example List All My Buckets

Request

```
GET / HTTP/1.1
Host: s3.amazonaws.com
Date: Wed, 28 Mar 2007 01:29:59 +0000

Authorization: AWS AKIAIOSFODNN7EXAMPLE:qGdzdER
IC03wnaRNKh6OqZehG9s=
```

StringToSign

```
GET\n
\n
\n
Wed, 28 Mar 2007 01:29:59
+0000\n
/
```

Example Unicode Keys

Request	StringToSign

`GET /diction ary/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: AWS AKIAIOSFODNN7EXAMPLE:DNEZGsoieTZ92F3bUfSPQcbGmlM=`
	`GET\n`
	`\n`
	`\n`
	`Wed, 28 Mar 2007 01:49:49 +0000\n`
	`/diction ary/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 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 "pre-signed" request and encode it as a URL that an end-user's browser can retrieve. Additionally, you can limit a pre-signed request by specifying an expiration time.

Creating a Signature

Following is an example query string authenticated Amazon S3 REST request.

```
GET /photos/puppy.jpg
?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1141889120&Signature=vjbyPxybdZaN
mGa%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( YourSecretAccessKeyID, UTF-8-Encoding-
Of( StringToSign ) ) ) );

StringToSign = HTTP-VERB + "\n" +
    Content-MD5 + "\n" +
    Content-Type + "\n" +
    Expires + "\n" +
    CanonicalizedAmzHeaders +
    CanonicalizedResource;
```

`YourSecretAccessKeyID` is the AWS Secret Access Key ID 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. Also note 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.

Example Query String Request Authentication

Request	StringToSign

`GET /photos/puppy.jpg?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&`
`    Signature=NpgCjnDzrM%2BWFzoENXmpNDUsSn8%3D&`
`    Expires=1175139620 HTTP/1.1`
 	
`Host: johnsmith.s3.amazonaws.com`
	`GET\n`
	`\n`
	`\n`
	`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. 52).

Virtual Hosting of Buckets

Topics

Virtual Hosting, in general, is the practice of serving multiple web sites from a single web server. One way to differentiate sites is by using the apparent host name of the request instead of just the path name part of the URI. An ordinary Amazon S3 REST request specifies a bucket using the first slash delimited component of the Request-URI path. Alternatively, using Amazon S3 Virtual Hosting, you can address a bucket in a REST API call 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 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.

Important

Amazon S3 supports virtual-hosted-style and path-style access in all Regions. The path-style syntax, however, 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, 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 a “PermanentRedirect” error, an HTTP response code 301, and a message indicating what the correct URI is for your resource if you try to access a non US Standard bucket with path-style syntax using:

- `http://s3.amazonaws.com`
- A different Region endpoint than where the bucket resides, for example, `http://s3-euwest-1.amazonaws.com` and the bucket was created with the location constraint of Northern-California

Note
Amazon S3 routes any virtual hosted style requests to the US standard region by default. When you create a bucket, in any region, Amazon S3 updates DNS to re-route 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 standard region and Amazon S3 redirects it with HTTP 307 redirect to the correct region. For more information, see Request Redirection and the REST API (p. 404).
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 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 example in the following table. Note that omitting the Host header is only legal for HTTP 1.0 requests.
- Otherwise, if the value of the `Host` header ends in '.s3.amazonaws.com', then 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 sub-domains of s3.amazonaws.com, and is illustrated by the third and fourth example in the following table.
- Otherwise, the bucket for the request will be the lower-cased 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 outside the scope of this document, but the result is illustrated by the final example in the following table.

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.

Following is the example URL.

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

Following is the example request.

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

Following is the example request with HTTP 1.0 omitting the host header.

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

For information about DNS compatible names, see Limitations (p. 63). 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.

Following is the example URL.

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

Following is the example request.

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

Following is the example request with the incorrect case. Notice that sentence case is irrelevant. However, uppercase buckets are not accessible using this method.

```
GET /homepage.html HTTP/1.1
Host: JohnSmith.net.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.

Following is the example URL.

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

Following is the example request.

```
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 web site or service. For example, if you host your web site's images on Amazon S3, you might prefer `http://images.johnsmith.net/` as opposed to `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 may be referenced as follows: `http://[`*`BucketName`*`].s3.amazonaws.com/[`*`Filename`*`]`, for example, `http://images.johnsmith.net.s3.amazonaws.com/mydog.jpg`. Using CNAME you can map `images.johnsmith.net` to an Amazon S3 host name so 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 host name. For example, if your bucket name (and domain name) is `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 may result in extra HTTP redirects.

Amazon S3 uses the host name to determine the bucket name. For example, assume 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:

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

Because Amazon S3 only sees the original host name `www.example.com` and is unaware of the CNAME mapping used to resolve the request, 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. 13).

To associate a host name with an Amazon S3 bucket using CNAMEs

1. Select a host name 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 host name.
   In this example, the host and bucket names are `images.johnsmith.net`.

   Note
   Your bucket name must exactly match the host name.
3. Create a CNAME record that defines the host name 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 will eventually result in unpredictable behavior.

   Note
   The exact procedure for configuring DNS depends on your DNS server or DNS provider and is beyond scope of this document.

Limitations

Because DNS names are case insensitive, only lower-case buckets are addressable using the virtual hosting method. For more information, see Bucket Restrictions and Limitations (p. 82).

Specifying the bucket for the request using the HTTP `Host` header is supported for non-SSL requests and when using the REST API. You cannot specify the bucket in SOAP by using a different endpoint.

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 present, the most likely symptom of this problem is to receive an unexpected `NoSuchBucket` error result code.

Request Redirection and the REST API

Topics

This section describes how to handle HTTP redirects using REST. For general information about Amazon S3 redirects, see Request Redirection and the REST API (p. 404).

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 relying on a library to fulfill the redirect requirement, test the following cases:

Launch Process

1	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.
2	Verify non-GET redirects, such as PUT and DELETE, work correctly.
3	Verify large PUT requests follow redirects correctly.
4	Verify PUT requests follow redirects correctly if the 100-continue response takes a long time to arrive.

HTTP user-agents that strictly conform to RFC2616 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 only issue redirects to hosts within the amazonaws.com domain and the effect of the redirected request will be the same as 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.

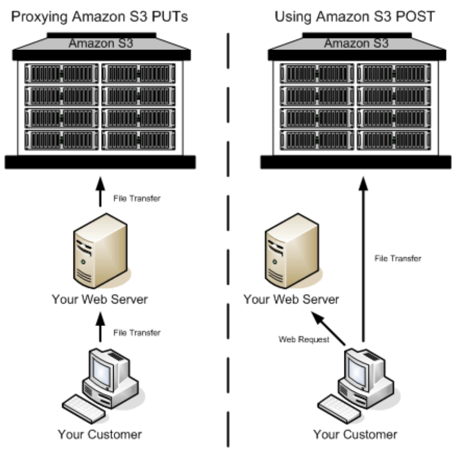
```
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2007 22:18:48 GMT

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

Browser-Based Uploads Using POST

Topics

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.

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

Topics

When communicating with Amazon S3, you normally use the REST or SOAP APIs to perform put, get, delete, and other operations. With POST, users upload data directly to Amazon S3 through their browsers, which do not understand SOAP APIs or how to make a REST PUT request.

To allow users to upload content to Amazon S3 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 make sure that it meets conditions that you specify.

Note
The form data and boundaries (excluding the contents of the file) cannot exceed 20K.

This section describes 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. For information about query string authentication, see Using Query String Authentication (p. 338).

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>
```

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 (go to RFC 1867) for both file uploads and text area uploads.

Example

This is a form declaration for the bucket "johnsmith".

```
<form action="http://johnsmith.s3.amazonaws.com/" method="post"

enctype="multipart/form-data">
```

HTML Form Fields

Following is a table that describes a list of fields that can be used within a 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 (e.g., "C:\Program Files\directory1\file.txt" will be interpreted as "file.txt"). If no file or filename is provided, the variable is replaced with an empty string.

Element Name	Description	Required
AWSAccessKeyId	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 is required if a policy document is included with the request.	Conditional
acl	Specifies an Amazon S3 access control list. If an invalid access control list is specified, an error is generated. For more information on ACLs, see Amazon S3 ACLs (p. 8). Type: String Default: private Valid Values: private | public-read | public-read-write | authenticated-read | bucket-owner-read | bucket-owner-full-control	No
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	REST-specific headers. For more information, see PUT Object.	No
key	The name of the uploaded key. To use the filename provided by the user, use the ${filename} variable. For example, if the user Betty uploads the file the file lolcatz.jpg and you specify /user/betty/${filename}, the file is stored as /user/betty/lolcatz.jpg. For more information, see Object Key and Metadata (p. 101) .	Yes
policy	Security Policy describing what is permitted in the request. Requests without a security policy are considered anonymous and only work on publicly writable buckets.	No

Element Name	Description	Required
success_action_redirect, redirect	The URL to which the client is redirected upon successful upload. Amazon S3 appends the bucket, key and etag values as query string parameters to the URL. If success_action_redirect is not specified, Amazon S3 returns the empty document type specified in the success_action_status field. If Amazon S3 cannot interpret the URL, it acts as if the field is not present. If the upload fails, Amazon S3 displays an error and does not redirect the user to a URL. For more information, see Redirection (p. 73). **Note** The redirect field name is deprecated and support for the redirect field name will be removed in the future.	No
success_action_status	The status code returned to the client upon successful upload if success_action_redirect is not specified. Accepts the values 200, 201, or 204 (default). If the value is set to 200 or 204, Amazon S3 returns an empty document with a 200 or 204 status code. If the value is set to 201, Amazon S3 returns an XML document with a 201 status code. For information on the content of the XML document, see POST Object. 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. **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.	No
signature	The HMAC signature constructed using the secret key of the provided AWSAccessKeyId. This is required if a policy document is included with the request. For more information, see Using Auth Access .	Conditional
x-amz-security-token	Amazon DevPay security token. Each request that uses Amazon DevPay requires two `x-amz-security-token` form fields: one for the product token and one for the user token. For more information, see Using DevPay .	No

Element Name	Description	Required
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, as any fields below it are ignored. You cannot upload more than one file at a time.	Yes

Policy Construction

Topics

The policy is a UTF-8 and Base64 encoded JSON document that specifies conditions which 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.

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 specifies the expiration date of the policy in ISO8601 GMT date format. For example, "2007-12-01T12:00:00.000Z" specifies that the policy is not valid after 12:00 GMT on 2007-12-01. Expiration is required in a policy.

Conditions

The conditions in the policy document are used to 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 prior to validating the policy. Therefore, all condition matching should be 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. 72).

Policy document conditions are described in the following table.

Element Name	Description
acl	Specifies conditions the ACL must meet. Supports exact matching and *`starts-with`*.
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 *`starts-with`*.
key	The name of the uploaded key. Supports exact matching and *`starts-with`*.
success_action_redirect, redirect	The URL to which the client is redirected upon successful upload. Supports exact matching and *`starts-with`*.
success_action_status	The status code returned to the client upon successful upload if success_action_redirect is not specified. Supports exact matching.
x-amz-security-token	Amazon DevPay security token. Each request that uses Amazon DevPay requires two `x-amz-security-token` form fields: one for the product token and one for the user token. As a result, the values must be separated by commas. For example, if the user token is *`eW91dHViZQ==`* and the product token is *`b0hnNVNKWVJIQTA=`*, you set the policy entry to: `{ "x-amz-security-token": "eW91dHViZQ==,b0hnNVNKWVJIQTA=" }`. For more information about Amazon DevPay, see Using DevPay .
Other field names prefixed with x-amz-meta-	User-specified metadata. Supports exact matching and *`starts-with`*.

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

Following is a table that 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:
	`{"acl": "public-read" }`
	This example is an alternate way to indicate that the ACL must be set to public-read:
	`[ "eq", "$acl", "public-read" ]`

Starts With	If the value must start with a certain value, use starts-with. This example indicates that the key must start with user/betty: `["starts-with", "$key", "user/betty/"]`
Matching Any Content	To configure the policy to allow any content within a field, use starts-with with an empty value. This example allows any success_action_redirect: `["starts-with", "$success_action_redirect", ""]`
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: `["content-length-range", 1048579, 10485760]`

Character Escaping

Characters that must be escaped within a policy document are described in the following table.

Escape Sequence	Description
\\	Backslash
\$	Dollar symbol
\b	Backspace

Escape Sequence	Description
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\u*xxxx*	All Unicode characters

Constructing a Signature

Step	Description
1	Encode the policy using UTF-8.
2	Encode those UTF-8 bytes using Base64.
3	Sign the policy with your Secret Access Key using HMAC SHA-1.
4	Encode the SHA-1 signature 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, 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 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

Topics

File Upload

This example shows the complete process for constructing a policy and form to upload a file attachment.

Policy and Form Construction

Following is a policy that supports uploads to Amazon S3 for the johnsmith bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "johnsmith"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
   {"success_action_redirect": "http://johnsmith.s3.amazonaws.com/successful_up
load.html"},
    ["starts-with", "$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 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/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

Following is a Base64 encoded version of this policy.

```
eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlci9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL3N1Y2Nlc3NmdWxfdXBsb2FkLmh0bWwifSwKICAgIFsic3RhcnRzLXdpdGgiL
CAiJENvbnRlbnQtVHlwZSIsICJpbWFnZS8iXSwKICAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MT
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRhZyIsICIiXQo
gIF0KfQo=
```

Using your credentials create a signature, for example `0RavWzkygo6QX9caELEqKi9kDbU=` is the signature for the preceding Policy document.

Following is a form that supports a POST to the johnsmith.net bucket using 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="mul
tipart/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://johns
mith.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,applica
tion/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"

eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlci9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL3N1Y2Nlc3NmdWxfdXBsb2FkLmh0bWwifSwKICAgIFsic3RhcnRzLXdpdGgiL
CAiJENvbnRlbnQtVHlwZSIsICJpbWFnZS8iXSwKICAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MT
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRhZyIsICIiXQo
gIF0KfQo=
--9431149156168
Content-Disposition: form-data; name="Signature"

0RavWzkygo6QX9caELEqKi9kDbU=
--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+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/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=john
smith&key=user/eric/MyPicture.jpg&etag=&quot;39d459df
bc0faabbb5e179358dfb94c3&quot;
Server: AmazonS3
```

Text Area Upload

Topics

This example shows the complete process for constructing a policy and form to upload a text area. This is useful for submitting user-created content such as blog postings.

Policy and Form Construction

Following is a policy that supports text area uploads to Amazon S3 for the johnsmith bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "johnsmith"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
    {"success_action_redirect": "http://johnsmith.s3.amazon
aws.com/new_post.html"},
    ["eq", "$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.

```
eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXR
pb25zIjogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiw
gIiRrZXkiLCAidXNlci9lcmljLyJd
LAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIHsic3VjY2Vzc19hY3Rpb25fcm
VkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0a
C5zMy5hbWF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlbnQtVHl
wZSIsICJ0ZXh0L2h0bWwiXSwKI
CAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MTIzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIi
wgIiR4LWFtei1tZXRhLXRhZy
IsICIiXQogIF0KfQo=
```

Using your credentials create a signature, for example `qA7FWXKq6VvU68lI9KdveT1cWgF=` is the signature for the preceding Policy document.

Following is a form that supports a POST to the johnsmith.net bucket using 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="mul
tipart/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://johns
mith.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,applica
tion/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"
eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlci9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlbnQtVHlwZSIs
ICJ0ZXh0L2h0bWwiXSwKICAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MTIzN
jUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRhZyIsICIiXQogIF0KfQo=

--178521717625888
Content-Disposition: form-data; name="Signature"

qA7FWXKq6VvU68lI9KdveT1cWgF=
--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=johns
mith&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 public-readable crossdomain.xml file to the bucket that will accept POST uploads. 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 web site.
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 building Adobe Flash applications that upload to Amazon S3 using the FileReference API, 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. When set, Amazon S3 returns an XML document with a 201 status code. For information on the content of the XML document, see POST Object. For information on form fields, go to Form Fields.

Working with Amazon S3 Buckets

Topics

Every object stored in Amazon S3 is contained in a bucket. Buckets partition the namespace of objects stored in Amazon S3 at the top level. Within a bucket, you can use any names for your objects, but bucket names must be unique across all of Amazon S3.

Buckets are similar to Internet domain names. Just as Amazon is the only owner of the domain name Amazon.com, only one person or organization can own a bucket within Amazon S3. Once you create a uniquely named bucket in Amazon S3, you can organize and name the objects within the bucket in any way you like and the bucket will remain yours for as long as you like and as long as you have the Amazon S3 account.

The similarities between buckets and domain names is not a coincidence—there is a direct mapping between Amazon S3 buckets and subdomains of s3.amazonaws.com. Objects stored in Amazon S3 are addressable using the REST API under the domain *`bucketname`*.s3.amazonaws.com. For example, if the object `homepage.html` is stored in the Amazon S3 bucket `mybucket` its address would be `http://mybucket.s3.amazonaws.com/homepage.html`. For more information, see Virtual Hosting of Buckets (p. 59).

To determine whether a bucket name exists using REST, use `HEAD`, specify the name of the bucket, and set `max-keys` to 0. To determine whether a bucket name exists using SOAP, use `ListBucket` and set `MaxKeys` to 0. A NoSuchBucket response indicates that the bucket is not available, an AccessDenied response indicates that someone else owns the bucket, and a Success response indicates that you own the bucket or have permission to access it.

For a list of AWS regions where you can create a bucket, go to Regions and Endpoints in the *Amazon Web Services General Reference*.

Bucket Restrictions and Limitations

A bucket is owned by the AWS account that created it. Each AWS account can own up to 100 buckets at a time. 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, however the name might not be available for you to resuse for various reasons. For example, some other account can create a bucket with that name. So if you want to use the same bucket name, don't delete the bucket. Note that it might take some time before the name can be reused.

There is no limit to the number of objects that can be stored in a bucket and no variation 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.

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 make bucket create or delete calls 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.

Note
If you are using Amazon DevPay, each of your customers can have up to 100 buckets for each Amazon DevPay product they use. For more information, see Using Amazon DevPay with Amazon S3 (p. 412).

Rules for Bucket Naming

In all regions except for the US Standard region a bucket name mustcomply with the following rules. These result in a DNS compliant bucket name.

- Bucket names must be at least 3 and no more than 63 characters long
- Bucket name must be a series of one or more labels separated by a period (.), where each label:
  - Must start with a lowercase letter or a number
  - Must end with a lowercase letter or a number
  - Can contain lowercase letters, numbers and dashes
- Bucket names must not be formatted as an IP address (e.g., 192.168.5.4)

The following are examples of valid bucket names:

- `myawsbucket`
- `my.aws.bucket`
- `myawsbucket.1`

The following are examples of invalid bucket names:

Invalid Bucket Name	Comment
`.myawsbucket`	Bucket name cannot start with a period (.).

Invalid Bucket Name	Comment
`myawsbucket.`	Bucket name cannot end with a period (.).
`my..examplebucket`	There can only be one period between labels.

The rules for bucket names in the US Standard region are similar but less restrictive:

- Bucket names can be as long as 255 characters.
- Bucket names can contain any combination of uppercase letters, lowercase letters, numbers, periods (.), dashes (-) and underscores (_)

These naming rules for US Standard region can result in a bucket name that is not DNS-compliant. For example, `MyAWSBucket`, is a valid bucket name, with uppercase letters in its name. If you try to access this bucket 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 will return a bucket not found error. To avoid this problem, we recommend as a best practice that you always DNS-compliant bucket names regardless of the region in which you create the bucket. For more information about virtual-hosted style access to your buckets, see Virtual Hosting of Buckets (p. 59).

Bucket Configuration Options

When creating buckets, you can take advantage of additional Amazon S3 features by attaching the `<CreateBucketConfiguration>` XML body to a `PUT Bucket` request. Currently, you can select a location constraint. For more information, see Buckets and Regions (p. 83).

Buckets created with `<CreateBucketConfiguration>` are subject to additional restrictions:

- You cannot make a request to a bucket created with `<CreateBucketConfiguration>` using a path-style request; you must use the virtual hosted-style request. For more information, see Virtual Hosting of Buckets (p. 59).
- You must follow additional bucket naming restrictions. For more information, see Bucket Restrictions and Limitations (p. 82).

Buckets and Regions

You can choose the geographical Region where Amazon S3 will store the buckets you create. You might choose a Region to optimize latency, minimize costs, or address regulatory requirements. For example, if you reside in Europe, you will probably find it advantageous to create buckets in the EU (Ireland) Region.

> **Important**
> The SOAP API does not support geographical constraints.

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

Amazon S3 supports the following Regions:

- **US Standard—**Uses Amazon S3 servers in the United States
  This is the default Region. The US Standard Region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps. To use this region, select US Standard as the

region when creating a bucket in the console. The US Standard Region provides eventual consistency for all requests.

- **US West (Oregon) Region—**Uses Amazon S3 servers in Oregon
  To use this Region, choose Oregon as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the US West (Oregon) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **US West (Northern California) Region—**Uses Amazon S3 servers in Northern California
  To use this Region, choose Northern California as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the US West (Northern California) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **EU (Ireland) Region—**Uses Amazon S3 servers in Ireland
  To use this Region, choose Ireland as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the EU (Ireland) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **Asia Pacific (Singapore) Region—**Uses Amazon S3 servers in Singapore
  To use this Region, choose Singapore as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the Asia Pacific (Singapore) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **Asia Pacific (Sydney) Region—**Uses Amazon S3 servers in Sydney
  To use this Region, choose Sydney as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the Asia Pacific (Sydney) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **Asia Pacific (Tokyo) Region—**Uses Amazon S3 servers in Tokyo
  To use this Region, choose Tokyo as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the Asia Pacific (Tokyo) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **South America (Sao Paulo) Region—**Uses Amazon S3 servers in Sao Paulo
  To use this Region, choose Sao Paulo as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the South America (Sao Paulo) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.

Regions are set at the bucket level. You specify a Region using the `LocationConstraint` bucket parameter. If you do not specify a Region, Amazon S3 hosts your buckets on servers in the US Standard Region.

> **Important**
> If you use a Region-specific endpoint to create a bucket, you *must* set the `LocationConstraint` bucket parameter to the same Region. For more information about Region-specific endpoints, see Request Endpoints (p. 13).

How to Specify a Bucket's Region

Use the following process to specify a bucket's Region.

Specifying a Bucket's Region

1	In a bucket creation request, set the `LocationContraint` parameter to a specific Region, for example, `CreateBucketConfiguration.LocationConstraint=us-west-1`

2	Optionally, when creating a bucket in the Northern California Region, you can also use a Region-specific endpoint in the request to create the bucket. For example, to match the Region specified in step 1, you would use the endpoint, `https://s3-us-west-1.amazonaws.com` (or `http://s3-us-west-1.amazonaws.com`). Using the Region-specific endpoint avoids the latency caused by the redirection of requests from US Standard to the Northern California Region. For more information, see Redirection (p. 85). (The EU (Ireland) Region does not have a Region-specific endpoint.) **Important** Even if you use a Region-specific endpoint in a request to create a Northern California Region bucket, you must set the value of *`LocationConstraint`* to the same Region.

Bucket Access

To access Amazon S3 buckets and objects that were created using `CreateBucketConfiguration`, you can use the virtual hosted-style request in all Regions. For example:

```
http://yourbucket.s3.amazonaws.com/yourobject
```

To use the path-style request, the bucket must be in the US Classic Region, or the bucket must be in the same Region as the endpoint in the request. For example:

```
http://s3.amazonaws.com/yourbucket/yourobject
```

Redirection

Amazon supports two types of redirects: temporary and permanent.

Temporary redirects automatically redirect users that do not have DNS information for the requested bucket. This occurs because DNS changes take time to propagate through the Internet. For example, if a user creates a bucket with a location constraint and immediately stores an object in the bucket, information about the bucket might not distribute throughout the Internet. Because the bucket is a sub domain of s3.amazonaws.com, Amazon S3 redirects it to the correct Amazon S3 location.

You can remove this (short lived) redirection latency by using a Region-specific endpoint in the bucket creation request. The *`LocationConstraint`* bucket parameter specifies the Region where the bucket will reside. Using the Region-specific endpoint is optional. The only Region you can do this in is US-West. For more information, see How to Specify a Bucket's Region (p. 84).

Transferring a Bucket's Contents to Another Region

Use the following process to transfer your bucket from one Region to another.

To transfer a bucket to another Region

1	Create a new Amazon S3 bucket in the Region you wish to transfer your data to.
2	Use the `copy` operation to transfer each of your objects from the source Region to the target Region. Bandwidth charges apply for this transfer. For more information, go to COPY Object.

Managing Bucket Website Configuration

Topics

- Managing Websites with the AWS SDK for Java (p. 86)
- Managing Websites with the AWS SDK for .NET (p. 89)
- Managing Websites with the AWS SDK for PHP (p. 93)
- Managing Websites with the REST API (p. 94)

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. 377). There are several ways you can manage your bucket's website configuration. AWS Management Console enables you to manage configuration without writing any code. You can programmatically create, update and delete the website configuration using the AWS SDK. The SDK provides wrapper classes around the Amazon S3 REST API; and if your application requires it, you can send the REST API requests directly from your application.

Managing Websites with the AWS Management Console

For more information, see Configure a Bucket for Website Hosting (p. 379).

Managing Websites with the AWS SDK for Java

The following tasks guide you through using the Java classes to manage website configuration to your bucket. For more information about the Amazon S3 website feature, see Hosting a Static Website on Amazon S3 (p. 377).

Managing Website Configuration

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	To add website configuration to a bucket, execute the `AmazonS3.setBucketWebsiteConfiguration` method. You need to provide the bucket name and the website configuration information including the index document and the error document names. You must provide the index document, however, the error document is optional. You provide website configuration information by creating a `BucketWebsiteConfiguration` object. To retrieve website configuration, execute the `AmazonS3.getBucketWebsiteConfiguration` method by providing the bucket name. To delete your bucket website configuration, execute the `AmazonS3.deleteBucketWebsiteConfiguration` method by providing the bucket name. After you remove the website configuration, the bucket is no longer available via the website endpoint. For more information, see see Website Endpoints (p. 378).

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                 myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
// Add website configuration.
s3Client.setBucketWebsiteConfiguration(bucketName,
      new BucketWebsiteConfiguration(indexDoc , errorDoc));

// Get website configuration.
BucketWebsiteConfiguration bucketWebsiteConfiguration =
         s3Client.getBucketWebsiteConfiguration(bucketName);
```

```

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

Example

The following Java code example adds a website configuration to the specified bucket, retrieves it and deletes the website configuration. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.IOException;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;


public class S3Sample {
 private static String bucketName = "mvphp1";
 private static String indexDoc = "index.html";
 private static String errorDoc = "error.html";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
           S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

        try {
         // Get existing website configuration, if any.
            getWebsiteConfig(s3Client);

      // Set new website configuration.
      s3Client.setBucketWebsiteConfiguration(bucketName,
         new BucketWebsiteConfiguration(indexDoc, errorDoc));

      // Verify (Get website configuration again).
            getWebsiteConfig(s3Client);

            // Delete
            s3Client.deleteBucketWebsiteConfiguration(bucketName);

         // Verify (Get website configuration again)
              getWebsiteConfig(s3Client);



        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
              " means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means"+

              " the client encountered " +
                    "a serious internal problem while trying to " +
                    "communicate with Amazon S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

 private static BucketWebsiteConfiguration getWebsiteConfig(
                                              AmazonS3 s3Client) {
  System.out.println("Get website config");

  // 1. Get website config.
  BucketWebsiteConfiguration bucketWebsiteConfiguration =
     s3Client.getBucketWebsiteConfiguration(bucketName);
  if (bucketWebsiteConfiguration == null)
  {
   System.out.println("No website config.");
  }
  else
  {
       System.out.println("Index doc:" +
         bucketWebsiteConfiguration.getIndexDocumentSuffix());
       System.out.println("Error doc:" +
         bucketWebsiteConfiguration.getErrorDocument());
  }
  return bucketWebsiteConfiguration;
 }
}
```

Managing Websites with the AWS SDK for .NET

The following tasks guide you through using the .NET classes to manage website configuration on your bucket. For more information about the Amazon S3 website feature, see Hosting a Static Website on Amazon S3 (p. 377).

Managing Bucket Website Configuration

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	To add website configuration to a bucket, execute the `AmazonS3.PutBucketWebsite` method. You need to provide the bucket name and the website configuration information including the index document and the error document names. You must provide the index document, however, the error document is optional. You provide this information by creating a `PutBucketWebsiteRequest` object. To retrieve website configuration, execute the `AmazonS3.GetBucketWebsite` method by providing the bucket name. To delete your bucket website configuration, execute the `AmazonS3.DeleteBucketWebsite` method by providing the bucket name. After you remove the website configuration, the bucket is no longer available via the website endpoint. For more information, see see Website Endpoints (p. 378).

The following C# code sample demonstrates the preceding tasks.

```
AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);
// Add website configuration.
PutBucketWebsiteRequest putRequest =
   new PutBucketWebsiteRequest()
        .WithBucketName(bucketName)
        .WithWebsiteConfiguration(new WebsiteConfiguration()
        .WithIndexDocumentSuffix(indexDocumentSuffix)
        .WithErrorDocument(errorDocument));

client.PutBucketWebsite(putRequest);

// Get bucket website configuration.
GetBucketWebsiteRequest getRequest =
   new GetBucketWebsiteRequest()
     .WithBucketName(bucketName);

GetBucketWebsiteResponse getResponse =
  client.GetBucketWebsite(getRequest);
// Print configuration data.
Console.WriteLine("Index document: {0}", getResponse.WebsiteConfiguration.Index
DocumentSuffix);
Console.WriteLine("Error document: {0}", getResponse.WebsiteConfiguration.Error
Document);

// Delete website configuration.
DeleteBucketWebsiteRequest deleteRequest =
    new DeleteBucketWebsiteRequest()
        .WithBucketName(bucketName);

client.DeleteBucketWebsite(deleteRequest);
```

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 Testing the .NET Code Examples (p. 436).

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.addwebsiteconfiguration
{
    class S3Sample
    {
        static string bucketName          = "*** Provide existing bucket name
***";
        static string indexDocumentSuffix = "*** Provide index document name
***";
        static string errorDocument       = "*** Provide error document name
***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                   AddWebsiteConfig(bucketName, indexDocumentSuffix, errorDoc
ument);
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void AddWebsiteConfig(string bucketName,
                                     string indexDocumentSuffix,
                                     string errorDocument)
        {
            try
            {
                PutBucketWebsiteRequest putRequest =
                    new PutBucketWebsiteRequest()
                    .WithBucketName(bucketName)
                    .WithWebsiteConfiguration(new WebsiteConfiguration()
                        .WithIndexDocumentSuffix(indexDocumentSuffix)
                        .WithErrorDocument(errorDocument));
                client.PutBucketWebsite(putRequest);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                    || amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))

                {
                    Console.WriteLine("Check the provided AWS Credentials.");
                    Console.WriteLine(
                        "Sign up for service at http://aws.amazon.com/s3");
                }
                else
                {
                    Console.WriteLine(
                       "Error:{0}, occurred when adding website configuration.
 Message:'{1}",
                      amazonS3Exception.ErrorCode, amazonS3Exception.Message);

                }
            }
        }

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }


            return true;
        }

    }
}
```

Managing Websites with the AWS SDK for PHP

The following tasks guide you through using the PHP classes to manage website configuration of your bucket. For more information about the Amazon S3 website feature, see Hosting a Static Website on Amazon S3 (p. 377).

Managing Bucket Website Configuration

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	To configure a bucket as a website, execute the `AmazonS3.create_website_config` method. You need to provide the bucket name and the website configuration information including the index document and the error document names. If you don't provide these document names, this method adds the `index.html` and `error.html` default names to the website configuration. You must verify that these documents are present in the bucket. To retrieve existing bucket website configuration, execute the `AmazonS3.get_website_config` method by passing the bucket name as a parameter. To delete website configuration from a bucket, execute the `AmazonS3.delete_website_config` method by passing the bucket name as a parameter. If you remove the website configuration, the bucket is no longer accessible from the website endpoints.

The following PHP code sample demonstrates the preceding tasks.

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';
header('Content-Type: text/plain; charset=utf-8');

$bucket = '*** Provide bucket name ***';
$indexDocument = '*** Provide index document name ****';
$errorDocument = '*** Provide error document name ***';

// Instantiate the class
$s3 = new AmazonS3();

// 1) Add website config. to bucket.
$response = $s3->create_website_config($bucket,
                                       array(
                                         'indexDocument' => $indexDocument,
                                         'errorDocument' => $errorDocument));

// 2) Retrieve website configuration.
$response = $s3->get_website_config($bucket);
header('Content-Type: text/plain; charset=utf-8');

// 3) Delete website configuration.
$response = $s3->delete_website_config($bucket);
```

Example

The following PHP code example first adds a website configuration to the specified bucket. The `create_website_config` method explicitly provides the index document and error document names. The sample 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. 377). For instructions on how to create and test a working sample, see Using the AWS SDK for PHP (p. 437).

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';
header('Content-Type: text/plain; charset=utf-8');

$bucket = '*** Provide bucket name ***';
$indexDocument = '*** Provide index document name ****';
$errorDocument = '*** Provide error document name ***';

// Instantiate the class
$s3 = new AmazonS3();

// 1) Add website config. to bucket.
$response = $s3->create_website_config($bucket,
                                       array(
                                         'indexDocument' => $indexDocument,
                                         'errorDocument' => $errorDocument));
// Success?
 var_dump($response->isOK());

// 2) Retrieve website configuration.
$response = $s3->get_website_config($bucket);
header('Content-Type: text/plain; charset=utf-8');
print_r($response);

// 3) Delete website configuration.
$response = $s3->delete_website_config($bucket);
var_dump($response->isOK());
```

Managing Websites with the REST API

Topics

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

Requester Pays Buckets

Topics

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 data sets, 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.

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 and GET 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.

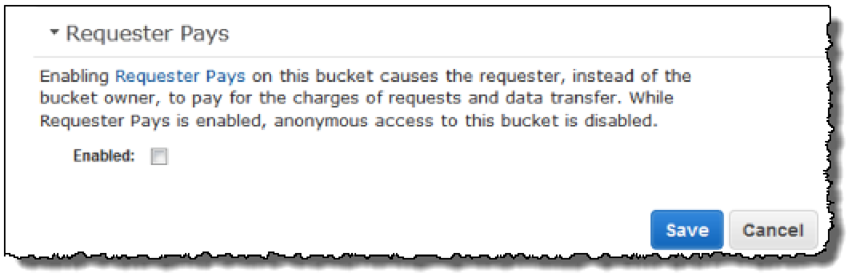
- Anonymous requests
- BitTorrent
- SOAP requests
- You cannot use a Requester Pays bucket as the target bucket for end user logging, or vice versa. However, you can turn on end user logging on a Requester Pays bucket where the target bucket is a non Requester Pays bucket.

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** checkbox.



Enabling Requester Pays on this bucket causes the requester, instead of the bucket owner, to pay for the charges of requests and data transfer. While Requester Pays is enabled, anonymous access to this bucket is disabled.

Enabled: ☐

Configure Requester Pays Using REST API

Topics

Setting the requestPayment Bucket Configuration

The bucket owner and 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. If you don't, the bucket, by default, is a non-Requester Pays bucket.

You use the value, `BucketOwner`, to revert Requester Pays buckets to regular buckets. Typically, you would use `BucketOwner` when uploading data to the Amazon S3 bucket, then 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
```

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

You can freely 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, for the configuration value to take effect.

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

Retrieving requestPayment Configuration

You can determine the *`Payer`* value 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 demonstrates 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 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.
> Make sure to include *`x-amz-request-payer`* and its value in your signature calculation. For more information, see Constructing the CanonicalizedAmzHeaders Element (p. 51).

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 `Access Denied` errors for requests trying to get objects from Requester Pays buckets. For more information, go to Error Responses.

DevPay and Requester Pays

You can use Amazon DevPay to sell the content stored in your Requester Pays bucket. For more information, go to "Using Amazon S3 Requester Pays with DevPay," in the Using Amazon S3 Requester Pays with DevPay.

Charge Details

The charge for successful Requester Pays requests is straight forward: the requester pays for data transfer and the request; the bucket owner pays for the data storage. However, the bucket owner is charged for the request if:

- The requester doesn't include the parameter *`x-amz-request-payer`* in the header (GET 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 Access Control (p. 277).

Billing and Reporting of Buckets

Fees for object storage and network data transfer are always billed to the owner of the bucket that contains the object unless the bucket was created as a Requester Pays bucket.

The reporting tools available at the Amazon Web Services developer portal organize your Amazon S3 usage reports by bucket. For more information, go to Amazon S3 Pricing page.

Cost Allocation Tagging

Cost allocation tagging allows you to label S3 buckets so you can more easily track their cost against projects or other criteria. .

Use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see Cost Allocation and Tagging in *About AWS Account Billing*.

A cost allocation tag is a name-value pair that you define and associate with an S3 Bucket. We recommend that you use a consistent set of tag keys to make it easier to track costs associated with your S3 Buckets.

Each S3 Bucket has a tag set, which contains all the tags that are assigned to that Bucket. A tag set can contain as many as ten tags, or it can be empty.

If you add a tag that has the same key as an existing tag on a Bucket, the new value overwrites the old value.

AWS does not apply any semantic meaning to your tags; tags are interpreted strictly as character strings. AWS does not automatically set any tags on Buckets.

You can use the Amazon S3 console, the CLI, or the S3 API to add, list, edit, or delete tags. For more information about creating tags in the console, go to Managing Cost Allocation Tagging in the *Amazon S3 Console User Guide*.

The following list describes the characteristics of a cost allocation tag.

- The tag key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "aws:". The string may only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$").

- The tag value is a required string value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "aws:". The string may only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$").

  Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of project/Trinity and cost-center/Trinity.

Bucket Configuration Errors

The following list shows the errors Amazon S3 can return in response to bucket configuration requests.

- MalformedXML
- MissingRequestBodyError

Working with Amazon S3 Objects

Topics

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. 101)
- **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. 103).
- **Value—**The content that you are storing.
  An object value can be any sequence of bytes. Objects can range from zero to 5 TB in size. For more information, see Uploading Objects (p. 156).
- **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. 101).
- **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. 103).

- **Access Control Information—**You can controls 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 Access Control (p. 277).

Object Key and Metadata

Each Amazon S3 object has data, a key, and metadata. When you create an object you specify the key name. This key name uniquely identifies the object in the bucket. For example, in Amazon S3 console (see AWS Management Console), when you highlight a bucket, a list of objects in your bucket appear. 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.

In addition to the key, each Amazon S3 object has metadata. It is a set of name-value pairs. There are two kinds of metadata: system metadata and user-defined metadata.

System-Defined 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:

- Metadata such as object creation date is system controlled where only Amazon S3 can modify the value.
- Other system metadata such as the storage class configured for the object, whether object has server-side encryption enabled are examples of system metadata whose values you control. If you have your bucket configured as a website, sometimes you might want to redirect a page request to another page or an external URL. In this case a web page 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. For more information about storage class and server-side encryption, see Using Data Encryption (p. 339).

The following table provides a list of system-defined metadata and whether you can update it.

Name	Description	Can User Modify the Value?
Date	Object creation date.	No
Content-Length	Object size in bytes.	No
Last-Modified	Date the object was last modified.	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, that is, the object data is encrypted at rest. For more information, see Using Server-Side Encryption (p. 340).	Yes

Name	Description	Can User Modify the Value?
x-amz-version-id	Object version. When you enable versioning on a bucket, Amazon S3 assigns version number to objects added to the bucket. For more information, see Using Versioning (p. 355).	No
x-amz-delete-marker	In a bucket that has version enabled, this boolean marker that indicates whether the object is a delete marker.	No
x-amz-storage-class	Storage class used for storing the object.	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 Configuring a Web Page Redirect (p. 388).	Yes

Storage Classes

Each Amazon S3 object is associated with a storage class. Amazon S3 supports the following storage classes:

- Standard — The Standard storage class provides 99.999999999% durability and 99.99% availability of objects over a given year. It is designed to sustain the concurrent loss of data in two facilities.
- Reduced Redundancy Storage (RRS) — The RRS storage class reduces costs by storing noncritical, reproducible data at lower levels of redundancy than the Standard storage class. It provides 99.99% durability and 99.99% availability of objects over a given year. This durability level corresponds to an average annual expected loss of 0.01% of objects
- Glacier — The Glacier storage class is suitable for archiving data, where data access is infrequent and a retrieval time of several hours is acceptable. The Glacier storage class uses the very low-cost Amazon Glacier storage service, but you still manage objects in this storage class through Amazon S3.

Note
You cannot associate an object with the Glacier storage class as you upload it. You transition existing Amazon S3 objects to the Glacier storage class by using lifecycle management. For more information, see Object Lifecycle Management (p. 106).

User-Defined Metadata

When uploading an object, you can also assign metadata to the object. You provide this optional information as a name, value pair when you send a PUT or POST request to create the object. When uploading 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 uploading 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.

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 the unprintable metadata entries.

Each name, value pair must conform to US-ASCII when using REST and UTF-8 when 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. User-defined metadata is a set of key-value pairs. 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.

Object Subresources

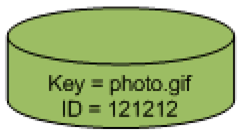
Amazon S3 defines a set of subresources associated with buckets and objects. Subresources are subordinates to objects; that is, subresources do not exist on their own, they are always associated with some other entity, such as an object or a bucket.

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 `acl` identifies the object owner as having full control over the object. You can retrieve an object ACL or replace it with updated list of grants. Any update to an ACL requires you to replace the existing ACL. For more information about ACLs, see Using ACLs (p. 323)
torrent	Amazon S3 supports the BitTorrent protocol. Amazon S3 uses the `torrent` subresource to return the torrent file associated with the specific object. To retrieve a torrent file, you specify the `torrent` subresource in your GET request. Amazon S3 creates a torrent file and returns it. You can only retrieve the `torrent` subresource, you cannot create, update or delete the `torrent` subresource. For more information, see Using BitTorrent with Amazon S3 (p. 409).

Object Versioning

Versioning enables you to keep multiple versions of an object in one bucket. You must explicitly enable versioning on your bucket. By default versioning is disabled. Regardless of whether you have enabled versioning on your bucket or not, each object has a version ID. If you have not enabled versioning on your bucket, then Amazon S3 sets the version ID value null. If you have enabled versioning on your bucket, Amazon S3 assigns a unique version ID value for the object.



Versioning enables you to keep multiple versions of an object in one bucket, for example, `my-image.jpg` (version 111111) and `my-image.jpg` (version 222222). You might enable versioning to recover from unintended overwrites and deletions or to archive objects so that you can retrieve previous versions of them.

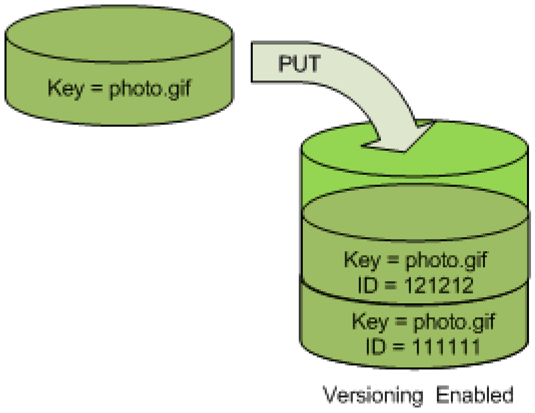
Note
The SOAP API does not support Versioning.

When you enable versioning on a bucket, existing objects, if any, 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.

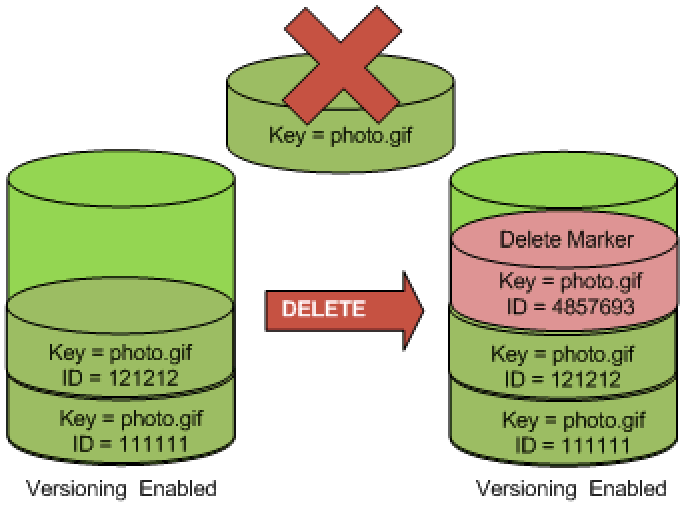
Note
For simplicity, we will use much shorter IDs in all our examples.

When you `PUT` an object in a version-enabled bucket, the old 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.

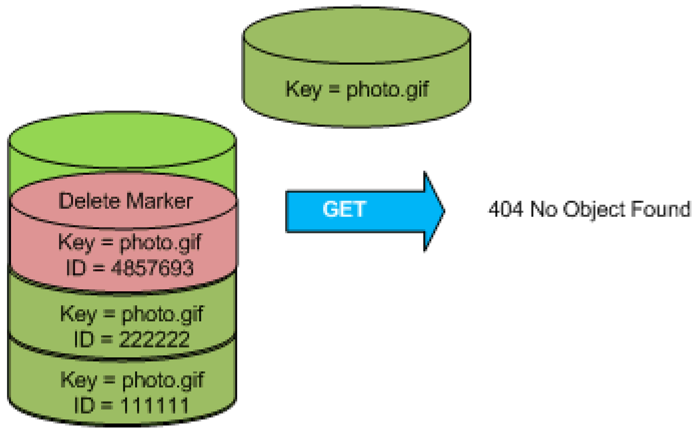


This functionality prevents you from accidentally overwriting or deleting objects and affords you the opportunity to retrieve a previous version of an object.

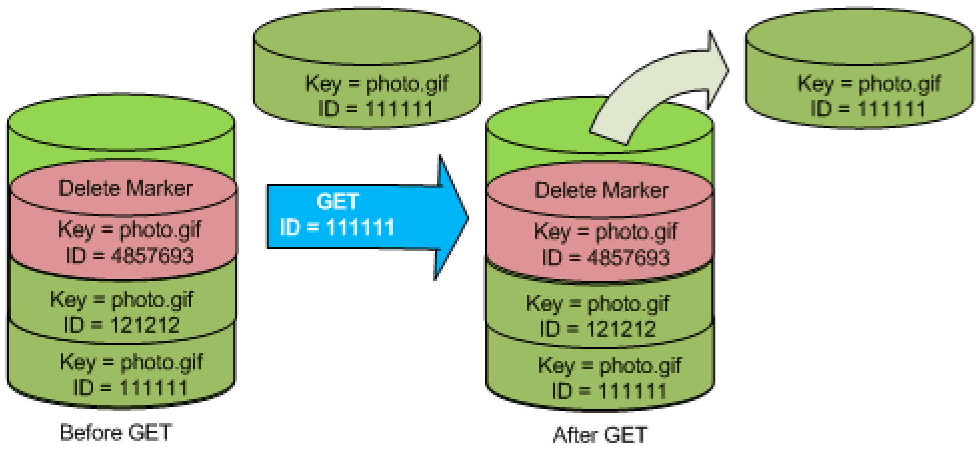
When you `DELETE` an object, all versions remain in the bucket and Amazon S3 inserts a delete marker, as shown in the following figure.



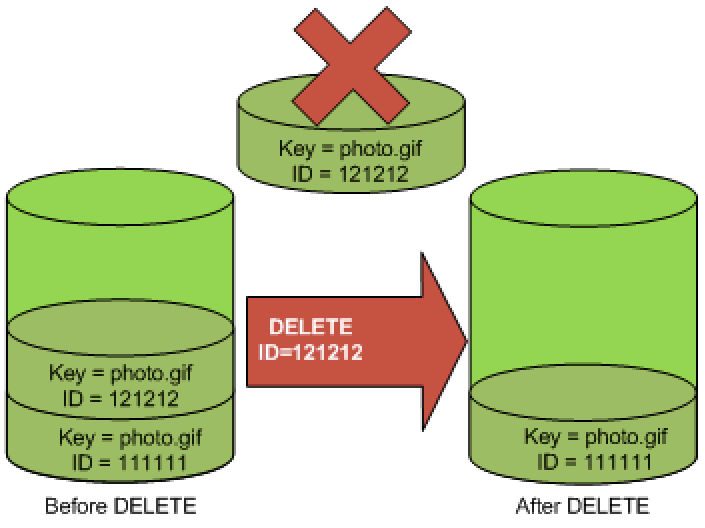
The delete marker becomes the latest version of the object. By default, `GET` requests retrieve the most recently stored version. Performing a simple `GET Object` request when the latest version is a delete marker returns a `404 Not Found` error, as shown in the following figure.



You can, however, `GET` an older 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 latest 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. 356).

For more information, see Using Versioning (p. 355).

Object Lifecycle Management

Topics

Lifecycle management defines how Amazon S3 manages objects during their lifetime.

Some objects that you store in an Amazon S3 bucket might have a well-defined lifecycle:

- If you are uploading periodic logs to your bucket, your application might need these logs for a week or a month after creation, and after that you might want to delete them.
- Some documents are frequently accessed for a limited period of time. After that, you might not need real-time access to these objects, but your organization might require you to archive them for a longer period and then optionally delete them later. Digital media archives, financial and healthcare records, raw genomics sequence data, long-term database backups, and data that must be retained for regulatory compliance are some kinds of data that you might upload to Amazon S3 primarily for archival purposes.

For such objects, you can define rules that identify the affected objects, a timeline, and specific actions you want Amazon S3 to perform on the objects.

Amazon S3 manages object lifetimes with a lifecycle configuration, which is assigned to a bucket and defines rules for individual objects. Each rule in a lifecycle configuration consists of the following:

- A object key prefix that identifies one or more objects to which the rule applies.
- An action or actions that you want Amazon S3 to perform on the specified objects.
- A date or a time period, specified in days since object creation, when you want Amazon S3 to perform the specified action.

You can add these rules to your bucket using either the Amazon S3 console or programmatically.

Lifecycle Configuration Rules

A rule can apply to a single object or multiple objects whose key name begins with the prefix specified in the rule. For example, suppose you have the following objects:

logs/day1

logs/day2

logs/day3

ExampleObject.jpg

If you specify `ExampleObject.jpg` as the prefix, the rule applies to the specific object. If you specify `logs/` as the prefix, the rule applies to the three objects whose key name begins with the string "logs/".

A rule can specify the following actions:

- Transition— When a specified date or time period in the object’s lifetime is reached, Amazon S3 sets the storage class to Glacier. For more information about storage classes, see Storage Classes (p. 102).

  For example, you can set a rule for Amazon S3 to transition "MyArchiveObject.jpg" to the Glacier storage class 30 days after creation or to transition all objects with the key prefix "glacierobjects/" to the Glacier storage class a year after creation.

- Expiration—When the specified time period is reached in the object's lifetime, Amazon S3 deletes it.

  For example, you can set a rule with an expiration action so that Amazon S3 will delete objects with the key prefix "log2012-01-01" 30 days after creation.

  Expiration applies to all Amazon S3 objects, including those that are archived in Amazon Glacier.

You can specify a date or a time period in days since object creation when the action will be taken on the specified object or objects.

- When you specify a number of days, 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 on 1/15/2012 10:30 a.m. UTC and you specify 3 days in a transition rule, then the transition date of the object would be calculated as 1/19/2012 00:00 UTC.

- When you specify a date in a rule, the specified action is executed on the specified date. That is, Amazon S3 transitions or expires the objects on the specified date.

These rules are available as a `lifecycle` subresource on the bucket. The following is an example lifecyle configuration:

```
<LifecycleConfiguration>
  <Rule>
    <ID>Example Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

A lifecycle configuration can contain as many as 1000 rules. The preceding rule defines a Transition action and an Expiration action. The rule applies to all objects whose key name starts with the prefix projectdocs/. The rule requests Amazon S3 to transition objects to the Glacier storage class 365 days after creation and to delete the objects 3650 days after creation.

Important
In the preceding example, the time period specified in both the transition and expiration actions is relative to *the date the objects are created*.

There is usually some lag before a new or updated lifecycle configuration is fully propagated to all the Amazon S3 systems. Expect some delay before lifecycle configuration fully takes effect.

Note
If your bucket is version-enabled or versioning is suspended, you cannot add a lifecycle configuration.

When you add a lifecycle configuration to a bucket, the configuration rules apply to both existing objects and to objects that you add later. For example, if you add a lifecycle configuration rule with an expiration action today 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.

Before You Decide to Archive Objects

The Transition action in the lifecycle configuration rule enables you to transition objects to the Glacier storage class—that is, archive data to Amazon Glacier. Before you decide to archive objects, note the following:

- Objects in the Glacier storage class 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 Glacier.

  Note that object restoration from an archive can take from three to five hours.
- The transition action allows only one-way transition to the Glacier storage class.

  You cannot use a lifecycle configuration rule to covert a Glacier object to a Standard or Reduced Redundancy Storage (RRS) object. If you want to change the storage class of an already archived object to either Standard or RRS, you must use the restore operation to make a temporary copy first. Then use the copy operation to overwrite the object as the Standard or the RRS object.

- Glacier objects are visible and available only through Amazon S3, not through Amazon Glacier

  Amazon S3 stores the archived objects in Amazon Glacier; however, these are Amazon S3 objects, and you can access them only by using the Amazon S3 console or the API. You cannot access the archived objects through the Amazon Glacier console or the API.
- A rule with an empty key prefix applies to all the objects in the bucket

  If you specify an empty prefix, the rule applies to all objects in the bucket. So if the rule specifies a transition action with an empty prefix, you are requesting Amazon S3 to archive all the objects in the bucket at a specific period in the objects' lifetime.

For more information, see Object Archival (Transition Objects to the Glacier Storage Class) (p. 112). For information about Amazon S3 storage classes, see Storage Classes (p. 102).

Before You Decide to Expire Objects

The Expiration action in the lifecycle configuration rule enables you to request Amazon S3 to delete objects. Before you decide to set this action in the lifecycle configuration, note the following:

- The Expiration action deletes objects

  You might have objects in Amazon S3 or archived to Amazon Glacier. No matter where these objects are, Amazon S3 will delete them. You will no longer be able to access these objects.
- A rule with an empty key prefix applies to all the objects in the bucket

  If you specify an empty prefix, the rule applies to all objects in the bucket. So if the rule specifies an expiration action with an empty prefix, you are requesting Amazon S3 to delete all the objects in the bucket at a specific period in the objects' lifetime.

Specifying a Lifecycle Configuration

You can set a lifecycle configuration on a bucket either programmatically or by using the Amazon S3 console.

Specify a Lifecycle Configuration Using the Amazon S3 Console

With the Amazon S3 console, you can configure as many as 1000 lifecycle rules for objects in a bucket. For more information, see Manage Object Lifecycle Using the AWS Management Console (p. 114).

Specify a Lifecycle Configuration Programmatically

The Amazon S3 provides API actions to add a lifecycle configuration to the bucket. Amazon S3 stores the lifecycle configuration in the `lifecycle` subresource on your bucket. For more information, see Manage Object Lifecycle Using the REST API (p. 124).

You can also use the AWS SDKs to add a lifecycle configuration to your bucket.

Whether you are using the Amazon S3 API or the AWS SDK to manage your lifecycle configuration, the process for defining it is the same. A lifecycle configuration is specified in XML. It contains a set of rules, as shown in the following fragment:

```
<LifecycleConfiguration>
  <Rule>
      ...
  </Rule>
  <Rule>
      ...
  </Rule>
   ...
</LifecycleConfiguration>
```

Each rule identifies an object or group of objects by key prefix, and it specifies an action for Amazon S3 to perform at a specified time or after a specified time period following object creation. The following XML elements define a rule:

- **<ID> element—**A unique identifier for the rule.
- **<Status> element—** Enabled or Disabled. Amazon S3 will not evaluate any disabled rules.
- **<Prefix> element—**A key prefix that identifies an object or objects to which the rule applies. If you do not supply a Prefix value, the rule applies to all objects in the bucket.
- **Action elements—**In a single rule, you can define either or both the following actions, although in a rule an action can appear at most once.
  - **<Expiration> action—**Defines in terms of a specific date or number of days since creation when the object lifetime expires. It can have <Date> or <Days> element children.
  - **<Transition> action—**Defines in terms of date or days since creation when the object transition to the Glacier storage class. It can have <Date> or <Days> element children.

The following are examples of the lifecycle configuration. For more information about the REST API or the AWS SDKs, click link provided at the beginning of this section.

Example 1: Specify a Rule

The following lifecycle configuration has a rule with two actions. It requests Amazon S3 to transition objects with key prefix `projectdocs/` to Amazon Glacier 365 days after creation and delete those objects 3650 days after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition and Expiration Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Instead of a time period, you can specify a date for each action; however, you cannot use both a date and a time period in the same rule.

In the preceding example, there is one rule with two actions. You can achieve the same result by defining two rules for one object, each of which performs one of the actions:

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
  </Rule>
  <Rule>
    <ID>Expiration Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Example 2: Specify a Rule that Applies to All Objects in the Bucket

The following lifecycle configuration specifies an empty prefix and zero transition days. If you don't specify a prefix, the rule applies to all objects in the bucket. Because the Days element is set to 0, all objects are immediately eligible for archival to Amazon Glacier, and Amazon S3 archives these objects using processes that run periodically.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Archive all object immediately upon creation</ID>
    <Prefix></Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

Example 3: Disabling a Rule

The following lifecycle configuration specifies two rules; however, one of them is disabled. Amazon S3 will not perform any action specified in a rule that is disabled.

```
<LifecycleConfiguration>
  <Rule>
    <ID>30 days log objects expire rule</ID>
    <Prefix>logs/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>30</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>1 year documents expire rule</ID>
    <Prefix>documents/</Prefix>
    <Status>Disabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Example 4: Overlapping Rules Not Allowed

Rules cannot overlap. For example, the following lifecycle configuration has a rule that sets objects with the prefix "documents/" to expire after 30 days. In another rule, objects with the prefix "documents/2011" are set to expire after 365 days. In this case, Amazon S3 returns an error message.

```
<LifecycleConfiguration>
  <Rule>
    <ID>111</ID>
    <Prefix>documents/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>30</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>222</ID>
    <Prefix>documents/2011</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

For more information about the REST API or the AWS SDKs, click link provided at the beginning of this section.

Object Archival (Transition Objects to the Glacier Storage Class)

Topics

Each object in Amazon S3 is associated with a storage class. When you upload an object, by default Amazon S3 associates it with the Standard storage class, in which Amazon S3 maintains redundant copies to ensure high durability. For non-critical, reproducible data, you can use the Reduced Redundancy Storage (RRS) option, in which Amazon S3 uses lower levels of redundancy than it does for the Standard storage class. RRS provides a cost-effective, highly available solution. Both the Standard and RRS objects are highly available in real-time. For objects that you do not need to access in real time, Amazon S3 also offers the Glacier storage class. This storage class is suitable for objects stored primarily for archival purposes.

You can specify the Standard or RRS storage class when you upload objects; however, you cannot assign the Glacier storage class when uploading objects. Instead, you transition existing objects to the Glacier storage class by using the Object Lifecycle Management (p. 106). Amazon S3 will archive objects for you and associate those objects with the Glacier storage class according to rules that you define.

> **Note**
> When you transition objects to the Glacier storage class, Amazon S3 internally uses Amazon Glacier for durable storage at lower cost. You should use the Glacier storage class only for long-term archival of data that is infrequently accessed. For information about pricing, go to the Pricing section of the *Amazon S3 product detail page*. Amazon S3 objects that are associated with the Glacier object class are visible and available only in Amazon S3.

The lifecycle configuration enables a one-way transition to Glacier. To change the storage class from Glacier to Standard or RRS, you must restore the object, as discussed in the following section and then make a copy of the restored object.

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. Restore jobs typically complete in three to five hours, and so it is important that you archive only objects that you will not need to access in real-time.

After you receive a temporary copy of the restored object, the object's storage class remains Glacier. Amazon S3 associates the temporary copy with the Reduced Redundancy Storage (RRS) class. Note that when you restore an archive you are paying for both the archive and a copy you restored temporarily. For information about pricing, go to the Pricing section of the *Amazon S3 product detail page*.

You can restore an object copy programmatically or by using the Amazon S3 console. You can process only one restore request at a time. 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.

Restoring Glacier Objects by Using Amazon S3 Console

For information about restoring Glacier objects by using the Amazon S3 console, see Restore an Object Using Amazon S3 Console (p. 270).

Restoring Glacier Objects Programmatically

You can restore Glacier objects programmatically directly from your application by using either the AWS SDKs or the Amazon S3 API. In either case, the following XML is sent over the wire to specify the restoration period.

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

When you use the AWS SDKs, the Amazon S3 API provides appropriate wrapper libraries to simplify your programming tasks; however, when the request is sent over the wire, the SDK sends the preceding XML in the request body.

For information about the REST API, see Manage Object Lifecycle Using the REST API (p. 124).

Object Expiration

Topics

Some objects that you store in an Amazon S3 bucket might have a well-defined lifetime. For example, you might be uploading periodic logs to your bucket, but you might need to retain those logs for a specific amount of time. You can use using the Object Lifecycle Management (p. 106) to specify a lifetime for objects in your bucket; when the lifetime of an object expires, Amazon S3 queues the objects for deletion.

Important
Note that upon expiration Amazon S3 deletes these objects, no matter whether they are in S3 or archived to Amazon Glacier. Even if the objects are archived, Amazon S3 will remove archived objects.

Note
When an object reaches the end of its lifetime, Amazon S3 queues it for removal and removes it asynchronously. There may be a lag 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.

Note
Whenever you overwrite an object, Amazon S3 resets its creation date and time to the date and time you updated the object. Accordingly, Amazon S3 also updates the object's expiration date from this new creation date and time.

To find when an object is scheduled to expire, you can use the GET Object or the HEAD object APIs. These APIs return response headers that provide object expiration information. For more information, go to HEAD Object and GET Object in the *Amazon Simple Storage Service API Reference*.

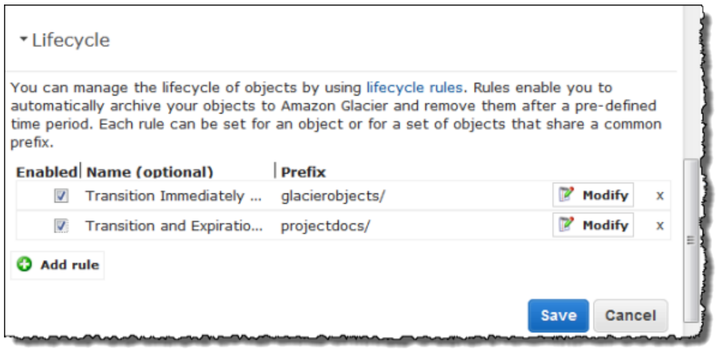
To find when an object was removed by Amazon S3, you can use Amazon S3 server access logs. The value "S3.EXPIRE.OBJECT" of the operation field in the log record indicates that Amazon S3 initiated the operation. For more information, see Server Access Logging (p. 420).

You can delete objects by explicitly calling the DELETE Object action or by creating a lifecycle configuration so that Amazon S3 will remove them for you. If you want to block users or accounts from removing or deleting objects from your bucket, you must deny them permission to call thes`s3:DeleteObject`, `s3:DeleteObjectVersion`, and `s3:PutLifecycleConfiguration` actions.

For examples, see Object Lifecycle Management (p. 106).

Manage Object Lifecycle Using the AWS Management Console

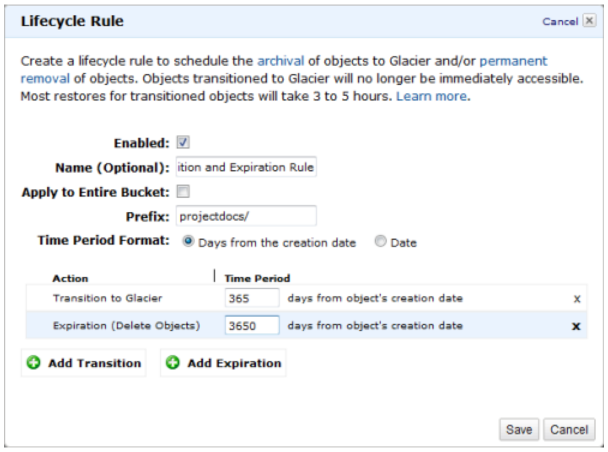
You can specify lifecycle rules (see Object Lifecycle Management (p. 106)) on a bucket using the Amazon S3 console. In the console, the bucket **Properties** provides a **Lifecycle** tab as shown in the following example screen shot. It shows bucket with two rules and both rules are enabled. You can click **Modify** to view the rule details or click **Add rule** to add a new rule.



You can manage the lifecycle of objects by using lifecycle rules. Rules enable you to automatically archive your objects to Amazon Glacier and remove them after a pre-defined time period. Each rule can be set for an object or for a set of objects that share a common prefix.

Enabled	Name (optional)	Prefix
✓	Transition Immediately ...	glacierobjects/
✓	Transition and Expiratio...	projectdocs/

The **Lifecycle Rule** wizard shown in the following example shows a rule's details. It shows that the rule applies to objects with key prefix "projectdocs/". The rule defines two actions for Amazon S3; the **Transition** action instructs Amazon S3 to transition objects to the Glacier storage class a year after creation, and the **Expiration** action instructs Amazon S3 to delete the objects 10 years after creation.

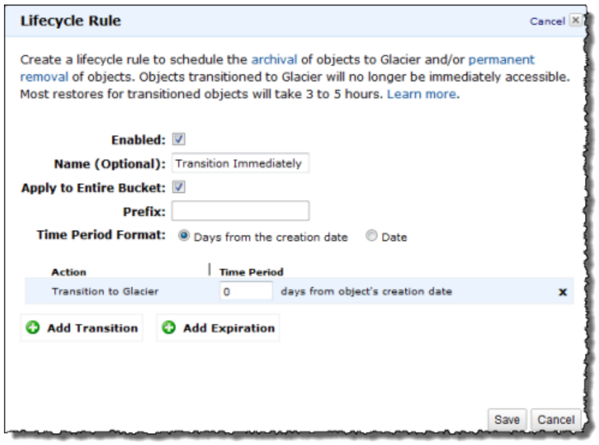


Create a lifecycle rule to schedule the archival of objects to Glacier and/or permanent removal of objects. Objects transitioned to Glacier will no longer be immediately accessible. Most restores for transitioned objects will take 3 to 5 hours. Learn more.

Enabled: ✓
Name (Optional): ition and Expiration Rule
Apply to Entire Bucket: ☐
Prefix: projectdocs/
Time Period Format: [● Days from the creation date] ○ Date

Action	Time Period	
Transition to Glacier	365	days from object's creation date
Expiration (Delete Objects)	3650	days from object's creation date

The following is an interesting rule. You might create a bucket for Glacier storage class objects and set a rule for Amazon S3 to immediately transition objects to Glacier storage class as shown in the following screen shot. Note the rule sets the time period to 0 days indicating immediate transition upon creation. The rule also sets empty **Prefix** indicating the rule applies to all objects in the bucket.



Create a lifecycle rule to schedule the archival of objects to Glacier and/or permanent removal of objects. Objects transitioned to Glacier will no longer be immediately accessible. Most restores for transitioned objects will take 3 to 5 hours. Learn more.

Enabled: ✓
Name (Optional): Transition Immediately
Apply to Entire Bucket: ✓
Prefix:
Time Period Format: [● Days from the creation date] ○ Date

Action	Time Period	
Transition to Glacier	0	days from object's creation date

Step-by-Step Instructions

For step-by-step instructions, see Managing Lifecycle Configuration in the *Amazon S3 Console User Guide*.

Manage Object Lifecycle Using the AWS SDK for Java

You can use the AWS SDK for Java to manage lifecycle configuration on a bucket. For more information about managing lifecycle configuration, see Object Lifecycle Management (p. 106).

This section provides sample code snippets for the tasks in the following table, followed by a complete sample program listing.

To add a lifecycle configuration to a bucket

1. Create a `BucketLifecycleConfiguration.Rule` object to specify a rule.
2. Create a `BucketLifecycleConfiguration` object using the rules.
3. Add the lifecycle configuration to the bucket by calling `s3Client.setBucketLifecycleConfiguration()`.

The following Java code snippet demonstrates the preceding tasks. The lifecycle configuration defines one rule ("Archive and delete rule") specifying two actions for Amazon S3 to perform on all objects with key prefix "projectdocs/".

```
Transition transToArchive = new Transition()
    .withDays(365)
    .withStorageClass(StorageClass.Glacier);

BucketLifecycleConfiguration.Rule ruleArchiveAndExpire = new BucketLifecycleCon
figuration.Rule()
    .withId("Archive and delete rule")
    .withPrefix("projectdocs/")
    .withTransition(transToArchive)
    .withExpirationInDays(3650)
    .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

List<BucketLifecycleConfiguration.Rule> rules = new ArrayList<BucketLifecycleCon
figuration.Rule>();
rules.add(ruleArchiveAndExpire);

BucketLifecycleConfiguration configuration = new BucketLifecycleConfiguration()

    .withRules(rules);

// Save configuration.
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);
```

To update an existing lifecycle configuration

- When you add a lifecycle configuration to a bucket, any existing lifecycle configuration is replaced. To update existing lifecycle configuration, you must first retrieve the existing lifecycle configuration, make changes and then add the revised lifecycle configuration to the bucket as shown in the following Java code snippet.

This snippet gets an existing configuration and adds a new rule with the ID `NewRule`.

```
// Retrieve configuration.
configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

// Add a new rule.
Calendar c = Calendar.getInstance(TimeZone.getTimeZone("GMT"));
c.add(Calendar.YEAR, 10);
c.set(Calendar.HOUR_OF_DAY, 0);
c.set(Calendar.MINUTE, 0);
c.set(Calendar.SECOND, 0);
c.set(Calendar.MILLISECOND, 0);
configuration.getRules().add(
        new BucketLifecycleConfiguration.Rule()
            .withId("NewRule")
            .withPrefix("YearlyDocuments/")
            .withExpirationDate(c.getTime())
            .withStatus(BucketLifecycleConfiguration.
                ENABLED.toString())
           );
// Save configuration.
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);
```

Example Program Listing

The following Java code example provides a complete code listing that adds, updates, and deletes a lifecycle configuration to a bucket. You will need to update the code and provide your bucket name to which the code can add the example lifecycle configuration.

For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.TimeZone;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration.Transition;
import com.amazonaws.services.s3.model.StorageClass;

public class S3LifecycleExample {
    public static String bucketName = "*** Provide bucket name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new PropertiesCredentials(
                S3LifecycleExample.class.getResourceAsStream(
                        "AwsCredentials.properties")));
        try {

            BucketLifecycleConfiguration.Rule rule1 =
            new BucketLifecycleConfiguration.Rule()
            .withId("Archive immediately rule")
            .withPrefix("glacierobjects/")
            .withTransition(new Transition()
                                .withDays(0)
                                .withStorageClass(StorageClass.Glacier))
            .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

            BucketLifecycleConfiguration.Rule rule2 =
                new BucketLifecycleConfiguration.Rule()
                .withId("Archive and then delete rule")
                .withPrefix("projectdocs/")
                .withTransition(new Transition()
                                    .withDays(365)
                                    .withStorageClass(StorageClass.Glacier))
                .withExpirationInDays(3650)
                .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

            List<BucketLifecycleConfiguration.Rule> rules =
            new ArrayList<BucketLifecycleConfiguration.Rule>();
            rules.add(rule1);
            rules.add(rule2);

            BucketLifecycleConfiguration configuration =
            new BucketLifecycleConfiguration()
            .withRules(rules);
            // Save configuration.
           s3Client.setBucketLifecycleConfiguration(bucketName, configuration);


            // Retrieve configuration.
           configuration = s3Client.getBucketLifecycleConfiguration(bucketName);


            // Add a new rule.
            Calendar c = Calendar.getInstance(TimeZone.getTimeZone("GMT"));
            c.add(Calendar.YEAR, 10);
            c.set(Calendar.HOUR_OF_DAY, 0);
            c.set(Calendar.MINUTE, 0);
            c.set(Calendar.SECOND, 0);
            c.set(Calendar.MILLISECOND, 0);
            configuration.getRules().add(
                    new BucketLifecycleConfiguration.Rule()
                        .withId("NewRule")
                        .withPrefix("YearlyDocuments/")
                        .withExpirationDate(c.getTime())
                        .withStatus(BucketLifecycleConfiguration.
                            ENABLED.toString())
                       );
            // Save configuration.
           s3Client.setBucketLifecycleConfiguration(bucketName, configuration);


            // Retrieve configuration.
           configuration = s3Client.getBucketLifecycleConfiguration(bucketName);


            // Verify there are now three rules.
           configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

            System.out.format("Expected # of rules = 3; found: %s\n",
                configuration.getRules().size());

            // Delete configuration.
            s3Client.deleteBucketLifecycleConfiguration(bucketName);

            // Retrieve nonexistent configuration.
           configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

            String s = (configuration == null) ? "No configuration found." :
"Configuration found.";
            System.out.println(s);


        } 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());
        }
    }
}
```

Manage Object Lifecycle Using the AWS SDK for .NET

You can use the AWS SDK for .NET to manage lifecycle configuration on a bucket. For more information about managing lifecycle configuration, see Object Lifecycle Management (p. 106).

Adding a Lifecycle Configuration to a Bucket

1. Create an instance of the `LifeCycleConfiguration` class and specify list of `LifecycleRules`.
2. Create a `PutLifeCycleConfigurationRequest` object.

   You will need to provide a bucket name and the `LifeCycleConfiguration` instance you created in the preceding step.
3. Execute the `client.PutLifecycleConfiguration`

The following C# code snippet demonstrates the preceding tasks. The lifecycle configuration defines one rule ("Archive and delete rule") specifying two actions for Amazon S3 to perform on all objects with key prefix "projectdocs/".

```
string bucketName = "examplebucket";

// Add a sample configuration
using (client = new AmazonS3Client()){
  var lifeCycleConfiguration = new LifecycleConfiguration()
  {
    Rules = new List<LifecycleRule>
    {
        new LifecycleRule
        {
             Id = "Archive and delete rule",
             Prefix = "projectdocs/",
             Status = LifecycleRuleStatus.Enabled,
              Transition = new LifecycleTransition()
              {
                   Days = 365,
                   StorageClass = S3StorageClass.Glacier
              },
              Expiration = new LifecycleRuleExpiration()
              {
                   Days = 3650
              }
        }
    }
  };

  PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest

  {
     BucketName = bucketName,
     Configuration = configuration
  };

  var response = client.PutLifecycleConfiguration(request);
}
```

To update an existing lifecycle configuration

- When you add a lifecycle configuraiton to a bucket, any existing lifecycle configuration is replaced. To update existing lifecycle configuration, you must first retrieve the existing lifecycle configuration, make changes and then add the revised lifecycle configuration to the bucket as shown in the following C# code snippet.

This snippet gets an existing configuration and adds a new rule with the ID `NewRule`.

```
// Retrieve lifecycle configuration.
GetLifecycleConfigurationRequest request = new GetLifecycleConfigurationRequest
{
    BucketName = bucketName
};
var response = client.GetLifecycleConfiguration(request);
var configuration = response.Configuration;

// Add new rule.
configuration.Rules.Add(new LifecycleRule
{
    Id = "NewRule",
    Prefix = "YearlyDocuments/",
    Expiration = new LifecycleRuleExpiration { Date = DateTime.Now.AddYears(10)
 }
});

PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest
{
   BucketName = bucketName,
   Configuration = configuration
};

var response = client.PutLifecycleConfiguration(request);
```

Example Program Listing

The following C# code example provides complete code listing that adds, updates, and deletes a lifecycle configuration to a bucket. You will need to update the code and provide your bucket name to which the code can add the example lifecycle configuration.

For instructions on how to create and test a working sample, see Using the AWS SDK for .NET Testing the .NET Code Examples.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using Amazon.S3;
using Amazon.S3.Model;

namespace aws.amazon.com.s3.documentation
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                AmazonS3Config s3Config = new AmazonS3Config();
                s3Config.ServiceURL = "s3.aws-master.amazon.com";
                using (client = new AmazonS3Client(s3Config))
                {
                    var lifeCycleConfiguration = new LifecycleConfiguration()
                    {
                        Rules = new List<LifecycleRule>
                        {
                            new LifecycleRule
                            {
                                 Id = "Archive immediately rule",
                                 Prefix = "glacierobjects/",
                                 Status = LifecycleRuleStatus.Enabled,
                                  Transition = new LifecycleTransition()
                                  {
                                       Days = 0,
                                       StorageClass = S3StorageClass.Glacier
                                  }
                            },
                            new LifecycleRule
                            {
                                 Id = "Archive and then delete rule",
                                 Prefix = "projectdocs/",
                                 Status = LifecycleRuleStatus.Enabled,
                                  Transition = new LifecycleTransition()
                                  {
                                       Days = 365,
                                       StorageClass = S3StorageClass.Glacier
                                  },
                                  Expiration = new LifecycleRuleExpiration()
                                  {
                                         Days = 3650
                                    }
                              }
                          }
                    };

                    // Add the configuration to the bucket
                    PutLifeCycleConfiguration(lifeCycleConfiguration);

                    // Retrieve an existing configuration
                    lifeCycleConfiguration = GetLifeCycleConfiguration();

                    // Add a new rule.
                    lifeCycleConfiguration.Rules.Add(new LifecycleRule
                    {
                        Id = "NewRule",
                        Prefix = "YearlyDocuments/",
                       Expiration = new LifecycleRuleExpiration { Date = Date
Time.Now.AddYears(10) }
                    });

                    // Add the configuration to the bucket
                    PutLifeCycleConfiguration(lifeCycleConfiguration);

                    // Verify that there are now three rules
                    lifeCycleConfiguration = GetLifeCycleConfiguration();
                    Console.WriteLine("Expected # of rulest=3; found:{0}", li
feCycleConfiguration.Rules.Count);

                    // Delete the configuration
                    DeleteLifecycleConfiguration();

                    // Retrieve a nonexistent configuration
                    lifeCycleConfiguration = GetLifeCycleConfiguration();
                    Debug.Assert(lifeCycleConfiguration == null);
                }

                Console.WriteLine("Example complete. To continue, click
Enter...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " +
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }

       static void PutLifeCycleConfiguration(LifecycleConfiguration configura
tion)
        {

            PutLifecycleConfigurationRequest request = new PutLifecycleConfig
urationRequest
```

```
            {
                BucketName = bucketName,
                Configuration = configuration
            };

            var response = client.PutLifecycleConfiguration(request);
        }

        static LifecycleConfiguration GetLifeCycleConfiguration()
        {
            GetLifecycleConfigurationRequest request = new GetLifecycleConfig
urationRequest
            {
                BucketName = bucketName

            };
            var response = client.GetLifecycleConfiguration(request);
            var configuration = response.Configuration;
            return configuration;
        }

        static void DeleteLifecycleConfiguration()
        {
          DeleteLifecycleConfigurationRequest request = new DeleteLifecycleCon
figurationRequest
            {
                BucketName = bucketName
            };
            client.DeleteLifecycleConfiguration(request);
        }
    }
}
```

Manage Object Lifecycle Using the REST API

You can use the AWS Management Console to set 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

Enabling Cross-Origin Resource Sharing

Topics

- Enabling Cross-Origin Resource Sharing (CORS) Using the REST API (p. 138)

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 in Amazon S3, you can build rich client-side web applications with Amazon S3 and selectively allow cross-origin access to your Amazon S3 resources.

Cross-Origin Resource Sharing: Examples

Cross-origin resource sharing enables several use cases. For example, suppose you are hosting a website in an Amazon S3 bucket named `website` as described in Hosting a Static Website on Amazon S3 (p. 377). Your users load the website endpoint `http://website.s3-website-us-east-1.amazonaws.com`. Now you want to use JavaScript on the web pages stored 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's 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`.

As another example, suppose you want to host a web font from your S3 bucket. Again, browsers require a CORS check (also referred as a preflight check) for loading web fonts, so you would configure the bucket that is hosting the web font to allow any origin to make these requests.

How Do I Enable CORS On My Bucket?

To configure your bucket to allow cross-origin requests, you create a CORS configuration, an XML document with rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) 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.

For example, the following `cors` configuration on a bucket has two rules, which are specified as `CORSRule` elements:

- The first rule allows cross-origin PUT, POST, and DELETE requests from the `https://www.example.com` origin. The rule also allows all headers in a preflight OPTIONS request through the `Access-Control-Request-Headers` header. In response to any preflight OPTIONS request, Amazon S3 will return any requested headers.
- The second rule allows cross-origin GET requests from all origins. The '*' wildcard character refers to all origins.

```
<CORSConfiguration>
 <CORSRule>
   <AllowedOrigin>http://www.example.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 following CORS configuration allows cross-origin PUT and POST 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 will cache 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 is to 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 will be 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 at most 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 JavasSript `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 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.

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 step-by-step instructions, see Editing Bucket Permissions in the *Amazon S3 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 Enabling Cross-Origin Resource Sharing (p. 124).

This section provides sample code snippets for following tasks, followed by a complete example program demonstrating all tasks.

- Creating an instance of the Amazon S3 client class
- Creating and adding a CORS configuration to a bucket
- Updating an existing CORS configuration

Cross-Origin Resource Sharing Methods

AmazonS3Client()	Constructs an `AmazonS3Client` object with the credentials defined in the AwsCredentials.properties file.
setBucketCrossOriginConfiguration()	Sets the CORS configuration that to be applied to the bucket. If a configuration already exists for the specified bucket, the new configuration will replace the existing one.
getBucketCrossOriginConfiguration()	Retrieves the CORS configuration for the specified bucket. If no configuration has been set for the bucket, the `Configuration` header in the response will be null.
deleteBucketCrossOriginConfiguration()	Deletes the CORS configuration for the specified bucket.

For more information about the AWS SDK for Java API, go to AWS SDK for Java API Reference.

Creating an Instance of the Amazon S3 Client Class

The following snippet creates a new `AmazonS3Client` instance for a class called `CORS_JavaSDK`. This example retrieves the values for `accessKey` and `secretKey` from the AwsCredentials.properties file.

```
AmazonS3Client client;
AWSCredentials credentials = new PropertiesCredentials(
     CORS_JavaSDK.class.getResourceAsStream("AwsCredentials.properties"));

client = new AmazonS3Client(credentials);
```

Creating and Adding a CORS Configuration to a Bucket

To add a CORS configuration to a bucket:

1. Create a `CORSRule` object that describes the rule.
2. Create a `BucketCrossOriginConfiguration` object, and then add the rule to the configuration object.
3. Add the CORS configuration to the bucket by calling the `client.setBucketCrossOriginConfiguration` method.

The following snippet creates two rules, `CORSRule1` and `CORSRule2`, and then adds each rule to the `rules` array. By using the `rules` array, it then adds the rules to the bucket `bucketName`.

```
// Add a sample configuration
BucketCrossOriginConfiguration configuration = new BucketCrossOriginConfigura
tion();

List<CORSRule> rules = new ArrayList<CORSRule>();

CORSRule rule1 = new CORSRule()
    .withId("CORSRule1")
    .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
            CORSRule.AllowedMethods.PUT, CORSRule.AllowedMethods.POST, COR
SRule.AllowedMethods.DELETE}))
    .withAllowedOrigins(Arrays.asList(new String[] {"http://*.example.com"}));

CORSRule rule2 = new CORSRule()
```

```
.withId("CORSRule2")
.withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
        CORSRule.AllowedMethods.GET}))
.withAllowedOrigins(Arrays.asList(new String[] {"*"}))
.withMaxAgeSeconds(3000)
.withExposedHeaders(Arrays.asList(new String[] {"x-amz-server-side-encryp
tion"}));

configuration.setRules(Arrays.asList(new CORSRule[] {rule1, rule2}));

// Save the configuration
client.setBucketCrossOriginConfiguration(bucketName, configuration);
```

Updating an Existing CORS Configuration

To update an existing CORS configuration

1. Get a CORS configuration by calling the `client.getBucketCrossOriginConfiguration` method.
2. Update the configuration information by adding or deleting rules to the list of rules.
3. Add the configuration to a bucket by calling the `client.getBucketCrossOriginConfiguration` method.

The following snippet gets an existing configuration and then adds a new rule with the ID `NewRule`.

```
// Get configuration.
BucketCrossOriginConfiguration configuration = client.getBucketCrossOriginCon
figuration(bucketName);

// Add new rule.
CORSRule rule3 = new CORSRule()
.withId("CORSRule3")
.withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
        CORSRule.AllowedMethods.HEAD}))
.withAllowedOrigins(Arrays.asList(new String[] {"http://www.example.com"}));

List<CORSRule> rules = configuration.getRules();
rules.add(rule3);
configuration.setRules(rules);

// Save configuration.
client.setBucketCrossOriginConfiguration(bucketName, configuration);
```

Example Program Listing

The following Java program incorporates the preceding tasks.

For information about creating and testing a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketCrossOriginConfiguration;
import com.amazonaws.services.s3.model.CORSRule;

public class CORS_JavaSDK {

    /**
     * @param args
     * @throws IOException
     */
    public static AmazonS3Client client;
    public static String bucketName = "***provide bucket name***";

    public static void main(String[] args) throws IOException {

        AWSCredentials credentials = new PropertiesCredentials(
                CORS_JavaSDK.class
                        .getResourceAsStream("AwsCredentials.properties"));

        client = new AmazonS3Client(credentials);

        // Create a new configuration request and add two rules
       BucketCrossOriginConfiguration configuration = new BucketCrossOriginCon
figuration();

        List<CORSRule> rules = new ArrayList<CORSRule>();

        CORSRule rule1 = new CORSRule()
            .withId("CORSRule1")
            .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {

                    CORSRule.AllowedMethods.PUT, CORSRule.AllowedMethods.POST,
 CORSRule.AllowedMethods.DELETE}))
            .withAllowedOrigins(Arrays.asList(new String[] {"http://*.ex
ample.com"}));

        CORSRule rule2 = new CORSRule()
        .withId("CORSRule2")
        .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
                CORSRule.AllowedMethods.GET}))
        .withAllowedOrigins(Arrays.asList(new String[] {"*"}))
        .withMaxAgeSeconds(3000)
        .withExposedHeaders(Arrays.asList(new String[] {"x-amz-server-side-en
cryption"}));
```

```
        configuration.setRules(Arrays.asList(new CORSRule[] {rule1, rule2}));

         // Add the configuration to the bucket.
        client.setBucketCrossOriginConfiguration(bucketName, configuration);

        // Retrieve an existing configuration.
        configuration = client.getBucketCrossOriginConfiguration(bucketName);
        printCORSConfiguration(configuration);

        // Add a new rule.
        CORSRule rule3 = new CORSRule()
        .withId("CORSRule3")
        .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
                CORSRule.AllowedMethods.HEAD}))
        .withAllowedOrigins(Arrays.asList(new String[] {"http://www.ex
ample.com"}));

        rules = configuration.getRules();
        rules.add(rule3);
        configuration.setRules(rules);
        client.setBucketCrossOriginConfiguration(bucketName, configuration);
        System.out.format("Added another rule: %s\n", rule3.getId());

        // Verify that the new rule was added.
        configuration = client.getBucketCrossOriginConfiguration(bucketName);
        System.out.format("Expected # of rules = 3, found %s", configura
tion.getRules().size());

        // Delete the configuration.
        client.deleteBucketCrossOriginConfiguration(bucketName);

        // Try to retrieve configuration.
        configuration = client.getBucketCrossOriginConfiguration(bucketName);
        System.out.println("\nRemoved CORS configuration.");
        printCORSConfiguration(configuration);
    }

   static void printCORSConfiguration(BucketCrossOriginConfiguration configur
ation)
    {

        if (configuration == null)
        {
            System.out.println("\nConfiguration is null.");
            return;
        }

        System.out.format("\nConfiguration has %s rules:\n", configura
tion.getRules().size());
        for (CORSRule rule : configuration.getRules())
        {
            System.out.format("Rule ID: %s\n", rule.getId());
           System.out.format("MaxAgeSeconds: %s\n", rule.getMaxAgeSeconds());

            System.out.format("AllowedMethod: %s\n", rule.getAllowedMeth
ods().toArray());
          System.out.format("AllowedOrigins: %s\n", rule.getAllowedOrigins());
```

```
            System.out.format("AllowedHeaders: %s\n", rule.getAllowedHeaders());

              System.out.format("ExposeHeader: %s\n", rule.getExposedHeaders());

          }
     }
}
```

Enabling Cross-Origin Resource Sharing (CORS) Using the AWS SDK for .NET

You can use the AWS SDK for .NET to manage cross-origin resource sharing (CORS) for a bucket. For more information about CORS, see Enabling Cross-Origin Resource Sharing (p. 124).

This section provides sample code snippets for the tasks in the following table, followed by a complete sample program listing.

Managing Cross-Origin Resource Sharing

1	Create an instance of the `AmazonS3Client` class.
2	Create a new CORS configuration.
3	Retrieve and modify an existing CORS configuration.
4	Add the configuration to the bucket.

Cross-Origin Resource Sharing Methods

AmazonS3Client()	Constructs `AmazonS3Client` with the credentials defined in the App.config file.
PutCORSConfiguration()	Sets the CORS configuration that should be applied to the bucket. If a configuration already exists for the specified bucket, the new configuration will replace the existing one.
GetCORSConfiguration()	Retrieves the CORS configuration for the specified bucket. If no configuration has been set for the bucket, the `Configuration` header in the response will be null.
DeleteCORSConfiguration()	Deletes the CORS configuration for the specified bucket.

For more information about the AWS SDK for .NET API, go to AWS SDK for .NET Reference.

Creating an Instance of the AmazonS3 Class

The following snippet uses the `CreateAmazonS3Client` method to create an instance of the `AmazonS3Client` class. This example retrieves the values for `accessKey` and `secretKey` from app.config.

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;

string accessKey = appConfig["AWSAccessKey"];
string secretKey = appConfig["AWSSecretKey"];
```

```
try
{
using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
    accessKey, secretKey ))
```

Adding a CORS Configuration to a Bucket

To add a CORS configuration to a bucket:

1. Create a `CORSConfiguration` object describing the rule.
2. Create a `PutCORSConfigurationRequest` object that provides the bucket name and the CORS configuration.
3. Add the CORS configuration to the bucket by calling `client.PutCORSConfiguration`.

The following snippet creates two rules, `CORSRule1` and `CORSRule2`, and then adds each rule to the `rules` array. By using the `rules` array, it then adds the rules to the bucket `bucketName`.

```
// Add a sample configuration
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"}
        }
    }
};

// Save the configuration
PutCORSConfiguration(configuration);

static void PutCORSConfiguration(CORSConfiguration configuration)
{

    PutCORSConfigurationRequest request = new PutCORSConfigurationRequest
    {
        BucketName = bucketName,
        Configuration = configuration
    };

    var response = client.PutCORSConfiguration(request);
}
```

Updating an Existing CORS Configuration

To update an existing CORS configuration

1. Get a CORS configuration by calling the `client.GetCORSConfiguration` method.
2. Update the configuration information by adding or deleting rules.
3. Add the configuration to a bucket by calling the `client.PutCORSConfiguration` method.

The following snippet gets an existing configuration and then adds a new rule with the ID `NewRule`.

```
// Get configuration.
configuration = GetCORSConfiguration();
// Add new rule.
configuration.Rules.Add(new CORSRule
{
    Id = "NewRule",
    AllowedMethods = new List<string> { "HEAD" },
    AllowedOrigins = new List<string> { "http://www.example.com" }
});

// Save configuration.
PutCORSConfiguration(configuration);
```

Example Program Listing

The following C# program incorporates the preceding tasks.

For information about creating and testing a working sample, see Testing the .NET Code Examples (p. 436).

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;
using System.Diagnostics;
using System.Collections.Generic;

namespace CORS_DotNetSDK
{
    class S3Sample
    {
        static string bucketName = "examplebucket";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
           /* Create an instance of the AmazonS3 class using CreateAmazonS3Cli
ent().
            ** CreateAmazonS3Client takes the access key and secret key as
parameters.
            ** This example uses the values in app.Config for security. */

            NameValueCollection appConfig =
              ConfigurationManager.AppSettings;

            string accessKey = appConfig["AWSAccessKey"];
            string secretKey = appConfig["AWSSecretKey"];
            try
            {
                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                  accessKey, secretKey))
                {
                    // 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://*.ex
ample.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 
                    PutCORSConfiguration(configuration);

                    // Retrieve an existing configuration 
                    configuration = GetCORSConfiguration();

                    // Add a new rule.
                    configuration.Rules.Add(new CORSRule
                    {
                        Id = "CORSRule3",
                        AllowedMethods = new List<string> { "HEAD" },
                        AllowedOrigins = new List<string> { "http://www.ex
ample.com" }
                    });

                    // Add the configuration to the bucket 
                    PutCORSConfiguration(configuration);

                    // Verify that there are now three rules
                    configuration = GetCORSConfiguration();
                    Console.WriteLine("Expected # of rulest=3; found:{0}", 
configuration.Rules.Count);

                    // Delete the configuration
                    DeleteCORSConfiguration();

                    // Retrieve a nonexistent configuration
                    configuration = GetCORSConfiguration();
                    Debug.Assert(configuration == null);
                }

                Console.WriteLine("Example complete. To continue, click 
Enter...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " + 
amazonS3Exception.ToString());
                Console.ReadKey();
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
                Console.ReadKey();
            }
        }
        static void PutCORSConfiguration(CORSConfiguration configuration)
        {

          PutCORSConfigurationRequest request = new PutCORSConfigurationRequest

            {
                BucketName = bucketName,
                Configuration = configuration
            };

            var response = client.PutCORSConfiguration(request);
        }

        static CORSConfiguration GetCORSConfiguration()
        {
          GetCORSConfigurationRequest request = new GetCORSConfigurationRequest

            {
                BucketName = bucketName

            };
            var response = client.GetCORSConfiguration(request);
            var configuration = response.Configuration;
            PrintCORSRules(configuration);
            return configuration;
        }

        static void DeleteCORSConfiguration()
        {
            DeleteCORSConfigurationRequest request = new DeleteCORSConfigura
tionRequest
            {
                BucketName = bucketName
            };
            client.DeleteCORSConfiguration(request);
        }

        static void PrintCORSRules(CORSConfiguration configuration)
        {
            Console.WriteLine();

            if (configuration == null)
            {
                Console.WriteLine("\nConfiguration is null");
                return;
            }

            Console.WriteLine("Configuration has {0} rules:", configura
tion.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

You can use the AWS Management Console to set CORS 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 actions related to the CORS configuration:

- PUT Bucket cors
- GET Bucket cors
- DELETE Bucket cors
- OPTIONS object

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. 167).
- **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. 208).

You can use the REST API (see Making Requests using the REST API (p. 45)) 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.

Multipart Upload Overview

Topics

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. 138)).

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 of all 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. 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.

Multipart Upload Completion (or Abort)

When you complete a multipart upload, Amazon S3 creates an object by concatenating 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 1000 parts. If there are more than 1000 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.

> **Note**
> 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.

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 1000 multipart uploads in progress, you need to send additional requests to retrieve the remaining multipart uploads.

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 you have uploaded, and you are no longer billed for them. For more information on pricing, go to the Amazon S3 Pricing page.

Quick Facts

The following table provides multipart upload (see Multipart Upload Overview (p. 139)) core specifications.

Item	Specification
Maximum object size	5 TB
Maximum number of parts per upload	10,000
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 *s3:PutObject* action on an object to initiate multipart upload. The bucket owner can allow other principals to perform the *s3:PutObject* action.
Upload Part	You must be allowed to perform the *s3:PutObject* action on an object to upload a part. Only the initiator of a multipart upload can upload parts. The bucket owner must allow the initiator to perform the *s3:PutObject* action on an object in order for the initiator to upload a part for that object.
Upload Part (Copy)	You must be allowed to perform the *s3:PutObject* action on an object to upload a part. Because your are uploading a part from an existing object, you must be allowed *s3:GetObject* on the source object. Only the initiator of a multipart upload can upload parts. The bucket owner must allow the initiator to perform the *s3:PutObject* action on an object in order for the initiator to upload a part for that object.
Complete Multipart Upload	You must be allowed to perform the *s3:PutObject* action on an object to complete a multipart upload. Only the initiator of a multipart upload can complete that multipart upload. The bucket owner must allow the initiator to perform the *s3:PutObject* action on an object in order for the initiator to complete a multipart upload for that object.

Action	Required Permissions
Abort Multipart Upload	You must be allowed to perform the *`s3:AbortMultipartUpload`* action to abort a multipart upload. 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. In addition to these defaults, the bucket owner can allow other principals to perform the *`s3:AbortMultipartUpload`* action on an object. The bucket owner can deny any principal the ability to perform the *`s3:AbortMultipartUpload`* action.
List Parts	You must be allowed to perform the *`s3:ListMultipartUploadParts`* action to list parts in a multipart upload. 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. In addition to these defaults, the bucket owner can allow other principals to perform the *`s3:ListMultipartUploadParts`* action on an object. The bucket owner can also deny any principal the ability to perform the *`s3:ListMultipartUploadParts`* action.
List Multipart Uploads	You must be allowed to perform the *`s3:ListBucketMultipartUploads`* action on a bucket to list multipart uploads in progress to that bucket. In addition to the default, the bucket owner can allow other principals to perform the *`s3:ListBucketMultipartUploads`* action on the bucket.

For additional information on actions and their use in policies, see Action (p. 451). For information on the relationship between policy actions and ACL permissions, see Relationship Between Actions and Permissions (p. 337). For information on IAM Users, go to Working with Users and Groups.

Getting Objects

Topics

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.

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 SDK for Java, .NET and PHP also provide necessary objects you can use to specify values for these response headers in your GET request.

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

Get an Object Using the AWS SDK for Java

When you download an object, you get all of 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.

Downloading Objects

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute one of the `AmazonS3Client.getObject()` method. You need to provide the request information, such as bucket name, and key name. You provide this information by creating an instance of the `GetObjectRequest` class.
3	Execute one of the `getObjectContent()` methods on the object returned to get a stream on the object data and process the response.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);
S3Object object = s3Client.getObject(
                  new GetObjectRequest(bucketName, key));
InputStream objectData = object.getObjectContent();
// Process the objectData stream.
objectData.close();
```

The `GetObjectRequest` object provides several options, including conditional downloading of objects based on modification times, ETags, and selectively downloading a range of an object. The following Java code sample demonstrates how you can specify a range of data bytes to retrieve from an object.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                   myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

GetObjectRequest rangeObjectRequest = new GetObjectRequest(
  bucketName, key);
rangeObjectRequest.setRange(0, 10); // retrieve 1st 10 bytes.
S3Object objectPortion = s3Client.getObject(rangeObjectRequest);
```

```
InputStream objectData = objectPortion.getObjectContent();
// Process the objectData stream.
objectData.close();
```

When retrieving an object, you can optionally override the response header values (see Getting Objects (p. 142)) by using the `ResponseHeaderOverrides` object and setting the corresponding request property, as shown in the following Java code sample.

```
GetObjectRequest request = new GetObjectRequest(bucketName, key);

ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
responseHeaders.setCacheControl("No-cache");
responseHeaders.setContentDisposition("attachment; filename=testing.txt");

// Add the ResponseHeaderOverides to the request.
request.setResponseHeaders(responseHeaders);
```

Example

The following Java code example retrieves an object from a specified Amazon S3 bucket. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";
 private static String key        = "*** Provide Key ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            System.out.println("Downloading an object");
            S3Object s3object = s3Client.getObject(new GetObjectRequest(
              bucketName, key));
            System.out.println("Content-Type: "  +
              s3object.getObjectMetadata().getContentType());
            displayTextInputStream(s3object.getObjectContent());

           // Get a range of bytes from an object.

            GetObjectRequest rangeObjectRequest = new GetObjectRequest(
              bucketName, key);
            rangeObjectRequest.setRange(0, 10);
            S3Object objectPortion = s3Client.getObject(rangeObjectRequest);

            System.out.println("Printing bytes retrieved.");
            displayTextInputStream(objectPortion.getObjectContent());

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
              " means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means"+

              " the client encountered " +
                    "an internal error while trying to " +
                    "communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

    private static void displayTextInputStream(InputStream input)
    throws IOException {
     // Read one text line at a time and display.
        BufferedReader reader = new BufferedReader(new
          InputStreamReader(input));
        while (true) {
            String line = reader.readLine();
            if (line == null) break;

            System.out.println("    " + line);
        }
        System.out.println();
    }
}
```

Get an Object Using the AWS SDK for .NET

The following tasks guide you through using the .NET classes to retrieve an object or a portion of the object.

Downloading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute one of the `AmazonS3.GetObject` methods. You need to provide information such as bucket name, file path, or a stream. You provide this information by creating an instance of the `GetObjectRequest` class.
3	Execute one of the `GetObjectResponse.WriteResponseStreamToFile` methods to save the stream to a file.

The following C# code sample demonstrates the preceding tasks.

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

GetObjectRequest request = new GetObjectRequest()
            .WithBucketName(bucketName).WithKey(keyName);
using (GetObjectResponse response = client.GetObject(request))
{
   string title = response.Metadata["x-amz-meta-title"];
   Console.WriteLine("The object's title is {0}", title);
   string dest = Path.Combine(Environment.GetFolderPath(
         Environment.SpecialFolder.Desktop), keyName);
   if (!File.Exists(dest))
   {
       response.WriteResponseStreamToFile(dest);
   }
}
```

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# code sample.

```
GetObjectRequest request = new GetObjectRequest()
        .WithBucketName(bucketName)
        .WithKey(keyName)
        .WithByteRange(0, 10);
```

When retrieving an object, you can optionally override the response header values (see Getting Objects (p. 142)) by using the `ResponseHeaderOverrides` object and setting the corresponding request property, as shown in the following C# code sample.

```
GetObjectRequest request = new GetObjectRequest()
     .WithBucketName(bucketName).WithKey(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. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.IO;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.retrieveobject
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static string keyName    = "*** Provide object key ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];
                try
                {
                    Console.WriteLine("Retrieving (getting) an object");
                    string data = ReadingAnObject(
                                         accessKeyID, secretAccessKeyID);
                }
                catch (AmazonS3Exception s3Exception)
                {
                    Console.WriteLine(s3Exception.Message,
                                      s3Exception.InnerException);
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static string ReadingAnObject(
                      string accessKeyID, string secretAccessKeyID)
        {
            string responseBody = "";
            using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                                         accessKeyID, secretAccessKeyID))
            {
                GetObjectRequest request = new GetObjectRequest()
                    .WithBucketName(bucketName).WithKey(keyName);
                using (GetObjectResponse response = client.GetObject(request))

                {
                    string title = response.Metadata["x-amz-meta-title"];
                    Console.WriteLine("The object's title is {0}", title);

                    using (Stream responseStream = response.ResponseStream)
                    {
                        using (StreamReader reader =
                            new StreamReader(responseStream))
                        {
                            responseBody = reader.ReadToEnd();
                        }
                    }
                }
            }
            return responseBody;
        }

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }
            if (string.IsNullOrEmpty(keyName))
            {
                Console.WriteLine("The variable keyName is not set.");
                return false;
            }

            return true;
        }
    }
}
```

Get an Object Using the AWS SDK for PHP

The following tasks guide you through using the PHP classes to retrieve an object. You can retrieve an entire object or only a specific byte range from the object.

Downloading Files

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the `AmazonS3::get_object()` method. You need to provide a bucket name, and a key name as parameters. Instead of retrieving the entire object you can optionally retrieve a specific byte range from the object data. You provide the range value by specifying the optional array parameter with the `range` key. You can save the object you retrieved from Amazon S3, to a file in your local file system by specifying the optional array parameter with the `fileDownload` key.

The following PHP code sample demonstrates the preceding tasks.

```
// Instantiate the class.
$s3 = new AmazonS3();
$response = $s3->get_object($bucket, $keyname);

// Get a range of bytes.
$response = $s3->get_object(
                     $bucket,
                     $keyname,
                     array('range'=> '0-10'));

// Save object to a file.
$response = $s3->get_object(
                    $bucket,
                    $keyname,
                    array('fileDownload'=> $filepath));
```

When retrieving an object, you can optionally override the response header values (see Getting Objects (p. 142)) by adding the optional `response` parameter to the `get_object` method, as shown in the following PHP code sample.

```
$response = $s3->get_object(
           $bucket,
           $keyname,
           array(
             'response' => array(
               'content-type'        => 'text/plain',
               'content-language'    => 'en-US',
               'content-disposition' => 'attachment; filename=testing.txt',
               'cache-control'       => 'No-cache',
               'expires' => gmdate(DATE_RFC2822, strtotime('1 January 1980'))
               )
         ));
```

Example

The following PHP example retrieves an object and displays object content in the browser. The example illustrates the use of the `get_object()` method.

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';

$bucket = '*** Provide bucket Name ***';
$keyname = '*** Provide object key name ***';
$filepath = '*** local file path to save the object ***';

// Instantiate the class.
$s3 = new AmazonS3();

// 1. Get object.
$response = $s3->get_object(
                      $bucket,
                      $keyname);

// Success?
if($response->isOK())
{
 header('Content-Type: ' . $response->header['content-type']);
 echo $response->body;
}
```

Get an Object Using REST API

You can use AWS SDK to list object keys in 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

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 pre-signed URL, using their own security credentials, to grant time-limited permission to download the objects.

When you create a pre-signed 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 pre-signed URLs are valid only for the specified duration.

Anyone who recieves the pre-signed 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 pre-signed URL.

Note
Anyone with valid security credentials can create a pre-signed URL. However, in order to successfully access an object, the pre-signed URL must be created by someone who has permission to perform the operation that the pre-signed URL is based upon.

You can generate pre-signed URL programmatically using the AWS SDK for Java and .NET.

Generate Pre-signed Object URL using AWS Explorer for Visual Studio

If you are using Visual Studio, you can generate a pre-signed URL for an object without writing any code by using the 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 and Explorers (p. 433).

Generate Pre-signed Object URL using AWS SDK for Java

The following tasks guide you through using the Java classes to generate a pre-signed URL.

Downloading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials. These credentials are used in creating a signature for authentication when you generate a pre-signed URL.
2	Execute the `AmazonS3.generatePresignedUrl` method to generate a pre-signed URL. You provide information including a bucket name, an object key, and an expiration date by creating an instance of the `GeneratePresignedUrlRequest` class.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

java.util.Date expiration = new java.util.Date();
long msec = expiration.getTime();
msec += 1000 * 60 * 60; // 1 hour.
expiration.setTime(msec);

GeneratePresignedUrlRequest generatePresignedUrlRequest =
              new GeneratePresignedUrlRequest(bucketName, objectKey);
generatePresignedUrlRequest.setMethod(HttpMethod.GET); // Default.
generatePresignedUrlRequest.setExpiration(expiration);

URL s = s3client.generatePresignedUrl(generatePresignedUrlRequest);
```

Example

The following Java code example generates a pre-signed URL that you can give to others so that they can retrieve the object. For instructions about how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.io.IOException;
import java.net.URL;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import java.net.*;
import java.io.*;

public class S3Sample {
 private static String bucketName = "*** Provide a bucket name ***";
 private static String objectKey = "*** Provide an object key ***";

 public static void main(String[] args) throws IOException {
  AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
   S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

  try {
   System.out.println("Generating pre-signed URL.");
   java.util.Date expiration = new java.util.Date();
   long milliSeconds = expiration.getTime();
   milliSeconds += 1000 * 60 * 60; // Add 1 hour.
   expiration.setTime(milliSeconds);

   GeneratePresignedUrlRequest generatePresignedUrlRequest =
        new GeneratePresignedUrlRequest(bucketName, objectKey);
   generatePresignedUrlRequest.setMethod(HttpMethod.GET);
   generatePresignedUrlRequest.setExpiration(expiration);

   URL url = s3client.generatePresignedUrl(generatePresignedUrlRequest);

   System.out.println("Pre-Signed URL = " + url.toString());
  } catch (AmazonServiceException exception) {
   System.out.println("Caught an AmazonServiceException, " +
     "which means your request made it " +
     "to Amazon S3, but was rejected with an error response " +
   "for some reason.");
   System.out.println("Error Message: " + exception.getMessage());
   System.out.println("HTTP  Code: "    + exception.getStatusCode());
   System.out.println("AWS Error Code:" + exception.getErrorCode());
   System.out.println("Error Type:    " + exception.getErrorType());
   System.out.println("Request ID:    " + exception.getRequestId());
  } catch (AmazonClientException ace) {
   System.out.println("Caught an AmazonClientException, " +
     "which means the client encountered " +
     "an internal error while trying to communicate" +
     " with S3, " +
   "such as not being able to access the network.");
   System.out.println("Error Message: " + ace.getMessage());
  }
 }
}
```

Generate Pre-signed Object URL using AWS SDK for .NET

The following tasks guide you through using the .NET classes to generate a pre-signed URL.

Downloading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials. These credentials are used in creating a signature for authentication when you generate a pre-signed URL.
2	Execute the `AmazonS3.GetPreSignedURL` method to generate a pre-signed URL. You provide information including a bucket name, an object key, and an expiration date by creating an instance of the `GetPreSignedUrlRequest` class.

The following C# code sample demonstrates the preceding tasks.

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
request.WithBucketName(bucketName);
request.WithKey(objectKey);
request.Verb = HttpVerb.GET; // Default.
request.WithExpires(DateTime.Now.AddMinutes(5));

string url = s3Client.GetPreSignedURL(request);
```

Example

The following C# code example generates a pre-signed URL for a specific object. For instructions about how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.presignedurl
{
    class S3Sample
    {
        static string bucketName = "*** Provide a bucket name ***";
        static string objectKey = "*** Provide an object name ***";
        static AmazonS3 s3Client;

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];
               using (s3Client = Amazon.AWSClientFactory.CreateAmazonS3Client(

                    accessKeyID, secretAccessKeyID))
                {
                        GeneratePreSignedURL();
                }


            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void GeneratePreSignedURL()
        {
            GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
            request.WithBucketName(bucketName);
            request.WithKey(objectKey);
            request.WithExpires(DateTime.Now.AddMinutes(5));

            try
            {
                string url = s3Client.GetPreSignedURL(request);
            }
            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);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

        }
    }
}
```

Uploading Objects

Topics

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. 156).
- **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. For more information, see Uploading Objects Using Multipart Upload API (p. 167).

We encourage Amazon S3 customers to use Multipart Upload for objects greater than 100 MB.

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

Uploading Objects in a Single Operation

Topics

- Upload an Object Using the AWS SDK for .NET (p. 158)
- Upload an Object Using the AWS SDK for PHP (p. 162)
- Upload an Object Using the AWS SDK for Ruby (p. 165)
- Upload an Object Using the REST API (p. 166)

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

The following tasks guide you through using the Java classes to upload a file. The API provides several variations, called *overloads*, of the `putObject` method to easily upload your data.

Uploading Objects

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute one of the `AmazonS3Client.putObject` overloads depending on whether you are uploading data from a file, or a stream.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                      myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.putObject(new PutObjectRequest(bucketName, keyName, file));
```

Example

The following Java code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.io.File;
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class S3Sample {
 private static String bucketName     = "*** Provide bucket name ***";
 private static String keyName        = "*** Provide key ***";
 private static String uploadFileName = "*** Provide file name ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            System.out.println("Uploading a new object to S3 from a file\n");
            File file = new File(uploadFileName);
            s3client.putObject(new PutObjectRequest(
                                 bucketName, keyName, file));

         } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which " +
              "means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which " +
              "means the client encountered " +
                    "an internal error while trying to " +
                    "communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Upload an Object Using the AWS SDK for .NET

The tasks in the following process guide you through using the .NET classes to upload an object. The API provides several variations, overloads, of the `PutObject` method to easily upload your data.

Uploading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute one of the `AmazonS3.PutObject`. You need to provide information such as a bucket name, file path, or a stream. You provide this information by creating an instance of the `PutObjectRequest` class.

The following C# code sample demonstrates the preceding tasks.

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

PutObjectRequest request = new PutObjectRequest();
request.WithFilePath(filePath)
       .WithBucketName(bucketName)
       .WithKey(keyName);
S3Response responseWithMetadata =
                     client.PutObject(titledRequest);
```

Note
When uploading large objects using the .NET API, timeout might occur even while data is being written to the request stream. You can set explicit timeout using the `PutObjectRequest` object.

Example

The following C# code example uploads an object. The object data is provided as a text string in the code. The example illustrates the use of the `AmazonS3.PutObject` to upload an object. The example uploads the object twice. In the first object upload, the `PutObjectRequest` specifies only the bucket name, key name, and sample object data. In the second object upload the `PutObjectRequest` provides additional information including the optional object metadata and a content type header. Each successive call to `AmazonS3.PutObject` replaces the previous upload. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.Net;

using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;

namespace s3.amazon.com.docsamples.createobject
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static string keyName    = "*** Provide key name ***";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {

                    Console.WriteLine("Uploading an object");
                    WritingAnObject();
                 }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

         static void WritingAnObject()
        {
            try
            {
                // 1. Simple object put.
                PutObjectRequest request = new PutObjectRequest();
                request.WithContentBody("Object data for simple put.")
```

```
                    .WithBucketName(bucketName)
                    .WithKey(keyName);

                S3Response response = client.PutObject(request);
                response.Dispose();

               // 2. Put a more complex object with metadata and http headers.

                PutObjectRequest request2 = new PutObjectRequest();
                request2.WithMetaData("title", "the title")
                    .WithContentBody("Object data for complex put.")
                    //.WithFilePath(filePath)
                    .WithBucketName(bucketName)
                    .WithKey(keyName);
                // Add a header to the request.
                request2.AddHeaders(AmazonS3Util.CreateHeaderEntry
                                            ("ContentType", "text/xml"));

             S3Response responseWithMetadata = client.PutObject(request2);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                    ||
                    amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                {
                    Console.WriteLine("Check the provided AWS Credentials.");
                    Console.WriteLine(
                        "For service sign up go to http://aws.amazon.com/s3");

                }
                else
                {
                    Console.WriteLine(
                        "Error occurred. Message:'{0}' when writing an object"

                        , amazonS3Exception.Message);
                }
            }
        }

         static bool checkRequiredFields()
         {
             NameValueCollection appConfig = ConfigurationManager.AppSettings;


             if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
             {
                 Console.WriteLine(
                     "AWSAccessKey was not set in the App.config file.");
                 return false;
             }
             if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
             {
                 Console.WriteLine(
                     "AWSSecretKey was not set in the App.config file.");
                 return false;
             }
             if (string.IsNullOrEmpty(bucketName))
             {
                 Console.WriteLine("The variable bucketName is not set.");
                 return false;
             }
             if (string.IsNullOrEmpty(keyName))
             {
                 Console.WriteLine("The variable keyName is not set.");
                 return false;
             }

             return true;
         }
    }
}
```

Upload an Object Using the AWS SDK for PHP

The following tasks guide you through PHP classes 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. 167).

Uploading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the `AmazonS3::create_object()` method. You need to provide information such as a bucket name, and key name. If you are uploading a file, you specify the file name by adding the array parameter with the `fileUpload` key. You can also provide the optional object metadata using the array parameter.

The following PHP code sample demonstrates the preceding tasks. The code sample creates an object by uploading a file specified in the `fileUpload` key in the array parameter.

```
// Instantiate the class.
$s3 = new AmazonS3();

$response = $s3->create_object(
    $bucket,
    $keyname2,
    array(
       'fileUpload'  => $filePath,
       'acl'         => AmazonS3::ACL_PUBLIC,
       'contentType' => 'text/plain',
       'storage'     => AmazonS3::STORAGE_REDUCED,
       'headers'     => array( // raw headers
                          'Cache-Control'    => 'max-age',
                          'Content-Encoding' => 'gzip',
                          'Content-Language' => 'en-US',
                          'Expires'       => 'Thu, 01 Dec 1994 16:00:00 GMT',
          ),
         'meta'     => array(
                         'param1' => 'value 1',
                         'param2' => 'value 2'
         ) )
    );
print_r($response);
```

Instead of specifying a file name, you can provide object data inline by specifying the array parameter with the `body` key, as shown in the following PHP code example.

```
// Instantiate the class.
$s3 = new AmazonS3();

$response = $s3->create_object(
                           $bucket,
                           $keyname1,
                           array('body' => 'Sample object data')
);
```

Example

The following PHP example creates two objects in a specified bucket. The example creates an object first by uploading a file and then another object using the text string provided in the code. The example illustrates the use of the `create_object()` method to upload a file.

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';

$bucket   = '*** Provide bucket name ***';
$keyname1 = '*** Provide object key ***';
$filepath = '*** Provide file name to upload ***';

$keyname2 = '*** Provide another object key ***';

// Instantiate the class
$s3 = new AmazonS3();

// 1. Create object from a file.
$response = $s3->create_object(
    $bucket,
    $keyname1,
    array('fileUpload' => $filepath)
    );

// Success?
print_r($response->isOK());
if ($response->isOK())
{
    echo "First upload successful!";
}

// 2. Create object from a string.
$response = $s3->create_object(
    $bucket,
    $keyname2,
    array(
       'body'        => 'This is object content',
       'acl'         => AmazonS3::ACL_PUBLIC,
       'contentType' => 'text/plain',
       'storage'     => AmazonS3::STORAGE_REDUCED,
       'headers'     => array( // raw headers
                          'Cache-Control' => 'max-age',
                          'Content-Encoding' => 'gzip',
                          'Content-Language' => 'en-US',
                          'Expires' => 'Thu, 01 Dec 2010 16:00:00 GMT',
          ),
       'meta'     => array(
                          'param1' => 'value 1',
                          'param2' => 'value 2'
          )
    )
    );

// Success?
print_r($response->isOK());
if ($response->isOK())
{
    echo "Second upload successful!";
}
else
{
    print_r($response);
}
```

Upload an Object Using the AWS SDK for Ruby

The following tasks guide you through using the Ruby classes to upload a file. The API provides provides a `#write` method that can take options that you can use to specify how to upload your data.

Uploading Objects

1	Create an instance of the `AWS::S3` class by providing your AWS credentials.
2	Use the `AWS::S3::S3Object#write` method which takes a data parameter and options hash which allow you to upload data from a file, or a stream.

The following Ruby code sample demonstrates the preceding tasks and uses the *`options`* hash *`:file`* to specify the path to the file to upload.

```
s3 = AWS::S3.new

# Upload a file.
key = File.basename(file_name)
s3.buckets[bucket_name].objects[key].write(:file => file_name)
```

Example

The following Ruby script example uploads a file to an Amazon S3 bucket. For instructions about how to create and test a working sample, see Using the AWS SDK for Ruby (p. 437).

```
#!/usr/bin/env ruby

require 'rubygems'
require 'aws-sdk'

AWS.config(
  :access_key_id => '*** Provide your access key ***',
  :secret_access_key => '*** Provide your secret key ***'
)

bucket_name = '*** Provide bucket name ***'
file_name = '*** Provide file name ****'

# Get an instance of the S3 interface.
s3 = AWS::S3.new

# Upload a file.
key = File.basename(file_name)
s3.buckets[bucket_name].objects[key].write(:file => file_name)
puts "Uploading file #{file_name} to bucket #{bucket_name}."
```

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, go to PUT Object.

Uploading Objects Using Multipart Upload API

Topics

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. 139).

Using the AWS SDK for Java for Multipart Upload

Topics

As described in Using the AWS SDK (see Using the AWS SDKs and Explorers (p. 433)), the SDK for Java provides support for the multipart upload API (see Uploading Objects Using Multipart Upload API (p. 167)). This section provides examples of using the both high-level and the low-level Java SDK API for uploading objects in parts.

Using the High-Level Java API for Multipart Upload

Topics

The AWS SDK for Java exposes a high-level API that simplifies multipart upload (see Uploading Objects Using Multipart Upload API (p. 167)). You can upload data from a file or a stream. You can optionally set advanced options, such as the part size you want to use for the multipart upload, or the number of threads you want to use when uploading the parts concurrently. You can also set optional object properties, the storage class, or ACL. You use the `PutObjectRequest` and the `TransferManagerConfiguration` classes to set these advanced options. The `TransferManager` class of the Java API provides the high-level API for you to upload data.

In addition to file upload functionality, the `TransferManager` class provides a method for you to abort multipart upload in progress. You must provide a `Date` value, and then the API aborts all the multipart uploads that were initiated before the specified date.

Upload a File

The following tasks guide you through using the high-level Java classes to upload a file. The API provides several variations, called *overloads*, of the `upload` method to easily upload your data.

High-Level API File Uploading Process

1	Create an instance of the `TransferManager` class by providing your AWS credentials.
2	Execute one of the `TransferManager.upload` overloads depending on whether you are uploading data from a file, or a stream.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);
// Asynchronous call.
Upload upload = tm.upload(existingBucketName, keyName, new File(filePath));
```

Example

The following Java code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class HighLevel_Java_UploadFile {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide existing bucket name ***";
        String keyName            = "*** Provide object key ***";
        String filePath           = "*** Provide file to upload ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
          HighLevel_Java_UploadFile.class.getResourceAsStream(
                                 "AwsCredentials.properties")));

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(
          existingBucketName, keyName, new File(filePath));

        try {
         // Or you can block and wait for the upload to finish
         upload.waitForCompletion();
        } catch (AmazonClientException amazonClientException) {
         System.out.println("Unable to upload file, upload was aborted.");
         amazonClientException.printStackTrace();
        }
    }
}
```

Abort Multipart Uploads

The `TransferManager` class provides a method, `abortMultipartUploads`, to abort multipart uploads in progress. An upload is considered to be in progress once you initiate it and until you complete it or abort it. You provide a `Date` value and this API aborts all the multipart uploads, on that bucket, that were initiated before the specified `Date` and are still in progress.

Because you are billed for all storage associated with uploaded parts (see Multipart Upload and Pricing (p. 140)), it is important that you either complete the multipart upload to have the object created or abort the multipart upload to remove any uploaded parts.

The following tasks guide you through using the high-level Java classes to abort multipart uploads.

High-Level API Multipart Uploads Aborting Process

1	Create an instance of the `TransferManager` class by providing your AWS credentials.
2	Execute the `TransferManager.abortMultipartUploads` method by passing the bucket name and a `Date` value.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);
tm.abortMultipartUploads(existingBucketName, someDate);
```

Example

The following Java code aborts all multipart uploads in progress that were initiated on a specific bucket over a week ago. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.util.Date;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.transfer.TransferManager;

public class HighLevel_Java_AbortMultipartUploads {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide existing bucket name ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
          HighLevel_Java_UploadFile.class.getResourceAsStream(
                           "AwsCredentials.properties")));

        int sevenDays = 1000 * 60 * 60 * 24 * 7;
  Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDays);

        try {
         tm.abortMultipartUploads(existingBucketName, oneWeekAgo);
        } catch (AmazonClientException amazonClientException) {
         System.out.println("Unable to upload file, upload was aborted.");
         amazonClientException.printStackTrace();
        }
    }
}
```

Note
You can also abort a specific multipart upload. For more information, see Abort a Multipart Upload (p. 177).

Track Multipart Upload Progress

The high-level multipart upload API provides a listen interface, `ProgressListener`, to track the upload progress when uploading data using the `TransferManager` class. To use the event in your code, you must import the `com.amazonaws.services.s3.model.ProgressEvent` and `com.amazonaws.services.s3.model.ProgressListener` types.

Progress events occurs periodically and notify the listener that bytes have been transferred.

The following Java code sample demonstrates how you can subscribe to the `ProgressEvent` event and write a handler.

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);

PutObjectRequest request = new PutObjectRequest(
    existingBucketName, keyName, new File(filePath));

// Subscribe to the event and provide event handler.
request.setProgressListener(new ProgressListener() {
   public void progressChanged(ProgressEvent event) {
    System.out.println("Transferred bytes: " +
      event.getBytesTransfered());
             }
});
```

Example

The following Java code uploads a file and uses the `ProgressListener` to track the upload progress. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.model.ProgressEvent;
import com.amazonaws.services.s3.model.ProgressListener;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class HighLevel_Java_UploadFile {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide bucket name ***";
        String keyName            = "*** Provide object key ***";
        String filePath           = "** Provide file to upload ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
         HighLevel_Java_UploadFile.class.getResourceAsStream(
                                 "AwsCredentials.properties")));

        // For more advanced uploads, you can create a request object
        // and supply additional request parameters (ex: progress listeners,
        // canned ACLs, etc)
        PutObjectRequest request = new PutObjectRequest(
          existingBucketName, keyName, new File(filePath));

        // You can ask the upload for its progress, or you can
        // add a ProgressListener to your request to receive notifications
        // when bytes are transfered.
        request.setProgressListener(new ProgressListener() {
   public void progressChanged(ProgressEvent event) {
    System.out.println("Transferred bytes: " +
      event.getBytesTransfered());
   }
  });

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(request);

        try {
         // You can block and wait for the upload to finish
         upload.waitForCompletion();
        } catch (AmazonClientException amazonClientException) {
         System.out.println("Unable to upload file, upload aborted.");
         amazonClientException.printStackTrace();
        }
    }
}
```

Using the Low-Level Java API for Multipart Upload

Topics

The AWS SDK for Java exposes a low-level API that closely resembles to the Amazon S3 REST API for multipart upload (see Uploading Objects Using Multipart Upload API (p. 167).

Upload a File

The following tasks guide you through using the low-level Java classes to upload a file.

Low-Level API File Uploading Process

1	Create an instance of the `AmazonS3Client` class, by providing your AWS credentials.
2	Initiate multipart upload by executing the `AmazonS3Client.initiateMultipartUpload` method. You will need to provide information required, i.e., bucket name and key name, to initiate the multipart upload by creating an instance of the `InitiateMultipartUploadRequest` class.
3	Save the upload ID that the `AmazonS3Client.initiateMultipartUpload` method returns. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Upload parts. For each part upload, execute the `AmazonS3Client.uploadPart` method. You need to provide part upload information, such as upload ID, bucket name, and the part number. You provide this information by creating an instance of the `UploadPartRequest` class.
5	Save the response of the `AmazonS3Client.uploadPart` method in a list. This response includes the ETag value and the part number you will need to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the `AmazonS3Client.completeMultipartUpload` method to complete the multipart upload.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
              LowLevel_Java_UploadFile.class.getResourceAsStream(
                                             "AwsCredentials.properties")));

// Create a list of UploadPartResponse objects. You get one of these for
// each part upload.
List<PartETag> partETags = new ArrayList<PartETag>();

// Step 1: Initialize.
InitiateMultipartUploadRequest initRequest = new InitiateMultipartUploadRequest(

                                                 existingBucketName, keyName);
InitiateMultipartUploadResult initResponse =
                              s3Client.initiateMultipartUpload(initRequest);

File file = new File(filePath);
long contentLength = file.length();
long partSize = 5 * 1024 * 1024; // Set part size to 5 MB.

try {
    // Step 2: Upload parts.
    long filePosition = 0;
    for (int i = 1; filePosition < contentLength; i++) {
        // Last part can be less than 5 MB. Adjust part size.
     partSize = Math.min(partSize, (contentLength - filePosition));

        // Create request to upload a part.
        UploadPartRequest uploadRequest = new UploadPartRequest()
            .withBucketName(existingBucketName).withKey(keyName)
            .withUploadId(initResponse.getUploadId()).withPartNumber(i)
            .withFileOffset(filePosition)
            .withFile(file)
            .withPartSize(partSize);

        // Upload part and add response to our list.
        partETags.add(s3Client.uploadPart(uploadRequest).getPartETag());

        filePosition += partSize;
    }

    // Step 3: complete.
    CompleteMultipartUploadRequest compRequest = new
                CompleteMultipartUploadRequest(existingBucketName,
                                               keyName,
                                               initResponse.getUploadId(),
                                               partETags);

    s3Client.completeMultipartUpload(compRequest);
} catch (Exception e) {
    s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
              existingBucketName, keyName, initResponse.getUploadId()));
}
```

Example

The following Java code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AbortMultipartUploadRequest;
import com.amazonaws.services.s3.model.CompleteMultipartUploadRequest;
import com.amazonaws.services.s3.model.InitiateMultipartUploadRequest;
import com.amazonaws.services.s3.model.InitiateMultipartUploadResult;
import com.amazonaws.services.s3.model.ListMultipartUploadsRequest;
import com.amazonaws.services.s3.model.ListPartsRequest;
import com.amazonaws.services.s3.model.MultipartUploadListing;
import com.amazonaws.services.s3.model.PartETag;
import com.amazonaws.services.s3.model.PartListing;
import com.amazonaws.services.s3.model.UploadPartRequest;

public class LowLevel_Java_UploadFile {

    public static void main(String[] args) throws IOException {
        String existingBucketName="*** Provide-Your-Existing-BucketName ***";

        String keyName             = "*** Provide-Key-Name ***";
        String filePath            = "*** Provide-File-Path ***";

        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
          LowLevel_Java_UploadFile.class.getResourceAsStream(
                              "AwsCredentials.properties")));

        // Create a list of UploadPartResponse objects. You get one of these
        // for each part upload.
        List<PartETag> partETags = new ArrayList<PartETag>();

        // Step 1: Initialize.
        InitiateMultipartUploadRequest initRequest = new
             InitiateMultipartUploadRequest(existingBucketName, keyName);
        InitiateMultipartUploadResult initResponse =
                            s3Client.initiateMultipartUpload(initRequest);

        File file = new File(filePath);
        long contentLength = file.length();
        long partSize = 5242880; // Set part size to 5 MB.

        try {
            // Step 2: Upload parts.
            long filePosition = 0;
            for (int i = 1; filePosition < contentLength; i++) {
                // Last part can be less than 5 MB. Adjust part size.
             partSize = Math.min(partSize, (contentLength - filePosition));

                // Create request to upload a part.
                UploadPartRequest uploadRequest = new UploadPartRequest()
                    .withBucketName(existingBucketName).withKey(keyName)
                   .withUploadId(initResponse.getUploadId()).withPartNumber(i)

                    .withFileOffset(filePosition)
                    .withFile(file)
                    .withPartSize(partSize);

                // Upload part and add response to our list.
                partETags.add(
                  s3Client.uploadPart(uploadRequest).getPartETag());

                filePosition += partSize;
            }

            // Step 3: complete.
            CompleteMultipartUploadRequest compRequest = new
                         CompleteMultipartUploadRequest(
                                    existingBucketName,
                                    keyName,
                                    initResponse.getUploadId(),
                                    partETags);

            s3Client.completeMultipartUpload(compRequest);
        } catch (Exception e) {
            s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
                    existingBucketName, keyName, initResponse.getUploadId()));

        }
    }
}
```

List Multipart Uploads

The following tasks guide you through using the low-level Java classes to list all in-progress multipart uploads on a bucket.

Low-Level API Multipart Uploads Listing Process

1	Create an instance of the `ListMultipartUploadsRequest` class and provide the bucket name.
2	Execute the `AmazonS3Client.listMultipartUploads` method. The method returns an instance of the `MultipartUploadListing` class that gives you information about the multipart uploads in progress .

The following Java code sample demonstrates the preceding tasks.

```
ListMultipartUploadsRequest allMultpartUploadsRequest =
     new ListMultipartUploadsRequest(existingBucketName);
MultipartUploadListing multipartUploadListing =
     s3Client.listMultipartUploads(allMultpartUploadsRequest);
```

Abort a Multipart Upload

You can abort an in-progress multipart upload by calling the `AmazonS3.abortMultipartUpload` method. This method deletes any parts that were uploaded to Amazon S3 and frees up the resources. You must provide the upload ID, bucket name, and the key name. The following Java code sample demonstrates how you can abort a multipart upload in progress.

```
AWSCredentials myCredentials = new
               BasicAWSCredentials(myAccessKeyID, mySecretKey);

InitiateMultipartUploadRequest initRequest =
    new InitiateMultipartUploadRequest(existingBucketName, keyName);
InitiateMultipartUploadResult initResponse =
               s3Client.initiateMultipartUpload(initRequest);

AmazonS3 s3Client = new AmazonS3Client(myCredentials);
s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
            existingBucketName, keyName, initResponse.getUploadId()));
```

Note
Instead of a specific multipart upload, you can abort all your multipart uploads initiated before a specific time that are still in progress. This clean-up operation is useful to abort old multipart uploads that you initiated but neither completed or aborted. For more information, see Abort Multipart Uploads (p. 169).

Using the AWS SDK for .NET for Multipart Upload

Topics

As described in Using the AWS SDK (see Using the AWS SDKs and Explorers (p. 433)), the SDK for .NET provides support for the multipart upload API (see Uploading Objects Using Multipart Upload API (p. 167)). This section provides examples of using the high-level and the low-level Java SDK API for uploading objects in parts.

Using the High-Level .NET API for Multipart Upload

Topics

The AWS SDK for .NET exposes a high-level API that simplifies multipart upload (see Uploading Objects Using Multipart Upload API (p. 167)). You can upload data from a file, directory, or a stream. When uploading data from a file, if you don't provide the object's key name, the API uses the file name for the object's key name. You must provide the object's key name if you are uploading data from a stream. You can optionally set advanced options such as the part size you want to use for the multipart upload, number of threads you want to use when uploading the parts concurrently, optional file metadata, the storage class (STANDARD or REDUCED_REDUNDANCY), or ACL. The high-level API provides the `TransferUtilityUploadRequest` class to set these advanced options.

The `TransferUtility` class provides a method for you to abort multipart uploads in progress. You must provide a `DateTime` value, and then the API aborts all the multipart uploads that were initiated before the specified date and time.

Upload a File

The following tasks guide you through using the high-level .NET classes to upload a file. The API provides several variations, *overloads*, of the `Upload` method to easily upload your data.

High-Level API File Uploading Process

1	Create an instance of the `TransferUtility` class by providing your AWS credentials.
2	Execute one of the `TransferUtility.Upload` overloads depending on whether you are uploading data from a file, a stream, or a directory.

The following C# code sample demonstrates the preceding tasks.

```
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey);
utility.Upload(filePath, existingBucketName);
```

When uploading large files using the .NET API, timeout might occur even while data is being written to the request stream. You can set explicit timeout using the `TransferUtilityConfig.DefaultTimeout` as demonstrated in the following C# code sample.

```
TransferUtilityConfig config = new TransferUtilityConfig();
config.DefaultTimeout = 11111;
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey,
config);
```

Example

The following C# code example uploads a file to an Amazon S3 bucket. The example illustrates the use of various `TransferUtility.Upload` overloads to upload a file; each successive call to upload replaces the previous upload. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;
using System.IO;

namespace s3.amazon.com.docsamples.highlevel_uploadfile
{
    class Program
    {
        static string accessKeyID     = "";
        static string secretAccessKey = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string keyName        = "*** Provide your object key ***";
        static string filePath       = "*** Provide file name ***";

        static void Main(string[] args)
        {

            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID     = appConfig["AWSAccessKey"];
            secretAccessKey = appConfig["AWSSecretKey"];

            try
            {
                TransferUtility fileTransferUtility = new
                    TransferUtility(accessKeyID, secretAccessKey);

                // 1. Upload a file, file name is used as the object key name.

                fileTransferUtility.Upload(filePath, existingBucketName);
                Console.WriteLine("Upload 1 completed");

                // 2. Specify object key name explicitly.
                fileTransferUtility.Upload(filePath,
                                          existingBucketName, keyName);
                Console.WriteLine("Upload 2 completed");

                // 3. Upload data from a type of System.IO.Stream.
                using (FileStream fileToUpload =
                    new FileStream(filePath, FileMode.Open, FileAccess.Read))
                {
                    fileTransferUtility.Upload(fileToUpload,
                                               existingBucketName, keyName);
                }
                Console.WriteLine("Upload 3 completed");

                // 4.// Specify advanced settings/options.
                TransferUtilityUploadRequest fileTransferUtilityRequest =
                    new TransferUtilityUploadRequest()
                    .WithBucketName(existingBucketName)
                    .WithFilePath(filePath)
                    .WithStorageClass(S3StorageClass.ReducedRedundancy)
                    .WithMetadata("param1", "Value1")
                    .WithMetadata("param2", "Value2")
                    .WithPartSize(6291456) // This is 6 MB.
                    .WithKey(keyName)
                    .WithCannedACL(S3CannedACL.PublicRead);
                fileTransferUtility.Upload(fileTransferUtilityRequest);
                Console.WriteLine("Upload 4 completed");
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
        }
    }
}
```

Upload a Directory

Using the `TransferUtility` class you can also upload an entire directory. By default, Amazon S3 only uploads the files at the root of the specified directory. You can, however, specify to recursively upload files in all the subdirectories.

You can also specify filtering expressions to select files, in the specified directory, based on some filtering criteria. For example, to upload only the .pdf files from a directory you specify a "*.pdf" filter expression.

When uploading files from a directory you cannot specify the object's key name. It is constructed from the file's location in the directory as well as its name. For example, assume you have a directory, c:\myfolder, with the following structure:

```
C:\myfolder
      \a.txt
      \b.pdf
      \media\
             An.mp3
```

When you upload this directory, Amazon S3 uses the following key names:

```
a.txt
b.pdf
media/An.mp3
```

The following tasks guide you through using the high-level .NET classes to upload a directory.

High-Level API Directory Uploading Process

1	Create an instance of the `TransferUtility` class by providing your AWS credentials.
2	Execute one of the `TransferUtility.UploadDirectory` overloads.

The following C# code sample demonstrates the preceding tasks.

```
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey);
utility.UploadDirectory(directoryPath, existingBucketName);
```

Example

The following C# code example uploads a directory to an Amazon S3 bucket. The example illustrates the use of various `TransferUtility.UploadDirectory` overloads to upload a directory, each successive call to upload replaces the previous upload. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;
using System.IO;

namespace s3.amazon.com.docsamples.highlevel_uploadDirectory
{
    class Program
    {
        static string accessKeyID        = "";
        static string secretAccessKey    = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string directoryPath = "*** Provide directory name ***";

        static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID        = appConfig["AWSAccessKey"];
            secretAccessKey    = appConfig["AWSSecretKey"];

             try
            {
                TransferUtility directoryTransferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);

                // 1. Upload a directory.
                directoryTransferUtility.UploadDirectory(directoryPath,
                                                         existingBucketName);
                Console.WriteLine("Upload statement 1 completed");

                // 2. Upload only the .txt files from a directory.
                //    Also, search recursively.
                directoryTransferUtility.UploadDirectory(
                                               directoryPath,
                                               existingBucketName,
                                               "*.txt",
                                               SearchOption.AllDirectories);
                Console.WriteLine("Upload statement 2 completed");

                // 3. Same as 2 and some optional configuration
                //    Search recursively for .txt files to upload).
                TransferUtilityUploadDirectoryRequest trUtilDirUpReq =
                    new TransferUtilityUploadDirectoryRequest()
                .WithBucketName(existingBucketName)
                .WithDirectory(directoryPath)
                .WithSearchOption(SearchOption.AllDirectories)
                .WithSearchPattern("*.txt");
                directoryTransferUtility.UploadDirectory(trUtilDirUpReq);
                Console.WriteLine("Upload statement 3 completed");
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
        }
    }
}
```

Abort Multipart Uploads

The `TransferUtility` class provides a method, `AbortMultipartUploads`, to abort multipart uploads in progress. An upload is considered to be in-progress once you initiate it and until you complete it or abort it. You provide a `DateTime` value and this API aborts all the multipart uploads, on that bucket, that were initiated before the specified `DateTime` and in progress.

Because you are billed for all storage associated with uploaded parts (see Multipart Upload and Pricing (p. 140)), it is important that you either complete the multipart upload to have the object created or abort the multipart upload to remove any uploaded parts.

The following tasks guide you through using the high-level .NET classes to abort multipart uploads.

High-Level API Multipart Uploads Aborting Process

1	Create an instance of the `TransferUtility` class by providing your AWS credentials.
2	Execute the `TransferUtility.AbortMultipartUploads` method by passing the bucket name and a `DateTime` value.

The following C# code sample demonstrates the preceding tasks.

```
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey);
utility.AbortMultipartUploads(existingBucketName, DateTime.Now.AddDays(-7));
```

Example

The following C# code aborts all multipart uploads in progress that were initiated on a specific bucket over a week ago. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;

namespace s3.amazon.com.docsamples.highlevel_abortmultipartupload
{
    class Program
    {
        static string accessKeyID        = "";
        static string secretAccessKey    = "";

        static string existingBucketName = "***Provide bucket name***";

        static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID        = appConfig["AWSAccessKey"];
            secretAccessKey    = appConfig["AWSSecretKey"];

            try
            {
                TransferUtility transferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);
                // Aborting uploads that were initiated over a week ago.
                transferUtility.AbortMultipartUploads(
                    existingBucketName, DateTime.Now.AddDays(-7));
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
        }
    }
}
```

Note
You can also abort a specific multipart upload. For more information, see List Multipart Uploads (p. 192).

Track Multipart Upload Progress

The high-level multipart upload API provides an event, `TransferUtilityUploadRequest.UploadProgressEvent`, to track the upload progress when uploading data using the `TransferUtility` class.

The event occurs periodically and returns multipart upload progress information such as the total number of bytes to transfer, and the number of bytes transferred at the time event occurred.

The following C# code sample demonstrates how you can subscribe to the `UploadProgressEvent` event and write a handler.

```
TransferUtility fileTransferUtility =
        new TransferUtility(accessKeyID, secretAccessKey);

// Use TransferUtilityUploadRequest to configure options.
// In this example we subscribe to an event.
TransferUtilityUploadRequest uploadRequest =
               new TransferUtilityUploadRequest()
               .WithBucketName(existingBucketName)
               .WithFilePath(filePath);

uploadRequest.UploadProgressEvent +=
         new EventHandler<UploadProgressArgs>(uploadRequest_UploadPartProgres
sEvent);

fileTransferUtility.Upload(uploadRequest);

static void uploadRequest_UploadPartProgressEvent(object sender, UploadProgres
sArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
```

Example

The following C# code example uploads a file to an Amazon S3 bucket and tracks the progress by subscribing to the `TransferUtilityUploadRequest.UploadProgressEvent` event. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;
using System.IO;


namespace s3.amazon.com.docsamples.highlevel_trackprogress
{
    class Program
    {
        static string accessKeyID        = "";
        static string secretAccessKey    = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string keyName            = "*** Provide key name ***";
        static string filePath        = "*** Provide file to upload ***";

        static void Main(string[] args)
        {

            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID        = appConfig["AWSAccessKey"];
            secretAccessKey    = appConfig["AWSSecretKey"];

            try
            {
                TransferUtility fileTransferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);

                // Use TransferUtilityUploadRequest to configure options.
                // In this example we subscribe to an event.
                TransferUtilityUploadRequest uploadRequest =
                    new TransferUtilityUploadRequest()
                    .WithBucketName(existingBucketName)
                    .WithFilePath(filePath);

                uploadRequest.UploadProgressEvent +=
                    new EventHandler<UploadProgressArgs>
                        (uploadRequest_UploadPartProgressEvent);

                fileTransferUtility.Upload(uploadRequest);
                Console.WriteLine("Upload completed");
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
        }
        static void uploadRequest_UploadPartProgressEvent(
            object sender, UploadProgressArgs e)
        {
            // Process event.
            Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
        }
    }
}
```

Using the Low-Level .NET API for Multipart Upload

Topics

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. 199)). 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 data in advance. Use the high-level API (see Using the High-Level .NET API for Multipart Upload (p. 178)), whenever you don't have these requirements.

Upload a File

The following tasks guide you through using the low-level .NET classes to upload a file.

Low-Level API File UploadingProcess

1	Create an instance of the `AmazonS3Client` class, by providing your AWS credentials.
2	Initiate multipart upload by executing the `AmazonS3Client.InitiateMultipartUpload` method. You will need to provide information required to initiate the multipart upload by creating an instance of the `InitiateMultipartUploadRequest` class.
3	Save the Upload ID that the `AmazonS3Client.InitiateMultipartUpload` method returns. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Upload the parts. For each part upload, execute the `AmazonS3Client.UploadPart` method. You will need to provide part upload information such as upload ID, bucket name, and the part number. You provide this information by creating an instance of the `UploadPartRequest` class.
5	Save the response of the `AmazonS3Client.UploadPart` method in a list. This response includes the ETag value and the part number you will later need to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the `AmazonS3Client.CompleteMultipartUpload` method to complete the multipart upload.

The following C# code sample demonstrates the preceding tasks.

```
AmazonS3 s3Client = new AmazonS3Client(AccessKeyID, SecretAccessKey);

// List to store upload part responses.
List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

// 1. Initialize.
InitiateMultipartUploadRequest initRequest =
    new InitiateMultipartUploadRequest()
    .WithBucketName(existingBucketName)
    .WithKey(keyName);

InitiateMultipartUploadResponse initResponse =
    s3Client.InitiateMultipartUpload(initRequest);

// 2. Upload Parts.
long contentLength = new FileInfo(filePath).Length;
long partSize = 5242880; // 5 MB

try
{
    long filePosition = 0;
    for (int i = 1; filePosition < contentLength; i++)
    {

        // Create request to upload a part.
        UploadPartRequest uploadRequest = new UploadPartRequest()
            .WithBucketName(existingBucketName)
            .WithKey(keyName)
            .WithUploadId(initResponse.UploadId)
            .WithPartNumber(i)
            .WithPartSize(partSize)
            .WithFilePosition(filePosition)
            .WithFilePath(filePath);

        // Upload part and add response to our list.
         uploadResponses.Add(s3Client.UploadPart(uploadRequest));

        filePosition += partSize;
    }

    // Step 3: complete.
    CompleteMultipartUploadRequest compRequest =
        new CompleteMultipartUploadRequest()
        .WithBucketName(existingBucketName)
        .WithKey(keyName)
        .WithUploadId(initResponse.UploadId)
        .WithPartETags(uploadResponses);

    CompleteMultipartUploadResponse completeUploadResponse =
        s3Client.CompleteMultipartUpload(compRequest);

}
catch (Exception exception)
{
    Console.WriteLine("Exception occurred: {0}", exception.Message);
    s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest()
        .WithBucketName(existingBucketName)
        .WithKey(keyName)
        .WithUploadId(initResponse.UploadId));
}
```

Note

When uploading large objects using the .NET API, timeout might occur even while data is being written to the request stream. You can set explicit timeout using the `UploadPartRequest`.

Example

The following C# code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Configuration;
using System.IO;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.LowLevel_UploadFromFile
{
    class Program
    {
        // Your AWS Credentials.
        static string AccessKeyID        = "";
        static string SecretAccessKey    = "";

        static string existingBucketName = "*** Provide bucket name";
        static string keyName = "*** Provide object key ***";
        static string filePath = "*** Provide file to upload ***";

         static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            AccessKeyID        = appConfig["AWSAccessKey"];
            SecretAccessKey    = appConfig["AWSSecretKey"];

            AmazonS3 s3Client =
                new AmazonS3Client(AccessKeyID, SecretAccessKey);

            // List to store upload part responses.
            List<UploadPartResponse> uploadResponses =
                new List<UploadPartResponse>();

            // 1. Initialize.
            InitiateMultipartUploadRequest initiateRequest =
                new InitiateMultipartUploadRequest()
                .WithBucketName(existingBucketName)
                .WithKey(keyName);

            InitiateMultipartUploadResponse initResponse =
                s3Client.InitiateMultipartUpload(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++)
                {

                    // Create request to upload a part.
                    UploadPartRequest uploadRequest = new UploadPartRequest()
                        .WithBucketName(existingBucketName)
                        .WithKey(keyName)
                        .WithUploadId(initResponse.UploadId)
                        .WithPartNumber(i)
                        .WithPartSize(partSize)
                        .WithFilePosition(filePosition)
                        .WithFilePath(filePath);

                    // Upload part and add response to our list.
                    uploadResponses.Add(s3Client.UploadPart(uploadRequest));

                    filePosition += partSize;
                }

                // Step 3: complete.
                CompleteMultipartUploadRequest completeRequest =
                    new CompleteMultipartUploadRequest()
                    .WithBucketName(existingBucketName)
                    .WithKey(keyName)
                    .WithUploadId(initResponse.UploadId)
                    .WithPartETags(uploadResponses);

                CompleteMultipartUploadResponse completeUploadResponse =
                    s3Client.CompleteMultipartUpload(completeRequest);

            }
            catch (Exception exception)
            {
              Console.WriteLine("Exception occurred: {0}", exception.Message);

               s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest()

                    .WithBucketName(existingBucketName)
                    .WithKey(keyName)
                    .WithUploadId(initResponse.UploadId));
            }
        }
    }
}
```

List Multipart Uploads

The following tasks guide you through using the low-level .NET classes to list all in-progress multipart uploads on a bucket.

Low-Level API Multipart Uploads Listing Process

1	Create an instance of the `ListMultipartUploadsRequest` class and provide the bucket name.
2	Execute the `AmazonS3Client.ListMultipartUploads` method. The method returns an instance of the `ListMultipartUploadsResponse` class, providing you the information about the in-progress multipart uploads.

The following C# code sample demonstrates the preceding tasks.

```
ListMultipartUploadsRequest allMultipartUploadsRequest = new ListMultipartUp
loadsRequest()
   .WithBucketName(existingBucketName);
ListMultipartUploadsResponse mpUploadsResponse = s3Client.ListMultipartUp
loads(allMultipartUploadsRequest);
```

Track Multipart Upload Progress

The low-level multipart upload API provides an event, `UploadPartRequest.UploadPartProgressEvent`, to track the upload progress.

The event occurs periodically and returns multipart upload progress information such as the total number of bytes to transfer, and the number of bytes transferred at the time event occurred.

The following C# code sample demonstrates how you can subscribe to the `UploadPartProgressEvent` event and write a handler.

```
UploadPartRequest uploadRequest = new UploadPartRequest();
// Provide request data (for example, bucket name, key name and part number).
// ...
// Subscribe to the event.
uploadRequest.UploadPartProgressEvent += new
EventHandler<UploadPartProgressArgs>(uploadRequest_UploadPartProgressEvent);

// Sample event handler.
static void uploadRequest_UploadPartProgressEvent(object sender,
                                                  UploadPartProgressArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
```

Abort a Multipart Upload

You can abort an in-progress multipart upload by calling the AmazonS3Client.AbortMultipartUpload method. This method deletes any parts that were uploaded to S3 and free up the resources. You must provide the upload ID, bucket name and the key name. The following C# code sample demonstrates how you can abort a multipart upload in progress.

```
s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest()
    .WithBucketName(existingBucketName)
    .WithKey(keyName)
    .WithUploadId(uploadID));
```

Note
Instead of a specific multipart upload, you can abort all your in-progress multipart uploads initiated prior to a specific time. This clean up operation is useful to abort old multipart uploads that you initiated but neither completed or aborted. For more information, see Abort Multipart Uploads (p. 184).

Using the AWS SDK for PHP for Multipart Upload

Topics

The AWS SDK for PHP provides support for the multipart upload API (see Uploading Objects Using Multipart Upload API (p. 167)).

High-level API

The high-level API simplifies the multipart upload flow. In just a few lines of code you can upload files to Amazon S3. This is recommended for simple file uploads.

Low-level API

The low-level APIs correspond to the multipart upload REST operations (see Using the REST API for Multipart Upload (p. 199)). 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 data in advance. If you do not have these requirements, use the high-level API.

Using the High-Level PHP API for Multipart Upload

Topics

The AWS SDK for PHP exposes a high-level method that simplifies the multipart upload flow (see Uploading Objects Using Multipart Upload API (p. 167)). You can upload data from a file or an open file stream. You can optionally set advanced options, such as the part size you want to use for the multipart upload, optional file metadata, the storage class (STANDARD or REDUCED_REDUNDANCY), or ACL.

Upload a File

The following tasks guide you through using the high-level PHP classes to upload a file.

High-Level API File Uploading Process

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the `AmazonS3::create_mpu_object()` method.

The following PHP code sample demonstrates the preceding tasks.

```
// Instantiate the class
$s3 = new AmazonS3();

$response = $s3->create_mpu_object($bucket, $keyname, array(
    'fileUpload' => $filepath
));

print_r($response);
```

Example

The following PHP example uploads a file to an Amazon S3 bucket. The example illustrates the use of the `AmazonS3::create_mpu_object()` method to upload a file. Each successive call to upload replaces the previous upload.

```
<?php
require_once '/path/to/sdk.class.php';

$bucket = '*** Provide your existing bucket name ***';
$keyname = '*** Provide object key ***';
$filepath = '*** Provide file to upload ***';

// Define a mebibyte
define('MB', 1048576);

// Instantiate the class
$s3 = new AmazonS3();

// 1. Upload file. Specify object key name explicitly.
$response = $s3->create_mpu_object($bucket, $keyname, array(
    'fileUpload' => $filepath
));

// Success?
header('Content-Type: text/plain; charset=utf-8');
print_r($response);
echo "Upload 1 completed!" . PHP_EOL . PHP_EOL;

// 2. Specify optional configuration.
$response = $s3->create_mpu_object($bucket, $keyname, array(
    'fileUpload' => $filepath,

    // Optional configuration
    'partSize' => 40*MB, // Defaults to 50MB
    'acl' => AmazonS3::ACL_PUBLIC,
    'storage' => AmazonS3::STORAGE_REDUCED,

    // Object metadata.
    'meta' => array(
        'param1' => 'value1',
        'param2' => 'value2',
     )
));

// Success?
print_r($response);
echo "Upload 2 completed!";
```

Using the Low-Level PHP API for Multipart Upload

Topics

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. 199)). 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 data in advance. Use the high-level API (see Using the High-Level PHP API for Multipart Upload (p. 194)) whenever you don't have these requirements.

Upload a File

The following tasks guide you through using the low-level PHP classes to upload a file.

Low-Level API File UploadingProcess

1	Create an instance of the `AmazonS3` class, by providing your AWS credentials.
2	Initiate multipart upload by executing the `AmazonS3::initiate_multipart_upload()` method. You need to provide the required information, i.e., bucket name and object key.
3	Retrieve the *`UploadID`* from the response body. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Call the `AmazonS3::get_multipart_counts()` to generate parts information. You need to provide the input file and the desired part size. The method returns an associative array of all the parts information, such as the part length and its starting position in the file.
5	Upload parts by executing the `AmazonS3::upload_part()` method. Save the response of each of the `upload_part()` methods in an array. Each response includes the ETag value you will later need to complete the multipart upload.
6	Execute the `AmazonS3::complete_multipart_upload()` method to complete the multipart upload.

The following PHP code sample demonstrates the preceding tasks.

```
// Instantiate the class.
$s3 = new AmazonS3();

// Define a megabyte
define('MB', 1048576);

// 1. Initiate a new multipart upload.
$response = $s3->initiate_multipart_upload($bucket, $keyname);

// Get the Upload ID.
$upload_id = (string) $response->body->UploadId;

// 2. Upload parts.
// Get part list for a given input file and given part size.
// Returns an associative array.
$parts = $s3->get_multipart_counts(filesize($filepath), 5*MB);

$responses = new CFArray(array());

foreach ($parts as $i => $part)
{
    // Upload part and save response in an array.
    $responses[] = $s3->upload_part($bucket, $keyname, $upload_id, array(
        'fileUpload' => $filepath,
        'partNumber' => ($i + 1),
        'seekTo' => (integer) $part['seekTo'],
        'length' => (integer) $part['length'],
    ));
}

// 3. Complete multipart upload. We need all part numbers and ETag values.
$parts = $s3->list_parts($bucket, $keyname, $upload_id);

$response = $s3->complete_multipart_upload(
    $bucket, $keyname, $upload_id, $parts);
```

Example

The following PHP code example uploads a file to an Amazon S3 bucket.

```
<?php
require_once '/path/to/sdk.class.php';

$bucket = '*** Provide your existing bucket name ***';
$keyname = '*** Provide object key name ***';
$filepath = '*** Provide file name to upload ***';

// Define a megabyte.
define('MB', 1048576);

// Instantiate the class
$s3 = new AmazonS3();

// 1. Initiate a new multipart upload. (Array parameter is optional)
$response = $s3->initiate_multipart_upload($bucket, $keyname, array(
    'acl' => AmazonS3::ACL_PUBLIC,
    'storage' => AmazonS3::STORAGE_REDUCED,
    'meta' => array(
        'param1' => 'value 1',
        'param2' => 'value 2',
        'param3' => 'value 3'
    )
));

if (!$response->isOK())
{
    throw new S3_Exception('Bad!');
}

// Get the Upload ID.
$upload_id = (string) $response->body->UploadId;

// 2. Upload parts.
// Get part list for a given input file and given part size.
// Returns an associative array.
$parts = $s3->get_multipart_counts(filesize($filepath), 5*MB);

$responses = new CFArray(array());

foreach ($parts as $i => $part)
{
    // Upload part and save response in an array.
    $responses[] = $s3->upload_part($bucket, $keyname, $upload_id, array(
        'fileUpload' => $filepath,
        'partNumber' => ($i + 1),
        'seekTo' => (integer) $part['seekTo'],
        'length' => (integer) $part['length'],
    ));
}

// Verify that no part failed to upload, otherwise abort.
if (!$responses->areOK())
{
    // Abort an in-progress multipart upload
    $response = $s3->abort_multipart_upload($bucket, $keyname, $upload_id);

    throw new S3_Exception('Failed!');
}

// 3. Complete the multipart upload. We need all part numbers and ETag values.
$parts = $s3->list_parts($bucket, $keyname, $upload_id);
$response = $s3->complete_multipart_upload(
                             $bucket, $keyname, $upload_id, $parts);

// Display the results
header('Content-Type: text/plain; charset=utf-8');
print_r($response);

if ($response->isOK())
{
    echo 'Object uploaded!';
}
```

List Multipart Uploads

The following tasks guide you through using the low-level PHP classes to list all in-progress multipart uploads on a bucket.

Low-Level API Multipart Uploads Listing Process

1	Create an instance of the `AmazonS3` class.
2	Execute the `AmazonS3::list_multipart_uploads()` method by providing a bucket name. The method returns all of the in-progress multipart uploads on that bucket.

The following PHP code sample demonstrates the preceding tasks.

```
$s3 = new AmazonS3();
$response = $s3->list_multipart_uploads($bucket);
print_r($response);
```

Abort a Multipart Upload

You can abort a multipart upload that is in progress by calling the `AmazonS3::abort_multipart_upload()` method. This method deletes any parts that were uploaded to S3 and frees up the resources. You must provide the upload ID, bucket name, and the object key to this method. The following PHP code sample demonstrates how you can abort a multipart upload in progress.

```
$s3 = new AmazonS3();
$response = $s3->abort_multipart_upload($bucket, $keyname, $upload_id);
print_r($response);
```

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. 141).

Uploading Objects Using Pre-Signed URLs

Topics

A pre-signed URL gives you access to the object identified in the URL, provided that the creator of the pre-signed URL has permissions to access that object. That is, if you receive a pre-signed URL to upload an object, you can upload the object only if the creator of the pre-signed URL has the necessary permissions to upload that object.

All objects and buckets by default are private. The pre-signed URLs are useful if you want your user/customer to be able upload a specific object to your bucket, but you don't require them to have AWS security credentials or permissions. When you create a pre-signed URL, you must provide your security credentials, specify a bucket name an object key, an HTTP method (PUT of uploading objects) and an expiration date and time. The pre-signed URLs are valid only for the specified duration.

You can generate a pre-signed URL programmatically using the AWS SDK for Java or the AWS SDK for .NET. If you are using Visual Studio, you can also use the AWS Explorer to generate a pre-signed object URL without writing any code. Anyone who receives a valid pre-signed URL can then programmatically upload an 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 and Explorers (p. 433).

> **Note**
> Anyone with valid security credentials can create a pre-signed URL. However, in order to successfully upload an object, the pre-signed URL must be created by someone who has permission to perform the operation that the pre-signed URL is based upon.

Upload an Object Using Pre-Signed URL - AWS SDK for Java

The following tasks guide you through using the Java classes to upload an object using a pre-signed URL.

Uploading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials. These credentials are used in creating a signature for authentication when you generate a pre-signed URL.
2	Generate a pre-signed URL by executing the `AmazonS3.generatePresignedUrl` method. You provide a bucket name, an object key, and an expiration date by creating an instance of the `GeneratePresignedUrlRequest` class. You must specify the HTTP verb PUT when creating this URL if you want to use it to upload an object.
3	Anyone with the pre-signed URL can upload an object. The upload creates an object or replaces any existing object with the same key that is specified in the pre-signed URL.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

java.util.Date expiration = new java.util.Date();
long msec = expiration.getTime();
msec += 1000 * 60 * 60; // Add 1 hour.
expiration.setTime(msec);

GeneratePresignedUrlRequest generatePresignedUrlRequest = new GeneratePresigned
UrlRequest(bucketName, objectKey);
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
generatePresignedUrlRequest.setExpiration(expiration);

URL s = s3client.generatePresignedUrl(generatePresignedUrlRequest);

// Use the pre-signed URL to upload an object.
```

Example

The following Java code example generates a pre-signed URL The example code then uses the pre-signed URL to upload sample data as an object. For instructions about how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
import java.io.IOException;
import java.net.URL;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import java.net.*;
import java.io.*;

public class S3Sample {
 private static String bucketName = "ExampleBucket";
 private static String objectKey = "JavaObj.xml";

 public static void main(String[] args) throws IOException {
  AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
   S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

  try {
   System.out.println("Generating pre-signed URL.");
   java.util.Date expiration = new java.util.Date();
   long milliSeconds = expiration.getTime();
   milliSeconds += 1000 * 60 * 60; // Add 1 hour.
   expiration.setTime(milliSeconds);

   GeneratePresignedUrlRequest generatePresignedUrlRequest =
        new GeneratePresignedUrlRequest(bucketName, objectKey);
   generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
   generatePresignedUrlRequest.setExpiration(expiration);

   URL url = s3client.generatePresignedUrl(generatePresignedUrlRequest);

   UploadObject(url);

   System.out.println("Pre-Signed URL = " + url.toString());
  } catch (AmazonServiceException exception) {
   System.out.println("Caught an AmazonServiceException, " +
     "which means your request made it " +
     "to Amazon S3, but was rejected with an error response " +
   "for some reason.");
   System.out.println("Error Message: " + exception.getMessage());
   System.out.println("HTTP  Code: "    + exception.getStatusCode());
   System.out.println("AWS Error Code:" + exception.getErrorCode());
   System.out.println("Error Type:    " + exception.getErrorType());
   System.out.println("Request ID:    " + exception.getRequestId());
  } catch (AmazonClientException ace) {
   System.out.println("Caught an AmazonClientException, " +
     "which means the client encountered " +
     "an internal error while trying to communicate" +
      " with S3, " +
    "such as not being able to access the network.");
    System.out.println("Error Message: " + ace.getMessage());
  }
 }

 public static void UploadObject(URL url) throws IOException
 {
  HttpURLConnection connection=(HttpURLConnection) url.openConnection();
  connection.setDoOutput(true);
  connection.setRequestMethod("PUT");
  OutputStreamWriter out = new OutputStreamWriter(
    connection.getOutputStream());
  out.write("This text uploaded as object.");
  out.close();
  int responseCode = connection.getResponseCode();

 }
}
```

Upload an Object Using Pre-Signed URL - AWS SDK for .NET

The following tasks guide you through using the .NET classes to upload an object using a pre-signed URL.

Uploading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials. These credentials are used in creating a signature for authentication when you generate a pre-signed URL.
2	Generate a pre-signed URL by executing the `AmazonS3.GetPreSignedURL` method. You provide a bucket name, an object key, and an expiration date by creating an instance of the `GetPreSignedUrlRequest` class. You must specify the HTTP verb PUT when creating this URL if you plan to use it to upload an object.
3	Anyone with the pre-signed URL can upload an object. You can create an instance of the `HttpWebRequest` class by providing the pre-signed URL and uploading the object.

The following C# code sample demonstrates the preceding tasks.

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);
// Generate a pre-signed URL.
GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
request.WithBucketName(bucketName);
request.WithKey(objectKey);
request.Verb = HttpVerb.PUT;
request.WithExpires(DateTime.Now.AddMinutes(5));
string url = null;
 url = s3Client.GetPreSignedURL(request);

// Upload a file using the pre-signed URL.
```

```
HttpWebRequest httpRequest = WebRequest.Create(url) as HttpWebRequest;
httpRequest.Method = "PUT";
using (Stream dataStream = httpRequest.GetRequestStream())
{
   // Upload object.
}

HttpWebResponse response = httpRequest.GetResponse() as HttpWebResponse;
```

Example

The following C# code example generates a pre-signed URL for a specific object and uses it to upload a file. For instructions about how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;
using System.Net;
using System.IO;

namespace s3.amazon.com.docsamples.putobjectusingpresignedurl
{
    class S3Sample
    {
        static AmazonS3 s3Client;
        // File to upload.
        static string filePath = @"*** Specify file to upload ***";
        // Information to generate pre-signed object URL.
        static string bucketName = "*** Provide bucket name ***";
        static string objectKey = "*** Provide object key for the new object
***";

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];

            try
            {
               using (s3Client = Amazon.AWSClientFactory.CreateAmazonS3Client(

                     accessKeyID, secretAccessKeyID))
                {
                    string url = GeneratePreSignedURL();
                    UploadObject(url);

                }
            }
            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);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }


        static void UploadObject(string url)
        {
            HttpWebRequest httpRequest = WebRequest.Create(url) as HttpWebRe
quest;
            httpRequest.Method = "PUT";
            using (Stream dataStream = httpRequest.GetRequestStream())
            {
                byte[] buffer = new byte[8000];
                using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
                {
                    int bytesRead = 0;
                    while ((bytesRead = fileStream.Read(buffer, 0, buf
fer.Length)) > 0)
                    {
                        dataStream.Write(buffer, 0, bytesRead);
                    }
                }
            }

            HttpWebResponse response = httpRequest.GetResponse() as HttpWebRe
sponse;
        }

        static string GeneratePreSignedURL()
        {
            GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
            request.WithBucketName(bucketName);
            request.WithKey(objectKey);
            request.Verb = HttpVerb.PUT;
            request.WithExpires(DateTime.Now.AddMinutes(5));
            string url = null;


                url = s3Client.GetPreSignedURL(request);

            return url;
        }
    }
}
```

Upload an Object Using Pre-Signed URL - AWS SDK for Ruby

For information on how to use the AWS SDK for Ruby to upload an object using pre-signed URL, go to PresignedPost.

Copying Objects

Topics

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., Northern California and EU)

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 creation date of 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 the user configurable metadata, even if you are only changing only one of the metadata values, present on the source object in your request.

For more information about the object metadata, see Object Key and Metadata (p. 101).

Note
Copying objects across locations incurs bandwidth charges.

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

Copying Objects in a Single Operation

Topics

- Copy an Object Using the REST API (p. 217)

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. 218).

Copy an Object Using the AWS SDK for Java

The following tasks guide you through using the Java classes to copy an object in Amazon S3.

Copying Objects

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute one of the `AmazonS3Client.copyObject` methods. You need to provide the request information, such as source bucket name, source key name, destination bucket name, and destination key. You provide this information by creating an instance of the `CopyObjectRequest` class or optionally providing this information directly with the `AmazonS3Client.copyObject` method.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                 myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.copyObject(sourceBucketName, sourceKey,
                    destinationBucketName, destinationKey);
```

Example

The following Java code example makes a copy of an object. The copied object with a different key is saved in the same source bucket. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.io.IOException;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CopyObjectRequest;

public class S3Sample {
 private static String bucketName     = "*** Provide bucket name ***";
 private static String key            = "*** Provide key ***  ";
 private static String destinationKey = "*** Provide dest. key ***";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            // Copying object
            CopyObjectRequest copyObjRequest = new CopyObjectRequest(
              bucketName, key, bucketName, destinationKey);
            System.out.println("Copying object.");
            s3client.copyObject(copyObjRequest);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, " +
              "which means your request made it " +
              "to Amazon S3, but was rejected with an error " +
                    "response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, " +
              "which means the client encountered " +
                    "an internal error while trying to " +
                    " communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Copy an Object Using the AWS SDK for .NET

The following tasks guide you through using the high-level .NET classes to upload a file. The API provides several variations, *overloads*, of the `Upload` method to easily upload your data.

Copying Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute one of the `AmazonS3.CopyObject`. You need to provide information such as source bucket, source key name, target bucket, and target key name. You provide this information by creating an instance of the `CopyObjectRequest` class.

The following C# code sample demonstrates the preceding tasks.

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

CopyObjectRequest request = new CopyObjectRequest();
request.SourceBucket = bucketName;
request.SourceKey = keyName;
request.DestinationBucket = bucketName;
request.DestinationKey = destKeyName;
S3Response response = client.CopyObject(request);
```

Example

The following C# code example makes a copy of an object in the same source bucket. The example illustrates the use of the `AmazonS3.CopyObject` method. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using System.Collections.Specialized;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.copyobject
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static string keyName = "*** Provide key name ***";
        static string destKeyName = "*** Provide destination key name ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                    Console.WriteLine("Copying an object");
                    CopyingObject();
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }
            if (string.IsNullOrEmpty(keyName))
            {
                Console.WriteLine("The variable keyName is not set.");
                return false;
            }

            return true;
        }

        static void CopyingObject()
        {
            try
            {
                // simple object put
                CopyObjectRequest request = new CopyObjectRequest();
                request.SourceBucket = bucketName;
                request.SourceKey = keyName;
                request.DestinationBucket = bucketName;
                request.DestinationKey = destKeyName;
                S3Response response = client.CopyObject(request);
                response.Dispose();
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
        }
    }
}
```

Copy an Object Using the AWS SDK for PHP

The following tasks guide you through using the PHP classes to copy an object within Amazon S3, from one bucket to another or to copy an object within the same bucket.

Copying Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	To copy an object, execute the `AmazonS3::copy_object()` low-level method. You need to provide information such as source bucket, source key name, target bucket, and target key name. You provide this information as a `ComplexType` parameter, where a `ComplexType` is a set of key-value pairs. If you want to copy several objects, you can optionally execute these requests in parallel by adding the `copy_object` requests to the batch queue and executing the batch by calling the `batch.send()` method.

The following PHP code sample demonstrates the preceding tasks.

```
// Instantiate the class.
$s3 = new AmazonS3();

// 1. Copy one object.
$response = $s3->copy_object(array( // Source.
                               'bucket'   => $sourcebucket,
                               'filename' => $sourcekeyname
                             ),
                             array( // Target.
                               'bucket'   => $targetbucket,
                               'filename' => $targetkeyname
                             )
);

// 2. Copy objects using batch.
$s3->batch()->copy_object(array( // Source.
                            'bucket' => $sourcebucket,
                            'filename' => $sourcekeyname
                          ),
                          array( // Target.
                            'bucket'   => $targetbucket,
                            'filename' => $targetkeyname
                          )
);

$s3->batch()->copy_object(array( // Source.
                            'bucket'   => $sourcebucket,
                            'filename' => $sourcekeyname
                          ),
                          array( // Target.
                             'bucket'   => $targetbucket,
                             'filename' => $targetkeyname
                          )
);

$responses = $s3->batch()->send();
```

Example

The following PHP example illustrates the use of the `copy_object()` method to copy the single object. First, the example copies a single object. The example then illustrates the use of the batch operation, where it adds two `copy_object()` requests to the batch, and later sends the batch for parallel execution.

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';

$sourcebucket  = '*** Source bucket ***';
$sourcekeyname = '*** Source object key name ***';

$targetbucket  = '*** Target bucket ***';
$targetkeyname = '*** Target object key name ***';

// Instantiate the class.
$s3 = new AmazonS3();

// Copy object.
$response = $s3->copy_object(array( // Source.
                               'bucket'   => $sourcebucket,
                               'filename' => $sourcekeyname
                            ),
                            array( // Target.
                               'bucket'   => $targetbucket,
                               'filename' => $targetkeyname
                            )
);
// Success?
print_r($response->isOK());

// Add copy_object request to the batch to execute in parallel later.
$s3->batch()->copy_object(array( // Source.
                             'bucket'   => $sourcebucket,
                             'filename' => $sourcekeyname
                          ),
                          array( // Target.
                             'bucket'   => $targetbucket,
                             'filename' => $targetkeyname
                          )
);

$s3->batch()->copy_object(array( // Source.
                            'bucket'   => $sourcebucket,
                            'filename' => $sourcekeyname
                          ),
                          array( // Target.
                            'bucket'   => $targetbucket,
                            'filename' => $targetkeyname
                          )
);

$responses = $s3->batch()->send();

if ($responses->areOK())
{
    //...
}
```

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	Create an instance of the `AWS::S3` class by providing your AWS credentials.
2	Execute either the `AWS::S3::S3Object#copy_to` or `AWS::S3::S3Object#copy_from` method. You need to provide the request information, such as source bucket name, source key name, destination bucket name, and destination key.

The following Ruby code sample demonstrates the preceding tasks using the `#copy_to` method to copy an object from one bucket to another.

```
s3 = AWS::S3.new

# Upload a file and set server-side encryption.
bucket1 = s3.buckets[source_bucket]
bucket2 = s3.buckets[target_bucket]
obj1 = bucket1.objects[source_key]
obj2 = bucket2.objects[target_key]

obj1.copy_to(obj2)
```

Example

The following Ruby script example makes a copy of an object using the `#copy_from` method. The copied object with a different key is saved in the same source bucket. For instructions about how to create and test a working sample, see Using the AWS SDK for Ruby (p. 437).

```
#!/usr/bin/env ruby

require 'rubygems'
require 'aws-sdk'

AWS.config(
  :access_key_id => '*** Provide access key ***',
  :secret_access_key => '*** Provide secret key ***'
)

bucket_name = '*** Provide bucket name ***'
source_key = '*** Provide source key ***'
target_key = '*** Provide target key ***'

# Get an instance of the S3 interface.
s3 = AWS::S3.new

# Copy the object.
s3.buckets[bucket_name].objects[target_key].copy_from(source_key)

puts "Copying file #{source_key} to #{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+iX8JQNyaLdTshL0KxatbaOZt
x-amz-request-id: 6B13C3C5B34AF333
Date: 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

Topics

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. 208).

Copy an Object Using the AWS SDK for Java Multipart Upload API

The following task guides you through using the Java SDK to copy an Amazon S3 object from one source location to another, such as from one bucket to another. You can use the code demonstrated here to copy objects greater than 5 GB. For objects less than 5 GB, use the single operation copy described in Copy an Object Using the AWS SDK for Java (p. 209).

Copying Objects

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Initiate a multipart copy by executing the `AmazonS3Client.initiateMultipartUpload` method. Create an instance of `InitiateMultipartUploadRequest`. You will need to provide a bucket name and a key name.
3	Save the upload ID from the response object that the `AmazonS3Client.initiateMultipartUpload` method returns. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Copy all the parts. For each part copy, create a new instance of the `CopyPartRequest` class and provide part information including source bucket, destination bucket, object key, uploadID, first byte of the part, last byte of the part, and the part number.
5	Save the response of the `CopyPartRequest` method in a list. The response includes the ETag value and the part number. You will need the part number to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the `AmazonS3Client.completeMultipartUpload` method to complete the copy.

The following Java code sample demonstrates the preceding tasks.

```
// Step 1: Create instance and provide credentials.
AmazonS3Client s3Client = new AmazonS3Client(new
    PropertiesCredentials(
       LowLevel_LargeObjectCopy.class.getResourceAsStream(
         "AwsCredentials.properties")));
```

```
// Create lists to hold copy responses
List<CopyPartResult> copyResponses =
        new ArrayList<CopyPartResult>();

// Step 2: Initialize
InitiateMultipartUploadRequest initiateRequest =
       new InitiateMultipartUploadRequest(targetBucketName, targetObjectKey);

InitiateMultipartUploadResult initResult =
        s3Client.initiateMultipartUpload(initiateRequest);

// Step 3: Save upload Id.
String uploadId = initResult.getUploadId();

try {

    // Get object size.
    GetObjectMetadataRequest metadataRequest =
     new GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);

   ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRequest);

    long objectSize = metadataResult.getContentLength(); // in bytes

     // Step 4. Copy parts.
    long partSize = 5 * (long)Math.pow(2.0, 20.0); // 5 MB
    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++)
    {
        // Step 5. Save copy response.
     CopyPartRequest copyRequest = new CopyPartRequest()
           .withDestinationBucketName(targetBucketName)
           .withDestinationKey(targetObjectKey)
           .withSourceBucketName(sourceBucketName)
           .withSourceKey(sourceObjectKey)
           .withUploadId(initResult.getUploadId())
           .withFirstByte(bytePosition)
           .withLastByte(bytePosition + partSize -1 > objectSize ? objectSize
- 1 : bytePosition + partSize - 1)
           .withPartNumber(i);

        copyResponses.add(s3Client.copyPart(copyRequest));
        bytePosition += partSize;
    }
    // Step 7. Complete copy operation.
    CompleteMultipartUploadResult completeUploadResponse =
        s3Client.completeMultipartUpload(completeRequest);
} catch (Exception e) {
    System.out.println(e.getMessage());
}
```

Example

The following Java code example copies an object from one Amazon S3 bucket to another. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435).

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

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.s3.model.*;

public class LowLevel_LargeObjectCopy {

    public static void main(String[] args) throws IOException {
        String sourceBucketName = "*** Source-Bucket-Name ***";
        String targetBucketName = "*** Target-Bucket-Name ***";
        String sourceObjectKey  = "*** Source-Object-Key ***";
        String targetObjectKey  = "*** Target-Object-Key ***";
        AmazonS3Client s3Client = new AmazonS3Client(new
          PropertiesCredentials(
            LowLevel_LargeObjectCopy.class.getResourceAsStream(
              "AwsCredentials.properties")));

        // List to store copy part responses.

        List<CopyPartResult> copyResponses =
                  new ArrayList<CopyPartResult>();

        InitiateMultipartUploadRequest initiateRequest =
        new InitiateMultipartUploadRequest(targetBucketName, targetObjectKey);


        InitiateMultipartUploadResult initResult =
         s3Client.initiateMultipartUpload(initiateRequest);

        try {
            // Get object size.
            GetObjectMetadataRequest metadataRequest =
             new GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);

          ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRe
quest);
            long objectSize = metadataResult.getContentLength(); // in bytes

            // Copy parts.
            long partSize = 5 * (long)Math.pow(2.0, 20.0); // 5 MB

            long bytePosition = 0;
            for (int i = 1; bytePosition < objectSize; i++)
            {
             CopyPartRequest copyRequest = new CopyPartRequest()
                   .withDestinationBucketName(targetBucketName)
                   .withDestinationKey(targetObjectKey)
                   .withSourceBucketName(sourceBucketName)
                   .withSourceKey(sourceObjectKey)
                   .withUploadId(initResult.getUploadId())
```

```
                    .withFirstByte(bytePosition)
                    .withLastByte(bytePosition + partSize -1 > objectSize ? ob
jectSize - 1 : bytePosition + partSize - 1)
                    .withPartNumber(i);

                copyResponses.add(s3Client.copyPart(copyRequest));
                bytePosition += partSize;

            }
            CompleteMultipartUploadRequest completeRequest = new
             CompleteMultipartUploadRequest(
               targetBucketName,
               targetObjectKey,
               initResult.getUploadId(),
               GetETags(copyResponses));

            CompleteMultipartUploadResult completeUploadResponse =
                s3Client.completeMultipartUpload(completeRequest);
        } catch (Exception e) {
         System.out.println(e.getMessage());
        }
     }

    // Helper function that constructs ETags.
    static List<PartETag> GetETags(List<CopyPartResult> responses)
    {
        List<PartETag> etags = new ArrayList<PartETag>();
        for (CopyPartResult response : responses)
        {
            etags.add(new PartETag(response.getPartNumber(), re
sponse.getETag()));
        }
        return etags;
    }
}
```

Copy an Object Using the AWS SDK for .NET Multipart Upload API

The following task guides you through using the .NET SDK to copy an Amazon S3 object from one source location to another, such as from one bucket to another. You can use the code demonstrated here to copy objects that are greater than 5 GB. For objects less than 5 GB, use the single operation copy described in Copy an Object Using the AWS SDK for .NET (p. 210).

Copying Objects

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Initiate a multipart copy by executing the `AmazonS3Client.InitiateMultipartUpload` method. Create an instance of the `InitiateMultipartUploadRequest`. You will need to provide a bucket name and key name.
3	Save the upload ID from the response object that the `AmazonS3Client.InitiateMultipartUpload` method returns. You will need to provide this upload ID for each subsequent multipart upload operation.

4	Copy all the parts. For each part copy, create a new instance of the `CopyPartRequest` class and provide part information including source bucket, destination bucket, object key, uploadID, first byte of the part, last byte of the part, and the part number.
5	Save the response of the `CopyPartRequest` method in a list. The response includes the ETag value and the part number you will need to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the `AmazonS3Client.CompleteMultipartUpload` method to complete the copy.

The following C# code sample demonstrates the preceding tasks.

```
// Step 1. Create instance and provide credentials.
AmazonS3Config config = new AmazonS3Config();
AmazonS3 s3Client = new AmazonS3Client(AccessKeyID, SecretAccessKey, config);

// List to store upload part responses.
List<UploadPartResponse> uploadResponses =
    new List<UploadPartResponse>();
List<CopyPartResponse> copyResponses =
    new List<CopyPartResponse>();
InitiateMultipartUploadRequest initiateRequest =
    new InitiateMultipartUploadRequest()
        .WithBucketName(targetBucket)
        .WithKey(targetObjectKey);

// Step 2. Initialize.
InitiateMultipartUploadResponse initResponse =
s3Client.InitiateMultipartUpload(initiateRequest);

// Step 3. Save Upload Id.
String uploadId = initResponse.UploadId;

try
{
    // Get object size.
    GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest();

    metadataRequest.BucketName = sourceBucket;
    metadataRequest.Key = sourceObjectKey;

    GetObjectMetadataResponse metadataResponse = s3Client.GetObject
Metadata(metadataRequest);
    long objectSize = metadataResponse.ContentLength; // in bytes

    // Step 4. Copy parts.
    long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB
    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++)
    {
        // Step 5. Save copy response.
        CopyPartRequest copyRequest = new CopyPartRequest()
            .WithDestinationBucket(targetBucket)
            .WithDestinationKey(targetObjectKey)
            .WithSourceBucket(sourceBucket)
            .WithSourceKey(sourceObjectKey)
            .WithUploadID(uploadId)
            .WithFirstByte(bytePosition)
            .WithLastByte(bytePosition + partSize -1 > objectSize ? objectSize
 - 1 : bytePosition + partSize - 1)
            .WithPartNumber(i);

        copyResponses.Add(s3Client.CopyPart(copyRequest));
        bytePosition += partSize;

    }
     // Step 7. Complete copy operation.
     CompleteMultipartUploadRequest completeRequest =
        new CompleteMultipartUploadRequest()
            .WithBucketName(targetBucket)
            .WithKey(targetObjectKey)
            .WithUploadId(initResponse.UploadId)
            .WithPartETags(GetETags(copyResponses));

     CompleteMultipartUploadResponse completeUploadResponse =
        s3Client.CompleteMultipartUpload(completeRequest);

 }
catch (Exception e) {
    Console.WriteLine(e.Message);
}
```

Example

The following C# code example copies an object from one Amazon S3 bucket to another. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436).

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Configuration;
using System.IO;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.LowLevel_LargeObjectCopy
{
    class MPUcopyObject
    {

        static string sourceBucket = "*** Source bucket name ***";
        static string targetBucket = "*** Target bucket name ***";
        static string sourceObjectKey = "*** Source object key ***";
        static string targetObjectKey = "*** Target object key ***";

        static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKey = appConfig["AWSSecretKey"];

            AmazonS3Config config = new AmazonS3Config();
           AmazonS3 s3Client = new AmazonS3Client(accessKeyID, secretAccessKey,
 config);

            // List to store upload part responses.
            List<UploadPartResponse> uploadResponses =
                new List<UploadPartResponse>();

            List<CopyPartResponse> copyResponses =
                      new List<CopyPartResponse>();
            InitiateMultipartUploadRequest initiateRequest =
                   new InitiateMultipartUploadRequest()
                    .WithBucketName(targetBucket)
                    .WithKey(targetObjectKey);

            InitiateMultipartUploadResponse initResponse =
                s3Client.InitiateMultipartUpload(initiateRequest);
            String uploadId = initResponse.UploadId;

            try
            {
                // Get object size.
                GetObjectMetadataRequest metadataRequest = new GetObject
MetadataRequest();
                metadataRequest.BucketName = sourceBucket;
                metadataRequest.Key = sourceObjectKey;

              GetObjectMetadataResponse metadataResponse = s3Client.GetObject
Metadata(metadataRequest);
```

```
                long objectSize = metadataResponse.ContentLength; // in bytes

                // Copy parts.
                long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

                long bytePosition = 0;
                for (int i = 1; bytePosition < objectSize; i++)
                {

                    CopyPartRequest copyRequest = new CopyPartRequest()
                       .WithDestinationBucket(targetBucket)
                       .WithDestinationKey(targetObjectKey)
                       .WithSourceBucket(sourceBucket)
                       .WithSourceKey(sourceObjectKey)
                       .WithUploadID(uploadId)
                       .WithFirstByte(bytePosition)
                       .WithLastByte(bytePosition + partSize - 1 > objectSize
? objectSize - 1 : bytePosition + partSize - 1)
                       .WithPartNumber(i);

                    copyResponses.Add(s3Client.CopyPart(copyRequest));

                    bytePosition += partSize;

                }
                CompleteMultipartUploadRequest completeRequest =
                      new CompleteMultipartUploadRequest()
                         .WithBucketName(targetBucket)
                         .WithKey(targetObjectKey)
                         .WithUploadId(initResponse.UploadId)
                         .WithPartETags(GetETags(copyResponses));

                CompleteMultipartUploadResponse completeUploadResponse =
                    s3Client.CompleteMultipartUpload(completeRequest);

            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        // Helper function that constructs ETags.
        static List<PartETag> GetETags(List<CopyPartResponse> responses)
        {
            List<PartETag> etags = new List<PartETag>();
            foreach (CopyPartResponse response in responses)
            {
                etags.Add(new PartETag(response.PartNumber, response.ETag));
            }
            return etags;
        }
    }
}
```

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. 141).

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 lexicographic (alphabetical) 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.

You can iterate through large collections of keys by making multiple, paginated, list requests. For example, an initial list request against the dictionary bucket might retrieve only information about the keys "quack" through "quartermaster". But a subsequent request would retrieve "quarters" through "quince", and so on.

For instructions on how to correctly handle large list result sets, see Iterating Through Multi-Page Results (p. 227).

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 Prefix and Delimiter (p. 227).

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, Amazon S3 uses 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.

Listing Keys Hierarchically Using Prefix and Delimiter

The prefix and delimiter parameters limit the kind of results returned by a list operation. Prefix limits results to only those keys that begin with the specified prefix, and delimiter causes list to roll up all 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/California/San Francisco
- North America/USA/Washington/Seattle

and so on.

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 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.

```
<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>&quot;d1a7fb5eab1c16cb4f7cf341cf188c3d&quot;</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

The following tasks guide you through using the Java classes to list object keys in a bucket. You have many options for listing the objects in your bucket. Note that for buckets with large number of objects you might get truncated results when listing the objects. You should check to see if the returned object listing is truncated, and use the `AmazonS3.listNextBatchOfObjects` operation to retrieve additional results.

Listing Object Keys

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute one of the `AmazonS3Client.listObjects` methods. You need to provide the request information, such as the bucket name, and the optional key prefix. You provide this information by creating an instance of the `ListObjectsRequest` class. For listing objects, Amazon S3 returns up to 1,000 keys in the response. If you have more than 1,000 keys in your bucket, the response will be truncated. You should always check for if the response is truncated.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                 myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);

ListObjectsRequest listObjectsRequest = new ListObjectsRequest()
.withBucketName(bucketName)
.withPrefix("m");
```

```
ObjectListing objectListing;

do {
 objectListing = s3client.listObjects(listObjectsRequest);
 for (S3ObjectSummary objectSummary :
  objectListing.getObjectSummaries()) {
  System.out.println( " - " + objectSummary.getKey() + "  " +
                "(size = " + objectSummary.getSize() +
    ")");
 }
 listObjectsRequest.setMarker(objectListing.getNextMarker());
} while (objectListing.isTruncated());
```

Example

The following Java code example list object keys in an Amazon S3 bucket. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435)

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.ListObjectsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;

public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            System.out.println("Listing objects");

            ListObjectsRequest listObjectsRequest = new ListObjectsRequest()
                .withBucketName(bucketName)
                .withPrefix("m");
            ObjectListing objectListing;
            do {
                objectListing = s3client.listObjects(listObjectsRequest);
                for (S3ObjectSummary objectSummary :
                 objectListing.getObjectSummaries()) {
                    System.out.println(" - " + objectSummary.getKey() + "  " +

                            "(size = " + objectSummary.getSize() +
                            ")");
                }
                listObjectsRequest.setMarker(objectListing.getNextMarker());
            } while (objectListing.isTruncated());
         } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, " +
              "which means your request made it " +
                    "to Amazon S3, but was rejected with an error response " +

                    "for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, " +
              "which means the client encountered " +
                    "an internal error while trying to communicate" +
                    " with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Listing Keys Using the AWS SDK for .NET

The following tasks guide you through using the .NET classes to list object keys in a bucket.

Listing Object Keys

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute one of the `AmazonS3.ListObjects`. You need to provide the bucket name to list the keys. You can also specify optional information, such as retrieving keys starting with a specific prefix, and limiting result set to a specific number of keys. By default, the result set returns up to 1,000 keys. You provide this information by creating an instance of the `ListObjectsRequest` class.
3	Process the `ListObjectResponse` by iterating over the `ListObjectResponse.S3Objects` collection. You should check to see if the result was truncated. If yes, you send a request for the next set of keys by setting the `ListObjectRequest.Marker` value to the `NextMarker` value received in the previous response.

The following C# code sample demonstrates the preceding tasks.

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

ListObjectsRequest request = new ListObjectsRequest();
request = new ListObjectsRequest();
request.BucketName = bucketName;
request.WithPrefix("m");
request.MaxKeys = 2;
do
{
   ListObjectsResponse response = client.ListObjects(request);

   // Process response.
   // ...

   // If 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);
```

Example

The following C# code example lists keys in the specified bucket. The example illustrates the use of `AmazonS3.ListObjects` method. It also illustrates how you can specify options to list keys, such as listing keys with a specific prefix, listing keys that start after a specific marker, and listing only a specific number of keys. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.listingkeys
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                   Console.WriteLine("Listing objects stored in a bucket");
                    ListingObjects();
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void ListingObjects()
        {
            try
            {
                ListObjectsRequest request = new ListObjectsRequest();
                request = new ListObjectsRequest();
                request.BucketName = bucketName;
                request.WithPrefix("m");
                request.MaxKeys = 2;

               do
               {
                   ListObjectsResponse response = client.ListObjects(request);
                   // Process response.
                   foreach (S3Object entry in response.S3Objects)
                   {
                       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.Marker = response.NextMarker;
                   }
                   else
                   {
                       request = null;
                   }
               } while (request != null);
            }
            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);
                }
            }
        }

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }

            return true;
        }
    }
}
```

Listing Keys Using the AWS SDK for PHP

When enumerating objects in a bucket you have option of using the low-level `list_objects()` method or the high-level `get_object_list()` method.

The low-level `list_objects()` method maps to the underlying Amazon S3 REST API. Each `list_objects()` request returns you a page of up to 1,000 object keys. If you have more than 1,000 objects in the bucket, your response will be truncated. You will need to send another `list_objects()` request to fetch the next set of 1,000 keys. This method returns the entire Amazon S3 response. For example, in addition to key names, the response includes object metadata such as content size and the last modified date.

The high-level `get_object_list()` method provides higher level of abstraction which simplifies your coding efforts. For example, you don't need to worry if the response is truncated or not. The method returns all the keys from the specified bucket. Internally, this API makes all the necessary calls to ensure all the keys are retrieved. However, the API returns only object keys names in an array. You use this high-level API is to enumerate object keys from a specific bucket.

The following tasks guide you through using the PHP classes to list object keys in a bucket.

Listing Object Keys

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the high-level `AmazonS3::get_object_list()` or the low-level `AmazonS3::list_objects()` method by providing the bucket name as input.

The following PHP code sample demonstrates the preceding tasks.

```
// Instantiate the class.
$s3 = new AmazonS3();
// Use the high-level API.
$response = $s3->get_object_list($bucket);

// Or, use the low-level API().
$response = $s3->list_objects($bucket);
// Process response.
```

Example

The following PHP example lists keys from a specified bucket. The example illustrates the use of the both the high-level `get_object_list()` method and the low-level `list_objects()` method to enumerate objects in a bucket. The example also illustrates processing the response returned by these methods.

```
<?php
require_once '../aws-sdk-for-php/sdk.class.php';
header('Content-Type: text/plain; charset=utf-8');

$bucket = '*** Provide bucket name ***';


// Instantiate the class.
$s3 = new AmazonS3();

//   1. High-level API returns all keys. However only key names are returned.

$response = $s3->get_object_list($bucket);

// Success?
print_r(gettype($response) === 'array');

if($response)
{
   echo "Keys retrieved!" . PHP_EOL;

 foreach ($response as $key)
    {
         echo $key . PHP_EOL;
    }
}

/*
    2. Low-level API. Response returns up to 1,000 keys.
       However, response has more information than just key names.
*/

$response = $s3->list_objects($bucket);

foreach ($response->body->Contents as $content)
{
   echo (string) $content->Key . ' - ' .
        $s3->util->size_readable((string) $content->Size) . ' - ' .
        date('j M Y', strtotime((string) $content->LastModified)) . PHP_EOL;
}
```

Listing Keys Using the REST API

You can use the AWS SDK to retrieve an object. However, if your application requires it, you can send REST requests directly. You can send a GET request to retrieve 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).

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

Deleting Objects

Topics

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 specific a 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. 103).
- **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 an Multi Factor Authentication (MFA) enabled bucket, note the following:

- If you provide an invalid MFA token, the request always fails.
- If you have 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 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 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. 356).

Related Resources

- Using the AWS SDKs and Explorers (p. 433)

Deleting One Object Per Request

Topics

Amazon S3 provides the DELETE API (see DELETE Object) for you to delete one object per request. To learn more about object deletion, see Deleting Objects (p. 236).

You can use the REST API directly or use the wrapper libraries provided by the AWS SDKs that can simplify your application development.

Deleting an Object Using the AWS SDK for Java

The following tasks guide you through using the AWS SDK for Java classes to delete an object.

Deleting an Object (Non-Versioned Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute one of the `AmazonS3Client.deleteObject` methods. You can provide a bucket name and an object name as parameters or provide the same information in a `DeleteObjectRequest` object and pass the object as a parameter. If you have not enabled versioning on the bucket, the operation deletes the object. If you have enabled versioning, the operation adds a delete marker. For more information, see Deleting One Object Per Request (p. 237).

The following Java sample demonstrates the preceding steps. The sample uses the `DeleteObjectRequest` class to provide a bucket name and an object key.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                     myAccessKeyID, mySecretKey);
```

```
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
```

Deleting a Specific Version of an Object (Version-Enabled Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute one of the `AmazonS3Client.deleteVersion` methods. You can provide a bucket name and an object key directly as parameters or use the `DeleteVersionRequest` to provide the same information.

The following Java sample demonstrates the preceding steps. The sample uses the `DeleteVersionRequest` class to provide a bucket name, an object key, and a version Id.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                       myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.deleteObject(new DeleteVersionRequest(bucketName, keyName, versionId));
```

Example 1: Deleting an Object (Non-Versioned Bucket)

The following Java example deletes an object from a bucket. If you have not enabled versioning on the bucket, Amazon S3 deletes the object. If you have enabled versioning, Amazon S3 adds a delete marker and the object is not deleted. For information about how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

public class S3Sample  {

    private static String bucketName = "*** Provide a Bucket Name ***";
    private static String keyName = "*** Provide a Key Name ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
          S3Sample.class.getResourceAsStream("AwsCredentials.properties")));
        try {
          s3Client.deleteObject(new DeleteObjectRequest(bucketName, keyName));

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Example 2: Deleting an Object (Versioned Bucket)

The following Java example deletes a specific version of an object from a versioned bucket. The `deleteObject` request removes the specific object version from the bucket.

To test the sample, you must provide a bucket name. The code sample performs the following tasks:

1. Enable versioning on the bucket.
2. Add a sample object to the bucket. In response, Amazon S3 returns the version ID of the newly added object.
3. Delete the sample object using the `deleteVersion` method. The `DeleteVersionRequest` class specifies both an object key name and a version ID.

For information about how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteVersionRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.PutObjectResult;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class S3Sample  {

    static String bucketName = "*** Provide a Bucket Name ***";
    static String keyName = "*** Provide a Key Name ****";
    static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new PropertiesCredentials(
          S3Sample.class.getResourceAsStream("AwsCredentials.properties")));
        try {
            // Make the bucket version-enabled.
            enableVersioningOnBucket(s3Client, bucketName);

            // Add a sample object.
            String versionId = putAnObject(keyName);

            s3Client.deleteVersion(
                    new DeleteVersionRequest(
                            bucketName,
                            keyName,
                            versionId));

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
            System.out.println("Error Message: " + ace.getMessage());
        }

    }

    static void enableVersioningOnBucket(AmazonS3Client s3Client,
            String bucketName) {
        BucketVersioningConfiguration config = new BucketVersioningConfigura
tion()
                .withStatus(BucketVersioningConfiguration.ENABLED);
        SetBucketVersioningConfigurationRequest setBucketVersioningConfigura
tionRequest = new SetBucketVersioningConfigurationRequest(
                bucketName, config);
       s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigura
tionRequest);
    }

    static String putAnObject(String keyName) {
        String content = "This is the content body!";
        String key = "ObjectToDelete-" + new Random().nextInt();
        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setHeader("Subject", "Content-As-Object");
        metadata.setHeader("Content-Length", content.length());
        PutObjectRequest request = new PutObjectRequest(bucketName, key,
                new ByteArrayInputStream(content.getBytes()), metadata)
                .withCannedAcl(CannedAccessControlList.AuthenticatedRead);
        PutObjectResult response = s3Client.putObject(request);
        return response.getVersionId();
    }
}
```

Deleting an Object Using the AWS SDK for .NET

You can delete an object from a bucket. If you have versioning enabled on the bucket, you can also delete a specific version of an object.

The following tasks guide you through using the .NET classes to delete an object.

Deleting an Object (Non-Versioned Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute the `AmazonS3.DeleteObject` method by providing a bucket name and an object key in an instance of `DeleteObjectRequest`. If you have not enabled versioning on the bucket, the operation deletes the object. If you have enabled versioning, the operation adds a delete marker. For more information, see Deleting One Object Per Request (p. 237).

The following C# code sample demonstrates the preceding steps.

```
static AmazonS3Client client;
client = new AmazonS3Client;

DeleteObjectRequest deleteObjectRequest =
    new DeleteObjectRequest()
    .WithBucketName(bucketName)
    .WithKey(keyName);

using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
     accessKeyID, secretAccessKeyID))
{
     client.DeleteObject(deleteObjectRequest);
}
```

Deleting a Specific Version of an Object (Version-Enabled Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Execute the `AmazonS3.DeleteObject` method by providing a bucket name, an object key name, and object version Id in an instance of `DeleteObjectRequest`. The `DeleteObject` method deletes the specific version of the object.

The following C# code sample demonstrates the preceding steps.

```
static AmazonS3Client client;
client = new AmazonS3Client();

DeleteObjectRequest deleteObjectRequest =
    new DeleteObjectRequest()
    .WithBucketName(bucketName)
    .WithKey(keyName)
    .WithVersionId(versionId);

using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
     accessKeyID, secretAccessKeyID))
{
     client.DeleteObject(deleteObjectRequest);
}
```

Example 1: Deleting an Object (Non-Versioned Bucket)

The following C# code example deletes an object from a bucket. It does not provide a version Id in the delete request. If you have not enabled versioning on the bucket, Amazon S3 deletes the object. If you have enabled versioning, Amazon S3 adds a delete marker and the object is not deleted. For information about how to create and test a working sample, see Testing the .NET Code Examples (p. 436).

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

namespace s3.amazon.com.docsamples.deleteobject
{
  class DeleteObject
  {
    static string bucketName = "*** Provide a bucket name ***";
    static string keyName = "*** Provide a key name ****";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      DeleteObjectRequest deleteObjectRequest =
          new DeleteObjectRequest()
          .WithBucketName(bucketName)
          .WithKey(keyName);

      using (client = new AmazonS3Client())
      {
        try
        {
          client.DeleteObject(deleteObjectRequest);
          Console.WriteLine("Deleting an object");
        }
        catch (AmazonS3Exception s3Exception)
        {
          Console.WriteLine(s3Exception.Message,
                            s3Exception.InnerException);
        }
      }
      Console.WriteLine("Press any key to continue...");
      Console.ReadKey();
    }
  }
}
```

Example 2: Deleting an Object (Versioned Bucket)

The following C# code example deletes an object from a versioned bucket. The `DeleteObjectRequest` instance specifies an object key name and a version ID. The `DeleteObject` method removes the specific object version from the bucket.

To test the sample, you must provide a bucket name. The code sample performs the following tasks:

1. Enable versioning on the bucket.
2. Add a sample object to the bucket. In response, Amazon S3 returns the version ID of the newly added object.
3. Delete the sample object using the `DeleteObject` method. The `DeleteObjectRequest` class specifies both an object key name and a version ID.

For information about how to create and test a working sample, see Testing the .NET Code Examples (p. 436).

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

namespace s3.amazon.com.docsamples
{
  class DeleteObject
  {
    static string bucketName = "*** Provide a Bucket Name ***";
    static string keyName = "*** Provide a Key Name ***";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      using (client = new AmazonS3Client())
      {
        try
        {
          // Make the bucket version-enabled.
          EnableVersioningOnBucket(bucketName);

          // Add a sample object.
          string versionID = 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");
          client.DeleteObject(request);

        }
        catch (AmazonS3Exception s3Exception)
        {
          Console.WriteLine(s3Exception.Message,
                            s3Exception.InnerException);
        }
      }
      Console.WriteLine("Press any key to continue...");
      Console.ReadKey();
    }

    static void EnableVersioningOnBucket(string bucketName)
    {
     SetBucketVersioningRequest setBucketVersioningRequest = new SetBucketVer
sioningRequest
      {
        BucketName = bucketName,
        VersioningConfig = new S3BucketVersioningConfig { Status = "Enabled" }

      };
      client.SetBucketVersioning(setBucketVersioningRequest);
    }

    static string PutAnObject(string objectKey)
    {

      PutObjectRequest request = new PutObjectRequest
      {
        BucketName = bucketName,
        Key = objectKey,
        ContentBody = "This is the content body!"
      };

      PutObjectResponse response = client.PutObject(request);
      return response.VersionId;

    }
  }
}
```

Deleting an Object Using the AWS SDK for PHP

You can delete an object from a bucket. If you have versioning enabled on the bucket, you can also delete a specific version of an object.

The following tasks guide you through using the PHP classes to delete an object. For more information about versioning, see Using Versioning (p. 355).

Deleting One Object (Non-Versioned Bucket)

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the `AmazonS3::delete_object()` method. You must provide a bucket name and a key name as parameters. If you have not enabled versioning on the bucket, the operation deletes the object. If you have enabled versioning, the operation adds a delete marker. For more information, see Deleting One Object Per Request (p. 237).

The following PHP code sample demonstrates the preceding steps using the `delete_object()` method.

```
// Instantiate the class.
$s3 = new AmazonS3();
$response = $s3->delete_object($bucket, $keyname);
```

Deleting a Specific Version of an Object (Version-Enabled Bucket)

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the `AmazonS3::delete_object()` method by providing a bucket name, an object key, and a version Id as parameters. The method deletes the specific version of the object.

The following PHP code sample demonstrates the preceding steps.

```
// Instantiate the class.
$s3 = new AmazonS3();
$response = $s3->delete_object($bucket, $keyname, array (
        'versionId' =>$versionId));
```

Example 1: Deleting an Object (Non-Versioned Bucket)

The following PHP code example deletes an object from a bucket. It does not provide a version Id in the delete request. If you have not enabled versioning on the bucket, Amazon S3 deletes the object. If you have enabled versioning, Amazon S3 adds a delete marker and the object is not deleted. For information about how to create and test a working sample, see Using the AWS SDK for PHP (p. 437).

```
<?php
require_once '../sdk.class.php';

$bucket = '*** Provide a Bucket Name ***';
$keyname = '*** Provide a Key Name ***';

// Instantiate the class.
$s3 = new AmazonS3();

// Delete the object.
$response = $s3->delete_object($bucket, $keyname);

// Success?
var_dump($response->isOK());
?>
```

Example 2: Deleting an Object (Versioned Bucket)

The following PHP code example deletes an object from a versioned-bucket. The `delete_object()` method specifies an object key name and a version ID and removes the specific object version from the bucket.

To test the sample, you must provide a bucket name. The code sample performs the following tasks:

1. Enable versioning on the bucket.
2. Add a sample object to the bucket. In response, Amazon S3 returns the version ID of the newly added object.
3. Delete the sample object using the `delete_object()` method.

For information about how to create and test a working sample, see Using the AWS SDK for PHP (p. 437).

```
<?php
require_once '../sdk.class.php';
$bucket = '***Provide Bucket Name***;
$keyname = '***Provide Object Key Name****';

// Instantiate the class.
$s3 = new AmazonS3();

// Enable versioning on the bucket.
$response = $s3->enable_versioning($bucket);

// Add a sample object.
$versionId = create_object($keyname, $bucket, $s3);

// Delete the object.
$response = $s3->delete_object($bucket,$keyname, array(
        'versionId' => $versionId));

// Success?
var_dump($response->isOK());

// Create a sample object.
function create_object($keyname, $bucket, $s3)
{
 $keys = array();
 $content = "This is the content body!";
 $response = $s3->create_object($bucket, $keyname, array(
     'body' => $content,
        'acl'  => AmazonS3::ACL_AUTH_READ,
     'contentType' => 'text/plain'
     )
 );
 return $response->header['x-amz-version-id'];
}
?>
```

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

Amazon S3 provides the Multi-Object Delete API (see Delete - Multi-Object Delete) that 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 which the response includes the result each keys deletion that was encountered in your request. In quiet mode, the response includes only keys where the delete operation encountered an error.

If all keys were successfully deleted when using the quiet mode, Amazon S3 returns empty response.

To learn more about object deletion, see Deleting Objects (p. 236).

You can use the REST API directly or use the AWS SDKs.

Deleting Multiple Objects Using the AWS SDK for Java

The following tasks guide you through using the AWS SDK for Java classes to delete multiple objects in a single HTTP request.

Deleting Multiple Objects (Non-Versioned Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Create an instance of the `DeleteObjectsRequest` class and provide a list of objects keys you want to delete.
3	Execute the `AmazonS3Client.deleteObjects` method.

The following Java code sample demonstrates the preceding steps.

```
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(buck
etName);

List<KeyVersion> keys = new ArrayList<KeyVersion>();
keys.add(new KeyVersion(keyName1));
keys.add(new KeyVersion(keyName2));
keys.add(new KeyVersion(keyName3));

multiObjectDeleteRequest.setKeys(keys);

try {
    DeleteObjectsResult delObjRes = s3Client.deleteObjects(multiObjectDelete
Request);
   System.out.format("Successfully deleted all the %s items.\n", delObjRes.get
DeletedObjects().size());

} catch (MultiObjectDeleteException e) {
    // Process exception.
}
```

In the event of an exception, you can review the `MultiObjectDeleteException` to determine which objects failed to delete and why as shown in the following Java example.

```
System.out.format("%s \n", e.getMessage());
System.out.format("No. of objects successfully deleted = %s\n", e.getDeletedOb
jects().size());
System.out.format("No. of objects failed to delete = %s\n", e.ge
tErrors().size());
System.out.format("Printing error data...\n");
for (DeleteError deleteError : e.getErrors()){
    System.out.format("Object Key: %s\t%s\t%s\n",
            deleteError.getKey(), deleteError.getCode(), deleteError.getMes
sage());
}
```

The following tasks guide you through deleting objects from a version-enabled bucket.

Deleting Multiple Objects (Version-Enabled Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Create an instance of the `DeleteObjectsRequest` class and provide a list of objects keys and optionally the version IDs of the objects that you want to delete. If you specify the version ID of the object that you want to delete, Amazon S3 deletes the specific object version. If you don't specify the version ID of the object that you want to delete, Amazon S3 adds a delete marker. For more information, see Deleting One Object Per Request (p. 237).
3	Execute the `AmazonS3Client.deleteObjects` method.

The following Java code sample demonstrates the preceding steps.

```
List<KeyVersion> keys = new ArrayList<KeyVersion>();
// Provide a list of object keys and versions.

DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(buck
etName)
.withKeys(keys);

try {
    DeleteObjectsResult delObjRes = s3Client.deleteObjects(multiObjectDelete
Request);
   System.out.format("Successfully deleted all the %s items.\n", delObjRes.get
DeletedObjects().size());

} catch (MultiObjectDeleteException e) {
    // Process exception.
}
```

Example 1: Multi-Object Delete (Non-Versioned Bucket)

The following Java code example uses the Multi-Object Delete API to delete objects from a non-versioned bucket. The example first uploads the sample objects to the bucket and then uses the `deleteObjects` method to delete the objects in a single request.

For information about how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.MultiObjectDeleteException.DeleteError;
import com.amazonaws.services.s3.model.MultiObjectDeleteException;
import com.amazonaws.services.s3.model.PutObjectResult;

public class S3Sample {

    static String bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {

        try {
            s3Client = new AmazonS3Client(new PropertiesCredentials(
                    S3Sample.class
                    .getResourceAsStream("AwsCredentials.properties")));
           // Upload sample objects.Because the bucket is not version-enabled,

            // the KeyVersions list returned will have null values for version
 IDs.
            List<KeyVersion> keysAndVersions1 = putObjects(3);

            // Delete specific object versions.
            multiObjectNonVersionedDelete(keysAndVersions1);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

    static List<KeyVersion> putObjects(int number) {
        List<KeyVersion> keys = new ArrayList<KeyVersion>();
        String content = "This is the content body!";
        for (int i = 0; i < number; i++) {
            String key = "ObjectToDelete-" + new Random().nextInt();
            ObjectMetadata metadata = new ObjectMetadata();
            metadata.setHeader("Subject", "Content-As-Object");
            metadata.setHeader("Content-Length", content.length());
            PutObjectRequest request = new PutObjectRequest(bucketName, key,
                    new ByteArrayInputStream(content.getBytes()), metadata)
                    .withCannedAcl(CannedAccessControlList.AuthenticatedRead);

            PutObjectResult response = s3Client.putObject(request);
           KeyVersion keyVersion = new KeyVersion(key, response.getVersionId());

            keys.add(keyVersion);
        }
        return keys;
    }

    static void multiObjectNonVersionedDelete(List<KeyVersion> keys) {

        // Multi-object delete by specifying only keys (no version ID).
       DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(

                bucketName).withQuiet(false);

        // Create request that include only object key names.
        List<KeyVersion> justKeys = new ArrayList<KeyVersion>();
        for (KeyVersion key : keys) {
            justKeys.add(new KeyVersion(key.getKey()));
        }
        multiObjectDeleteRequest.setKeys(justKeys);
        // Execute DeleteObjects - Amazon S3 add delete marker for each object

        // deletion. The objects no disappear from your bucket (verify).
        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch (MultiObjectDeleteException mode) {
            printDeleteResults(mode);
        }
    }
    static void printDeleteResults(MultiObjectDeleteException mode) {
        System.out.format("%s \n", mode.getMessage());
        System.out.format("No. of objects successfully deleted = %s\n",
mode.getDeletedObjects().size());
        System.out.format("No. of objects failed to delete = %s\n", mode.ge
tErrors().size());
        System.out.format("Printing error data...\n");
        for (DeleteError deleteError : mode.getErrors()){
            System.out.format("Object Key: %s\t%s\t%s\n",
                 deleteError.getKey(), deleteError.getCode(), deleteError.get
Message());
        }
    }
}
```

Example 2: Multi-Object Delete (Version-Enabled Bucket)

The following Java code example uses the Multi-Object Delete API to delete objects from a version-enabled bucket.

Before you can test the sample, you must create a sample bucket and provide the bucket name in the example. You can use the AWS Management Console to create a bucket.

The example performs the following actions:

1. Enable versioning on the bucket.
2. Perform a versioned-delete.
   The example first uploads the sample objects. In response, Amazon S3 returns the version IDs for each sample object that you uploaded. The example then deletes these objects using the Multi-Object Delete API. In the request, it specifies both the object keys and the version IDs (that is, versioned delete).
3. Perform a non-versioned delete.

   The example uploads the new sample objects. Then, it deletes the objects using the Multi-Object API. However, in the request, it specifies only the object keys. In this case, Amazon S3 adds the delete markers and the objects disappear from your bucket.
4. Delete the delete markers.

   To illustrate how the delete markers work, the sample deletes the delete markers. In the Multi-Object Delete request, it specifies the object keys and the version IDs of the delete markers it received in the response in the preceding step. This action makes the objects reappear in your bucket.

For information about how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult.DeletedObject;
import com.amazonaws.services.s3.model.MultiObjectDeleteException.DeleteError;
import com.amazonaws.services.s3.model.MultiObjectDeleteException;
import com.amazonaws.services.s3.model.PutObjectResult;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class S3Sample {

    static String bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {

        try {
            s3Client = new AmazonS3Client(new PropertiesCredentials(
                    S3Sample.class
                   .getResourceAsStream("AwsCredentials.properties")));

            // 1. Enable versioning on the bucket.
            enableVersioningOnBucket(s3Client, bucketName);

            // 2a. Upload sample objects.
            List<KeyVersion> keysAndVersions1 = putObjects(3);
            // 2b. Delete specific object versions.
            multiObjectVersionedDelete(keysAndVersions1);

            // 3a. Upload samples objects.
            List<KeyVersion> keysAndVersions2 = putObjects(3);
            // 3b. Delete objects using only keys. Amazon S3 creates a delete
marker and
            // returns its version Id in the response.
            DeleteObjectsResult response = multiObjectNonVersionedDelete(key
sAndVersions2);
            // 3c. Additional exercise - using multi-object versioned delete,
remove the
            // delete markers received in the preceding response. This results
 in your objects
            // reappear in your bucket
            multiObjectVersionedDeleteRemoveDeleteMarkers(response);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

    static void enableVersioningOnBucket(AmazonS3Client s3Client,
            String bucketName) {
        BucketVersioningConfiguration config = new BucketVersioningConfigura
tion()
                .withStatus(BucketVersioningConfiguration.ENABLED);
        SetBucketVersioningConfigurationRequest setBucketVersioningConfigura
tionRequest = new SetBucketVersioningConfigurationRequest(
                bucketName, config);
       s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigura
tionRequest);
    }

    static List<KeyVersion> putObjects(int number) {
        List<KeyVersion> keys = new ArrayList<KeyVersion>();
        String content = "This is the content body!";
        for (int i = 0; i < number; i++) {
            String key = "ObjectToDelete-" + new Random().nextInt();
            ObjectMetadata metadata = new ObjectMetadata();
            metadata.setHeader("Subject", "Content-As-Object");
            metadata.setHeader("Content-Length", content.length());
            PutObjectRequest request = new PutObjectRequest(bucketName, key,
                    new ByteArrayInputStream(content.getBytes()), metadata)
                    .withCannedAcl(CannedAccessControlList.AuthenticatedRead);

            PutObjectResult response = s3Client.putObject(request);
           KeyVersion keyVersion = new KeyVersion(key, response.getVersionId());

            keys.add(keyVersion);
        }
        return keys;
    }

    static void multiObjectVersionedDelete(List<KeyVersion> keys) {
       DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(

                bucketName).withKeys(keys);

        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch(MultiObjectDeleteException mode) {
            printDeleteResults(mode);
        }
    }

    static DeleteObjectsResult multiObjectNonVersionedDelete(List<KeyVersion>
keys) {

        // Multi-object delete by specifying only keys (no version ID).
       DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(

                bucketName);

        // Create request that include only object key names.
        List<KeyVersion> justKeys = new ArrayList<KeyVersion>();
        for (KeyVersion key : keys) {
            justKeys.add(new KeyVersion(key.getKey()));
        }

        multiObjectDeleteRequest.setKeys(justKeys);
        // Execute DeleteObjects - Amazon S3 add delete marker for each object

        // deletion. The objects no disappear from your bucket (verify).
        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch (MultiObjectDeleteException mode) {
            printDeleteResults(mode);
        }
        return delObjRes;
    }
    static void multiObjectVersionedDeleteRemoveDeleteMarkers(
            DeleteObjectsResult response) {

        List<KeyVersion> keyVersionList = new ArrayList<KeyVersion>();
        for (DeletedObject deletedObject : response.getDeletedObjects()) {
            keyVersionList.add(new KeyVersion(deletedObject.getKey(),
                    deletedObject.getDeleteMarkerVersionId()));
        }
        // Create a request to delete the delete markers.
        DeleteObjectsRequest multiObjectDeleteRequest2 = new DeleteObjects
Request(
                bucketName).withKeys(keyVersionList);

        // Now delete the delete marker bringing your objects back to the
bucket.
        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest2);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch (MultiObjectDeleteException mode) {
            printDeleteResults(mode);
        }
    }
    static void printDeleteResults(MultiObjectDeleteException mode) {
        System.out.format("%s \n", mode.getMessage());
        System.out.format("No. of objects successfully deleted = %s\n",
mode.getDeletedObjects().size());
        System.out.format("No. of objects failed to delete = %s\n", mode.ge
tErrors().size());
        System.out.format("Printing error data...\n");
        for (DeleteError deleteError : mode.getErrors()){
            System.out.format("Object Key: %s\t%s\t%s\n",
                 deleteError.getKey(), deleteError.getCode(), deleteError.get
Message());
        }
    }
}
```

Deleting Multiple Objects Using the AWS SDK for .NET

The following tasks guide you through using the AWS SDK for .NET classes to delete multiple objects in a single HTTP request.

Deleting Multiple Objects (Non-Versioned Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Create an instance of the `DeleteObjectsRequest` class and provide list of the object keys you want to delete.
3	Execute the `AmazonS3Client.DeleteObjects` method. If one or more objects fail to delete, Amazon S3 throws a `DeleteObjectsException`.

The following C# code sample demonstrates the preceding steps.

```
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest();
multiObjectDeleteRequest.BucketName = bucketName;

multiObjectDeleteRequest.AddKey(*** Object1 Key ***);
multiObjectDeleteRequest.AddKey(*** Object2 Key ***);
multiObjectDeleteRequest.AddKey(*** Object3 Key ***);

try
{
  DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
  Console.WriteLine("Successfully deleted all the {0} items", response.Delete
dObjects.Count);
}
catch (DeleteObjectsException e)
{
  // Process exception.
}
```

In the event of an exception, you can review the `DeleteObjectsException` to determine which objects failed to delete and why as shown in the following C# code example.

```
var errorResponse = e.ErrorResponse;
Console.WriteLine("No. of objects successfully deleted = {0}", errorResponse.De
letedObjects.Count);
Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.Delet
eErrors.Count);
Console.WriteLine("Printing error data...");
foreach (DeleteError deleteError in errorResponse.DeleteErrors)
{
   Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key, deleteEr
ror.Code, deleteError.Message);
}
```

The following tasks guide you through deleting objects from a version-enabled bucket.

Deleting Multiple Objects (Version-Enabled Bucket)

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Create an instance of the `DeleteObjectsRequest` class and provide a list of object keys and optionally the version IDs of the objects that you want to delete. If you specify the version ID of the object you want to delete, Amazon S3 deletes the specific object version. If you don't specify the version ID of the object that you want to delete, Amazon S3 adds a delete marker. For more information, see Deleting One Object Per Request (p. 237).
3	Execute the `AmazonS3Client.DeleteObjects` method.

The following C# code sample demonstrates the preceding steps.

```
List<KeyVersion> keysAndVersions = new List<KeyVersion>();
// provide a list of object keys and versions.

DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
  {
    BucketName = bucketName,
    Keys = keysAndVersions
  };

try
{
  DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
  Console.WriteLine("Successfully deleted all the {0} items", response.Delete
dObjects.Count);
}
catch (DeleteObjectsException e)
{
  // Process exception.
}
```

Example 1: Multi-Object Delete (Non-Versioned Bucket)

The following C# code example uses the Multi-Object API to delete objects from a bucket that is not version-enabled. The example first 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 how to create and test a working sample, see Testing the .NET Code Examples (p. 436).

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
  class S3Sample
  {
    static string bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      using (client = new AmazonS3Client())
      {
        var keysAndVersions = PutObjects(3);
        // Delete the objects.
        MultiObjectNonVersionedDelete(keysAndVersions);
      }

      Console.WriteLine("Click ENTER to continue.....");
      Console.ReadLine();
    }

    static void MultiObjectNonVersionedDelete(List<KeyVersion> keys)
    {
      // a. multi-object delete by specifying the key names and version IDs.
      DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest

      {
        BucketName = bucketName,
        Keys = keys // This includes the object keys and null version IDs.
      };

      try
      {
       DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
       Console.WriteLine("Successfully deleted all the {0} items", response.De
letedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
      }
    }
    private static void PrintDeletionReport(DeleteObjectsException e)
    {
      var errorResponse = e.ErrorResponse;
      Console.WriteLine("No. of objects successfully deleted = {0}", errorRes
ponse.DeletedObjects.Count);
     Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.De
leteErrors.Count);
      Console.WriteLine("Printing error data...");
      foreach (DeleteError deleteError in errorResponse.DeleteErrors)
      {
        Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key, delet
eError.Code, deleteError.Message);
      }
    }

    static List<KeyVersion> PutObjects(int number)
    {
      List<KeyVersion> keys =
                   new List<KeyVersion>();

      for (int 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!",

        };

        PutObjectResponse response = client.PutObject(request);
        KeyVersion keyVersion = new KeyVersion(key, response.VersionId);

        keys.Add(keyVersion);
      }
      return keys;
    }
  }
}
```

Example 2: Multi-Object Delete (Version-Enabled Bucket)

The following C# code example uses the Multi-Object API to delete objects from a version-enabled bucket. In addition to showing the DeleteObjects Multi-Object Delete API usage, it also illustrates how versioning works in a version-enabled bucket.

Before you can test the sample, you must create a sample bucket and provide the bucket name in the example. You can use the AWS Management Console to create a bucket.

The example performs the following actions:

1. Enable versioning on the bucket.
2. Perform a versioned-delete.
   The example first uploads the sample objects. In response, Amazon S3 returns the version IDs for each sample object that you uploaded. The example then deletes these objects using the Multi-Object Delete API. In the request, it specifies both the object keys and the version IDs (that is, versioned delete).
3. Perform a non-versioned delete.

   The example uploads the new sample objects. Then, it deletes the objects using the Multi-Object API. However, in the request, it specifies only the object keys. In this case, Amazon S3 adds the delete markers and the objects disappear from your bucket.
4. Delete the delete markers.

   To illustrate how the delete markers work, the sample deletes the delete markers. In the Multi-Object Delete request, it specifies the object keys and the version IDs of the delete markers it received in the response in the preceding step. This action makes the objects reappear in your bucket.

For information about how to create and test a working sample, see Testing the .NET Code Examples (p. 436).

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
  class S3Sample
  {
    static string bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      using (client = new AmazonS3Client())
      {

        // 1. Enable versioning on the bucket.
        EnableVersioningOnBucket(bucketName);

        // 2a. Upload the sample objects.
        var keysAndVersions1 = PutObjects(3);
        // 2b. Delete the specific object versions.
        VersionedDelete(keysAndVersions1);
        // 3a. Upload the sample objects.
        var keysAndVersions2 = PutObjects(3);

        // 3b. Delete objects using only keys. Amazon S3 creates a delete
marker and
        // returns its version Id in the response.
        List<DeletedObject> deletedObjects = NonVersionedDelete(keysAndVer
sions2);

        // 3c. Additional exercise - using a multi-object versioned delete,
remove the
        // delete markers received in the preceding response. This results in
your objects
        // reappearing in your bucket.
        RemoveMarkers(deletedObjects);
      }

      Console.WriteLine("Click ENTER to continue.....");
      Console.ReadLine();
    }

    private static void PrintDeletionReport(DeleteObjectsException e)
    {
      var errorResponse = e.ErrorResponse;
      Console.WriteLine("No. of objects successfully deleted = {0}", errorRes
ponse.DeletedObjects.Count);
     Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.De
leteErrors.Count);
      Console.WriteLine("Printing error data...");
      foreach (DeleteError deleteError in errorResponse.DeleteErrors)
      {
        Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key, delet
eError.Code, deleteError.Message);
      }
    }


    static void EnableVersioningOnBucket(string bucketName)
    {
     SetBucketVersioningRequest setBucketVersioningRequest = new SetBucketVer
sioningRequest
      {
        BucketName = bucketName,
        VersioningConfig = new S3BucketVersioningConfig { Status = "Enabled" }

      };
      client.SetBucketVersioning(setBucketVersioningRequest);
    }

    static void VersionedDelete(List<KeyVersion> keys)
    {
     // a. Perform a multi-object delete by specifying the key names and version
 IDs.
      DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest

        {
          BucketName = bucketName,
          Keys = keys // This includes the object keys and specific version
```

```
IDs.
        };
      try
      {
        Console.WriteLine("Executing VersionedDelete...");
       DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
       Console.WriteLine("Successfully deleted all the {0} items", response.De
letedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
      }
    }

    static List<DeletedObject> NonVersionedDelete(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 = client.DeleteObjects(multiObjectDeleteRequest);
       Console.WriteLine("Successfully deleted all the {0} items", response.De
letedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
        throw; // Some deletes failed. Investigate before continuing.
      }

     return response.DeletedObjects; // This reponse contains the DeltedObjects
 list which we use to delete the delete markers.
    }

    private static void RemoveMarkers(List<DeletedObject> deletedObjects)
    {
      List<KeyVersion> keyVersionList = new List<KeyVersion>();

      foreach (var deletedObject in deletedObjects)
      {
        KeyVersion keyVersion = new KeyVersion(deletedObject.Key, deletedOb
ject.DeleteMarkerVersionId);
        keyVersionList.Add(keyVersion);
      }
      // Create another request to delete the delete markers.
      var multiObjectDeleteRequest = new DeleteObjectsRequest
      {
        BucketName = bucketName,
        Keys = keyVersionList
      };

     // Now, delete the delete marker to bring your objects back to the bucket.


      try
      {
        Console.WriteLine("Removing the delete markers .....");
        var deleteObjectResponse = client.DeleteObjects(multiObjectDelete
Request);
        Console.WriteLine("Successfully deleted all the {0} delete markers",
deleteObjectResponse.DeletedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
      }
    }

    static List<KeyVersion> PutObjects(int number)
    {
      List<KeyVersion> keys =
                   new List<KeyVersion>();

      for (int 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!",

        };

        PutObjectResponse response = client.PutObject(request);
        KeyVersion keyVersion = new KeyVersion(key, response.VersionId);

        keys.Add(keyVersion);
      }
      return keys;
    }
  }
}
```

Deleting Multiple Objects Using the AWS SDK for PHP

The following tasks guide you through using the PHP classes to delete multiple objects. For more information about versioning, see Using Versioning (p. 355).

Deleting Multiple Objects (Non-Versioned Bucket)

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the `AmazonS3::delete_objects()` method. You need to provide a bucket name and an array of object keys as parameters.

The following PHP code sample demonstrates the preceding steps.

```
// Instantiate the class.
$s3 = new AmazonS3();
$response = $s3->delete_objects ($bucket, array(
        'objects' => array(
                array('key' => $keyname1),
                array('key' => $keyname2)
        )
));
```

Amazon S3 returns a response that shows the objects that were deleted and objects it could not delete because of errors (for example, permission errors). The following PHP code sample prints the object keys for objects that were deleted. It also prints the object keys that were not deleted and related error messages.

```
header('Content-Type: text/plain; charset=utf-8');
// Process response.
echo 'Printing object keys that were deleted successfully:' . PHP_EOL;
foreach ($response->body->Deleted as $item ) {

    echo('Key = ' . (string) $item->Key . ', ' .
         'VersionId = ' . (string)$item->VersionId .
         PHP_EOL);
}
echo PHP_EOL;

echo 'The following objects could not be deleted:' . PHP_EOL;
foreach ($response->body->Error as $item) {
    echo 'Key= ' . (string)$item->Key . ', ' .
         'VersionId = ' . (string)$item->VersionId .
         PHP_EOL;
    echo 'Code = ' . (string)$item->Code . ', ' .
      'Message = ' . (string)$item->Message .
         PHP_EOL;
}
```

The following tasks guide you through deleting objects from a version-enabled bucket.

Deleting Multiple Objects (Version-Enabled Bucket)

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Execute the `AmazonS3::delete_objects()` method and provide a list of objects keys and optionally the version IDs of the objects that you want to delete. If you specify version ID of the object that you want to delete, Amazon S3 deletes the specific object version. If you don't specify the version ID of the object that you want to delete, Amazon S3 adds a delete marker. For more information, see Deleting One Object Per Request (p. 237).

The following PHP code sample demonstrates the preceding steps.

```php
// Instantiate the class.
$s3 = new AmazonS3();
$response = $s3->delete_objects
    ($bucket,
    array('objects' => array(
            array('key' => $keyname1, 'version_id' => $versionId1),
            array('key' => $keyname2, 'version_id' => $versionId2)
        )
));
```

Example 1: Multi-Object Delete (Non-Versioned Bucket)

The following PHP code example uses the Multi-Object API to delete objects from a bucket that is not version-enabled. The example first uploads the sample objects to the bucket and then uses the `delete_objects` method to delete the objects in a single request. In the `delete_objects` method, the example specifies only the object key names because the version IDs are null.

For information about how to create and test a working sample, see Using the AWS SDK for PHP (p. 437).

```
<?php
require_once '../sdk.class.php';
$bucket = '***Provide Bucket Name***;

// Instantiate the class.
$s3 = new AmazonS3();

// Upload the sample objects.
$keys = create_objects(3, $bucket, $s3);

// Delete specific object versions.
$response = $s3->delete_objects($bucket, array(
        'objects' => $keys,
  'quiet' => 'false'
));

// Success?
var_dump($response->isOK());

// Create the sample objects.
function create_objects($number, $bucket, $s3)
{
 $keys = array();
 $content = "This is the content body!";

 for ($i=1; $i < $number + 1; $i++){

  $keyName = "ObjectToDelete-" . mt_rand();

  $response = $s3->create_object($bucket, $keyName, array(
          'body' => $content,
          'acl'  => AmazonS3::ACL_AUTH_READ,
          'contentType' => 'text/plain',
      ));
  $keys[$i] = array('key' => $keyName);
 }
 return $keys;
}
?>
```

Example 2: Multi-Object Delete (Version-Enabled Bucket)

The following PHP code example uses the Multi-Object API to delete objects from a version-enabled bucket. In addition to showing the `delete_objects` Multi-Object Delete API usage, it also illustrates how versioning works in a version-enabled bucket.

Before you can test the sample, you must create a sample bucket and provide the bucket name in the example. You can use the AWS Management Console to create a bucket.

The example performs the following actions:

1. Enable versioning on the bucket.
2. Perform a versioned-delete.
   The example first uploads the sample objects. In response, Amazon S3 returns the version IDs for each sample object that you uploaded. The example then deletes these objects using the Multi-Object Delete API. In the request, it specifies both the object keys and the version IDs (that is, versioned delete).
3. Perform a non-versioned delete.

   The example uploads the new sample objects. Then, it deletes the objects using the Multi-Object API. However, in the request, it specifies only the object keys. In this case, Amazon S3 adds the delete markers and the objects disappear from your bucket.
4. Delete the delete markers.

   To illustrate how the delete markers work, the sample deletes the delete markers. In the Multi-Object Delete request, it specifies the object keys and the version IDs of the delete markers it received in the response in the preceding step. This action makes the objects reappear in your bucket.

For information about how to create and test a working sample, see Using the AWS SDK for PHP (p. 437).

```
<?php
require_once '../sdk.class.php';
$bucket = '***Provide Bucket Name***;

// Instantiate the class.
$s3 = new AmazonS3();

// 1. Enable versioning on the bucket.
$response = $s3->enable_versioning($bucket);

// 2a. Upload the sample objects.
$keys = create_objects(3, $bucket, $s3);

// 2b. Delete specific object versions.
$response = $s3->delete_objects($bucket, array(
        'objects' => $keys,
  'quiet' => 'false'
));

// 3a. Upload the sample objects.
$keys = create_objects(3, $bucket, $s3);

// 3b. Delete the objects using only keys. Amazon S3 creates a delete marker
and
// returns its version Id in the response.
$justKeys = array();
```

```
for ($i=1; $i < count($keys)+1; $i++) {
 $justKeys[$i] = array('key' => $keys[$i]['key']);
}
$response = $s3->delete_objects($bucket, array(
        'objects' => $justKeys,
  'quiet' => 'false'
));

// Success?
var_dump($response->isOK());

// 3c. Additional exercise - using a multi-object versioned delete, remove the

// delete markers received in the preceding response. This results in your ob
jects
// reappearing in your bucket.
$keysAndVersions = array();
$i = 1;
foreach ($response->body->Deleted as $item ) {
 $keysAndVersions[$i] = array(
 'key' => $item->Key,
 'version_id' => $item->VersionId);
}
// Create a request to delete the delete markers.
// Deleting the delete markers brings the objects back to the bucket.
$response = $s3->delete_objects ($bucket, array(
        'objects' => $keysAndVersions,
  'quiet' => 'false'
));

// Create the sample objects.
function create_objects($number, $bucket, $s3)
{
 $keys = array();
 $content = "This is the content body!";

 for ($i=1; $i < $number + 1; $i++){

  $keyName = "ObjectToDelete-" . mt_rand();

  $response = $s3->create_object($bucket, $keyName, array(
          'body' => $content,
          'acl'  => AmazonS3::ACL_AUTH_READ,
          'contentType' => 'text/plain',
      ));
  $keys[] = array('key' => $keyName);
 }
 return $keys;
}
?>
```

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*.

Restoring Objects

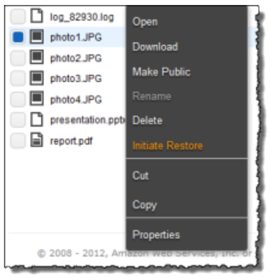
Topics

You must initiate a restore request before you can access an archived object. This section describes how to initiate a restore request programmatically. For more information about archiving objects, see Object Lifecycle Management (p. 106).

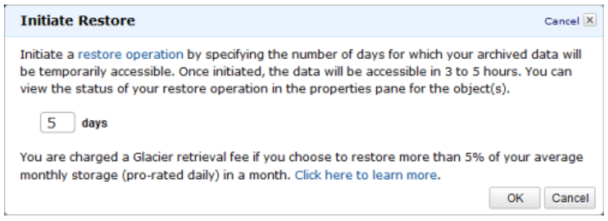
It takes about four hours for a restore operation to make a copy of the archive available for you to access. After initiating a restore, you can check status of the restoration in the console (see Restoring Glacier Objects by Using Amazon S3 Console (p. 113)) or send HEAD request to programmatically retrieve object's metadata which includes the restoration status information.

Restore an Object Using Amazon S3 Console

You can use Amazon S3 console to restore a copy of an archived object. In the console you right-mouse click on the object and select **Initiate Restore**.



You specify the number of days you want the object copy restored.


Initiate Restore Cancel

Initiate a restore operation by specifying the number of days for which your archived data will be temporarily accessible. Once initiated, the data will be accessible in 3 to 5 hours. You can view the status of your restore operation in the properties pane for the object(s).

5 days

You are charged a Glacier retrieval fee if you choose to restore more than 5% of your average monthly storage (pro-rated daily) in a month. Click here to learn more.

OK Cancel

It takes about three to five hours for Amazon S3 to complete the restoration. The object properties in the console shows the object restoration status.

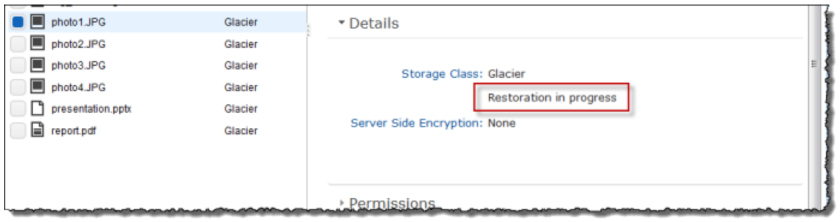


photo1.JPG	Glacier
photo2.JPG	Glacier
photo3.JPG	Glacier
photo4.JPG	Glacier
presentation.pptx	Glacier
report.pdf	Glacier

Details

Storage Class: Glacier
Restoration in progress
Server Side Encryption: None

Permissions

When object copy is restored, the object properties in the console shows the object is restored and when Amazon S3 will remove the restored copy. The console also gives you option to modify the restoration period.

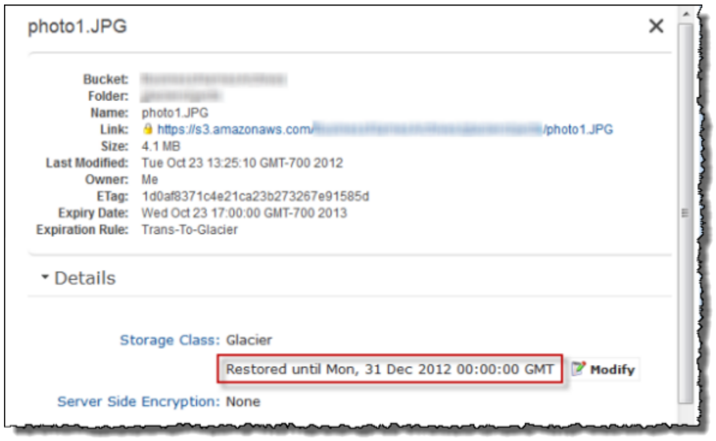


photo1.JPG

Bucket: [redacted]
Folder: [redacted]
Name: photo1.JPG
Link: https://s3.amazonaws.com/[redacted]/photo1.JPG
Size: 4.1 MB
Last Modified: Tue Oct 23 13:25:10 GMT-700 2012
Owner: Me
ETag: 1d0af8371c4e21ca23b273267e91585d
Expiry Date: Wed Oct 23 17:00:00 GMT-700 2013
Expiration Rule: Trans-To-Glacier

Details

Storage Class: Glacier
Restored until Mon, 31 Dec 2012 00:00:00 GMT
Server Side Encryption: None

Note that when you restore an archive you are paying for both the archive and a copy you restored temporarily. For information about pricing, go to the Pricing section of the *Amazon S3 product detail page*.

Amazon S3 restores a temporary copy of the object only for the specified duration. After that Amazon S3 deletes the restored object copy. You can modify the expiration period of a restored copy, by reissuing a restore, in which case Amazon S3 updates the expiration period, relative to the current time.

Amazon S3 calculates expiration time of the restored object copy by adding the number of days specified in the restoration request to the current time and rounding the resulting time to the next day midnight UTC. For example, if an object was created on 10/15/2012 10:30 am UTC and the restoration period was specified as 3 days, then the restored copy expires on 10/19/2012 00:00 UTC at which time Amazon S3 delete the object copy.

You can restore an object copy for any number of days. However you should restore objects only for the duration you need because of the storage costs associated with the object copy. For pricing information, go to the Pricing section of the Amazon S3 product detail page.

Restore an Object Using the AWS SDK for Java

The following tasks guide you through use the AWS SDK for Java to initiate a restoration of an archived object.

Downloading Objects

1	Create an instance of the `AmazonS3Client` class by providing your AWS credentials.
2	Create an instance of `RestoreObjectRequest` class by providing bucket name, object key to restore and the number of days for which you the object copy restored.
3	Execute one of the `AmazonS3.RestoreObject` methods to initiate the archive restoration.

The following Java code sample demonstrates the preceding tasks.

```
String bucketName = "examplebucket";
String objectkey = "examplekey";
```

```
AmazonS3Client s3Client = new AmazonS3Client();

RestoreObjectRequest request = new RestoreObjectRequest(bucketName, objectkey,
 2);
s3Client.restoreObject(request);
```

Amazon S3 maintains the restoration status in the object metadata. You can retrieve object metadata and check the value of the `RestoreInProgress` property as shown in the following Java code snippet.

```
String bucketName = "examplebucket";
String objectkey = "examplekey";
AmazonS3Client s3Client = new AmazonS3Client();

client = new AmazonS3Client();

GetObjectMetadataRequest request = new GetObjectMetadataRequest(bucketName,
objectKey);

ObjectMetadata response = s3Client.getObjectMetadata(request);

Boolean restoreFlag = response.getOngoingRestore();
System.out.format("Restoration status: %s.\n",
          (restoreFlag == true) ? "in progress" : "finished");
```

Example

The following Java code example initiates a restoration request for the specified archived object. You must update the code and provide a bucket name and an archived object key name. For instructions on how to create and test a working sample, see Testing the Java Code Examples (p. 435).

```
import java.io.IOException;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.GetObjectMetadataRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.RestoreObjectRequest;

public class S3LifecycleRestoreExample {

    public static String bucketName = "*** Provide bucket name ***";
    public static String objectKey = "*** Provide object key name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
       AmazonS3Client s3Client = new AmazonS3Client(new PropertiesCredentials(

                S3LifecycleRestoreExample.class.getResourceAsStream(
                        "AwsCredentials.properties")));

        try {

          RestoreObjectRequest requestRestore = new RestoreObjectRequest(buck
etName, objectKey, 2);
          s3Client.restoreObject(requestRestore);

          GetObjectMetadataRequest requestCheck = new GetObjectMetadataRe
quest(bucketName, objectKey);
          ObjectMetadata response = s3Client.getObjectMetadata(requestCheck);

          Boolean restoreFlag = response.getOngoingRestore();
          System.out.format("Restoration status: %s.\n",
                  (restoreFlag == true) ? "in progress" : "finished");

        } 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());
        }
    }
}
```

Restore an Object Using the AWS SDK for .NET

The following tasks guide you through using the AWS SDK for .NET to initiate a restoration of an archived object.

Downloading Objects

1	Create an instance of the `AmazonS3` class by providing your AWS credentials.
2	Create an instance of `RestoreObjectRequest` class by providing bucket name, object key to restore and the number of days for which you the object copy restored.
3	Execute one of the `AmazonS3.RestoreObject` methods to initiate the archive restoration.

The following C# code sample demonstrates the preceding tasks.

```
AmazonS3 client;
string bucketName = "examplebucket";
string objectKey = "examplekey";

client = new AmazonS3Client();

RestoreObjectRequest restoreRequest = new RestoreObjectRequest()
 {
     BucketName = bucketName,
     Key = objectKey,
     Days = 2
 };

client.RestoreObject(restoreRequest);
```

Amazon S3 maintains the restoration status in the object metadata. You can retrieve object metadata and check the value of the `RestoreInProgress` property as shown in the following C# code snippet.

```
AmazonS3 client;
string bucketName = "examplebucket";
string objectKey = "examplekey";

client = new AmazonS3Client();

GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest()
{
      BucketName = bucketName,
      Key = objectKey
};
GetObjectMetadataResponse response = client.GetObjectMetadata(metadataRequest);
Console.WriteLine("Restoration status: {0}", response.RestoreInProgress);
if (response.RestoreInProgress == false)
   Console.WriteLine("Restored object copy expires on: {0}", response.Restore
Expiration);
```

Example

The following C# code example initiates a restoration request for the specified archived object. You must update the code and provide a bucket name and an archived object key name. For instructions on how to create and test a working sample, see Testing the .NET Code Examples (p. 436)

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

namespace aws.amazon.com.s3.documentation
{
    class S3Sample
    {
        static string bucketName = "*** provide bucket name ***";
        static string objectKey = "*** archived object keyname ***";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                using (client = new AmazonS3Client())
                {
                    RestoreArchivedObject(client, bucketName, objectKey);
                    CheckRestorationStatus(client, bucketName, objectKey);
                }

                Console.WriteLine("Example complete. To continue, click 
Enter...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " + 
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }

        static void RestoreArchivedObject(AmazonS3 client, string bucketName, 
string objectKey)
        {
            RestoreObjectRequest restoreRequest = new RestoreObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Days = 2
            };
            RestoreObjectResponse response = client.RestoreObject(restore
Request);
        }

        static void CheckRestorationStatus(AmazonS3 client, string bucketName,
 string objectKey)
        {
           GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRe
quest()
            {
                  BucketName = bucketName,
                  Key = objectKey
            };
            GetObjectMetadataResponse response = client.GetObject
Metadata(metadataRequest);
           Console.WriteLine("Restoration status: {0}", response.RestoreInPro
gress);
            if (response.RestoreInProgress == false)
                Console.WriteLine("Restored object copy expires on: {0}", re
sponse.RestoreExpiration);
        }
    }
}
```

Restore an Object Using REST API

Amazon S3 provides an API for you to initiate an archive restoration. For more information, go to POST Object restore in the *Amazon Simple Storage Service API Reference*.

Access Control

Topics

Amazon S3 enables you to manage access to objects and buckets using access control lists (ACLs), bucket policies and IAM policies. You can use them independently or together. This section describes both.

An ACL is a list of grants. A grant consists of one grantee and one permission to access Amazon S3 resources (buckets and objects). ACLs only grant permissions; they do not deny them. ACLs can contain the following grantee types:

- Specific AWS accounts
- All AWS accounts
- Any anonymous request

Bucket policies provide access control management at the bucket level for both a bucket and the objects in it. Bucket policies are a collection of JSON statements written in the *access policy language*. The policies provide a fine granularity of access control for Amazon S3 resources. The policies also allow you to set permissions for a large number of objects with one statement.

AWS Identity and Access Management (IAM) enables you to create multiple users within your AWS account and manage their permissions via IAM policies. These policies are attached to the users, enabling centralized control of permissions for users under your AWS account. Note that bucket policies are attached to a bucket and the IAM policies are attached to individual users in your account.

Using IAM Policies

Topics

AWS Identity and Access Management (IAM) enables you to create multiple users within your AWS account, assign them security credentials, and manage their permissions. You manage user permissions via IAM policies. These policies are attached to the users, enabling centralized control of permissions for users under your AWS account.

Amazon S3 supports resource-based permissions on objects and buckets using access control list (ACL) and bucket policies. The ACLs and bucket policies are attached to buckets and objects defining which AWS accounts (or other groups of actors) have access and the type of access.

You can use both of these in conjunction with IAM user policies to control access to your Amazon S3 resources. The following table summarizes similarities and differences between ACLs, bucket policies, and IAM policies.

Type of Access Control	AWS Account-Level Control?	User-Level Control?	Format
ACLs	Yes	No	Special XML-based format defined by Amazon S3
Bucket policies	Yes	Yes	Access policy language
IAM policies	No	Yes	Access policy language

With ACLs, you can only grant other AWS accounts access to your Amazon S3 resources. With IAM policies, you can only grant users within your own AWS account permission to your Amazon S3 resources. With bucket policies, you can do both.

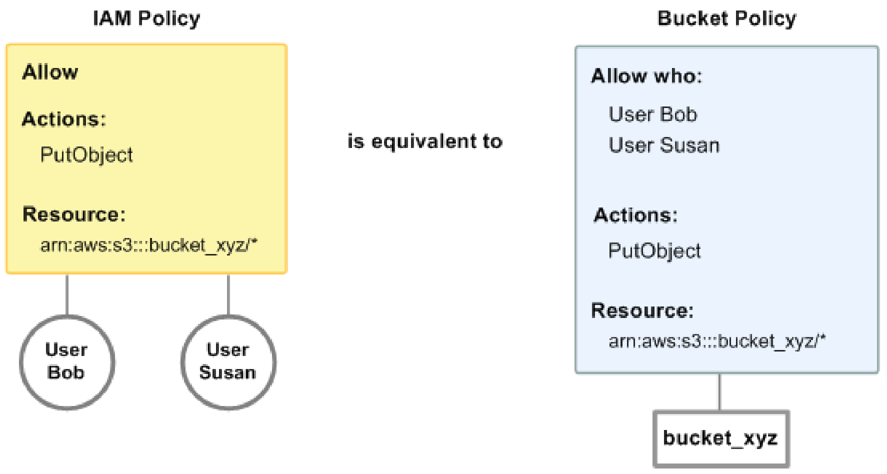
This section describes how IAM works with bucket policies and ACLs.

IAM and Bucket Policies Together

When someone sends a request to use your AWS account's Amazon S3 resources, we evaluate all applicable ACLs, bucket policies, and IAM policies together to determine whether to give the requester access.

Although you can still use ACLs to grant permission, this section focuses on how to use IAM policies and bucket policies to give *users in your AWS account* permissions to access your Amazon S3 resources. You can use bucket policies, IAM policies, or both. For the most part, you can achieve the same results with either. For example, the following diagram shows an IAM policy and a bucket policy that are equivalent. The IAM policy (on the left) allows the Amazon S3 `PutObject` action for the bucket called bucket_xyz in your AWS account, and it's attached to the users Bob and Susan (which means Bob and Susan have the permissions stated in the policy).

The bucket policy (on the right) is attached to bucket_xyz. As with the IAM policy, the bucket policy gives Bob and Susan permission to access `PutObject` on bucket_xyz.

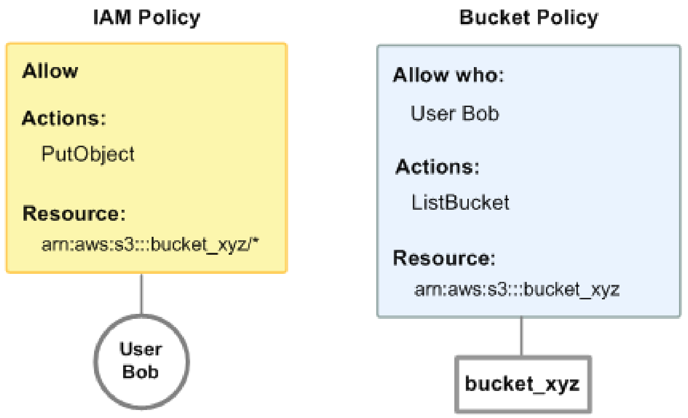


Note
The preceding example shows simple policies with no conditions. You could specify a particular condition in either policy and get the same result.

IAM policies lets you manage access to your Amazon S3 resources based on user; whereas bucket policies let you manage access based on the specific resources. The following examples further illustrate how the two policy systems work together.

Example 1

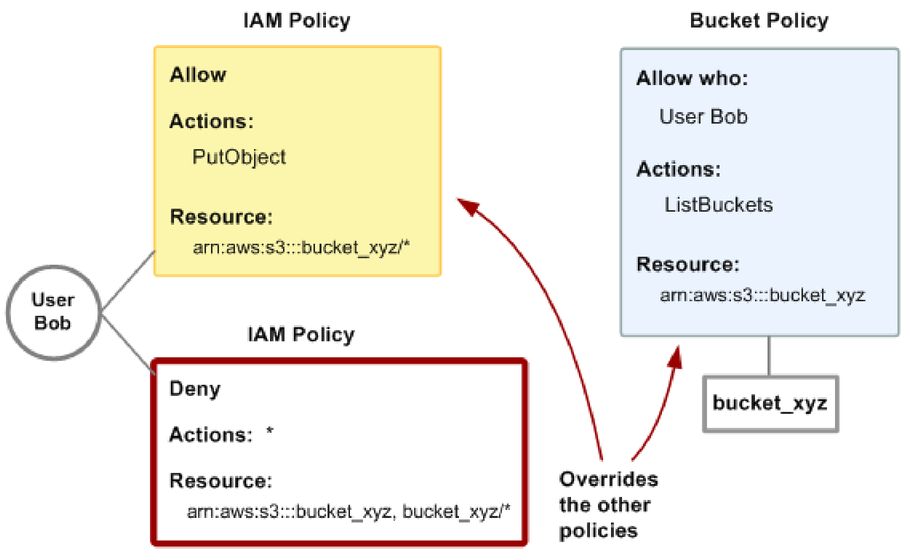
In this example, Bob has both an IAM policy and a bucket policy that apply to him. The IAM policy gives him permission to use `PutObject` on bucket_xyz, whereas the bucket policy gives him permission to use `ListBucket` on that same bucket. The following diagram illustrates the concept.



If Bob were to send a request to put an object into bucket_xyz, the IAM policy would allow the action. If Bob were to send a request to list the objects in bucket_xyz, the bucket policy would allow the action.

Example 2

In this example, we build on the previous example (where Bob has two policies that apply to him). Let's say that Bob abuses his access to bucket_xyz, so you want to remove his entire access to that bucket. The easiest thing to do is add a policy that denies him access to all actions on the bucket. This third policy overrides the other two, because an explicit deny always overrides an allow (for more information about policy evaluation logic, see Evaluation Logic (p. 446)). The following diagram illustrates the concept.



Alternatively, you could add an additional statement to the bucket policy that denies Bob any type of access to the bucket. It would have the same effect as adding a IAM policy that denies him access to the bucket.

For examples of policies that cover Amazon S3 actions and resources, see Example Policies for Amazon S3 (p. 289). For more information about writing S3 policies, go to the Amazon Simple Storage Service Developer Guide.

Permissions for Resource Creator

Amazon S3 by default gives the AWS account that created the bucket or object full permissions on that resource. However, if a user (not the AWS account) creates a bucket or object, that user doesn't by default have permission to perform other actions on that resource. The user must be granted the additional permissions in an IAM policy.

Amazon S3 ARNs

For Amazon S3, the resources you can specify in a policy are buckets and objects. The Amazon Resource Name (ARN) follows this format:

```
arn:aws:s3:::bucket_name/key_name
```

where `arn:aws:s3:::bucket_name` refers only to the bucket, and the entire string `arn:aws:s3:::bucket_name/key_name` refers to the object.

For example:

```
arn:aws:s3:::example_bucket/developers/design_info.doc
```

The Region and AWS Account ID portions of the ARN must be empty because bucket names are global.

Amazon S3 Actions

The Amazon S3 actions that you can specify in a policy are divided into groups based on the type of resource.

Actions Related to Objects

- `s3:GetObject` (covers REST GET Object, REST HEAD Object, REST GET Object torrent, SOAP `GetObject`, and SOAP `GetObjectExtended`)
- `s3:GetObjectVersion` (covers REST GET Object, REST HEAD Object, REST GET Object torrent, SOAP `GetObject`, and SOAP `GetObjectExtended`)
- `s3:PutObject` (covers the REST PUT Object, REST POST Object, REST Initiate Multipart Upload, REST Upload Part, REST Complete Multipart Upload, SOAP `PutObject`, and SOAP `PutObjectInline`)
- `s3:GetObjectAcl`
- `s3:GetObjectVersionAcl`
- `s3:PutObjectAcl`
- `s3:PutObjectVersionAcl`
- `s3:DeleteObject`
- `s3:DeleteObjectVersion`
- `s3:ListMultipartUploadParts`
- `s3:AbortMultipartUpload`
- `s3:GetObjectTorrent`
- `s3:GetObjectVersionTorrent`
- `s3:RestoreObject`

Actions Related to Buckets

- `s3:CreateBucket`
- `s3:DeleteBucket`
- `s3:ListBucket`
- `s3:ListBucketVersions`
- `s3:ListAllMyBuckets` (covers REST GET Service and SOAP `ListAllMyBuckets`)
- `s3:ListBucketMultipartUploads`

Actions Related to Bucket Sub-Resources

- `s3:GetBucketAcl`
- `s3:PutBucketAcl`
- `s3:GetBucketVersioning`
- `s3:PutBucketVersioning`
- `s3:GetBucketRequestPayment`
- `s3:PutBucketRequestPayment`
- `s3:GetBucketLocation`

- `s3:GetBucketPolicy`
- `s3:DeleteBucketPolicy`
- `s3:PutBucketPolicy`
- `s3:GetBucketNotification`
- `s3:PutBucketNotification`
- `s3:GetBucketLogging`
- `s3:PutBucketLogging`
- `s3:GetBucketWebsite`
- `s3:PutBucketWebsite`
- `s3:DeleteBucketWebsite`
- `s3:GetLifecycleConfiguration`
- `s3:PutLifecycleConfiguration`

You can delete objects by explicitly calling the DELETE Object API or configure its lifecycle (see Object Expiration (p. 114)) to enable Amazon S3 to remove them for you. If you want to block users or accounts from removing or deleting objects from your bucket you must deny them `s3:DeleteObject`, `s3:DeleteObjectVersion` and `s3:PutLifecycleConfiguration` actions.

Amazon S3 Policy Keys

Policy keys let you restrict access to resources based on information other than just the API action being requested. They let you restrict access based on contextual information about the request, such as the IP address of the requester, the time and date of the request, etc. Amazon S3 has a rich integration with IAM that lets you restrict access based on a wide variety of contextual information about the request. For example, you can restrict access based on values of general HTTP headers or Amazon S3-specific headers in the request. This section lists the policy keys available for you to use with policies that restrict access to your AWS account's Amazon S3 resources. For more information about policy keys, see Condition (p. 452).

Amazon S3 implements the AWS-wide policy keys in the following table.

Available Keys

AWS provides a set of common keys supported by all AWS products that adopt the access policy language for access control. These keys are:

- `aws:CurrentTime`–Key to use with date conditions to restrict access based on request time. For date/time conditions (see Date Conditions (p. 456))
- `aws:MultiFactorAuthAge`–Key that provides a numeric value indicating how long ago (in seconds) the temporary credential for Multi-Factor Authentication (MFA) validation, included in the request, was created.
- `aws:SecureTransport`–Boolean representing whether the request was sent using SSL (see Boolean Conditions (p. 456))
- `aws:SourceIp`–The requesters IP address, for use with IP address conditions (see IP Address (p. 457))
- `aws:UserAgent`–Information about the requesters client application, for use with string conditions (see String Conditions (p. 455))
- `aws:EpochTime`–Number of seconds since epoch.
- `aws:Referer`–Same as the HTTP *`referer`* field.

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

Note

If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

Each AWS service that uses the access policy language might also provide service-specific keys. For a list of service-specific keys you can use, see Special Information for Amazon S3 Policies (p. 459).

Amazon S3 also has action-specific policy keys. They're grouped by resource type and applicable action in the following tables. Some policy keys are applicable to more than one resource type or action.

Important

IAM cannot evaluate a policy for validity within Amazon S3. If you specify an invalid key/action combination in the policy, IAM doesn't throw an error when you upload the policy to IAM. Also, you will not receive an error message from Amazon S3. Amazon S3 can determine only that the policy doesn't apply because it cannot fulfill the conditions. However, if you use a policy condition in an unexpected way (for example, you use a string field as a numeric comparison), Amazon S3 will throw an exception on the request and access will be denied.

Unless otherwise noted, each key is for use with the access policy language's string conditions (for more information, see String Conditions (p. 455)).

Object Keys in Amazon S3 Policies

The following list shows the keys related to objects that can be in Amazon S3 policies.

Action	Applicable Keys	Description
`s3:PutObject`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-copy-source`	The header that specifies the name of the source bucket and key name of the source object, separated by a slash (/). Used when copying an object. Example value: `/bucketname/keyname`
	`s3:x-amz-grant-read, s3:x-amz-grant-write, s3:x-amz-grant-read-acp, s3:x-amz-grant-write-acp , s3:x-amz-grant-full -control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Object. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).
	`s3:x-amz-server-side -encryption`	Allow the specific action only if `x-amz-server-side-encryption` header is present in the request and its value matches the specified condition. with Valid values: AES256 Example value: AES256
	`s3:x-amz-metadata-directive`	

Action	Applicable Keys	Description
		The header that specifies whether the metadata is copied from the source object or replaced with metadata provided in the request. If copied, the metadata, except for the version ID, remains unchanged. Otherwise, all original metadata is replaced by the metadata you specify. Used when copying an object. Valid values: `COPY` | `REPLACE`. The default is `COPY`. Example value: `REPLACE`
`s3:PutObjectAcl`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Object acl. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).
`s3:GetObjectVersion`	`s3:VersionId`	The version ID of the object being retrieved. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`
`s3:GetObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being retrieved. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`

Action	Applicable Keys	Description
`s3:PutObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being PUT. Example value: `Upfdndhfd8438MNFDN93 jdnJFkdmqnh893`
	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Object acl. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).
`s3:DeleteObjectVersion`	`s3:VersionId`	The version ID of the object being deleted. Example value: `Upfdndhfd8438MNFDN93 jdnJFkdmqnh893`

Bucket Keys in Amazon S3 Policies

The following table shows the keys related to buckets that can be in Amazon S3 policies.

Action	Applicable Keys	Description
`s3:CreateBucket`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:LocationConstraint`	Specifies the Region where the bucket will be created. Valid values are `us-west-1` (for Northern California) or `EU` (for Ireland). Do not specify a value for US Standard. Example value: `us-west-1`
	`s3:x-amz-grant-read, s3:x-amz-grant-write, s3:x-amz-grant-read-acp, s3:x-amz-grant-write-acp, s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Bucket. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).

Action	Applicable Keys	Description
`s3:ListBucket`	`s3:prefix`	Limits the response to objects that begin with the specified prefix. Use this to allow or deny access to objects that begin with the prefix. Example value: `home`
	`s3:delimiter`	The character you use to group objects. Example value: `/`
	`s3:max-keys`	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions (for more information, see Numeric Conditions (p. 455)). Example value: `100`
`s3:ListBucketVersions`	`s3:prefix`	Header that lets you limit the response to include only keys that begin with the specified prefix. Example value: `home`
	`s3:delimiter`	The character you use to group objects. Example value: `/`
	`s3:max-keys`	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions (for more information, see Numeric Conditions (p. 455)). Example value: `100`

Action	Applicable Keys	Description
`s3:PutBucketAcl`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Bucket acl. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).

Example Policies for Amazon S3

This section shows several simple IAM policies for controlling user access to Amazon S3.

Note
In the future, Amazon S3 might add new actions that should logically be included in one of the following policies, based on the policy's stated goals.

Example 1: Allow each user to have a home directory in Amazon S3

In this example, we create a policy that we'll attach to the user named Bob. The policy gives Bob access to the following home directory in Amazon S3: my_corporate_bucket/home/bob. Bob is allowed to access only the specific Amazon S3 actions shown in the policy, and only with the objects in his home directory.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":["s3:PutObject","s3:GetObject","s3:GetObjectVersion",
      "s3:DeleteObject","s3:DeleteObjectVersion"],
      "Resource":"arn:aws:s3:::my_corporate_bucket/home/bob/*"
   }
   ]
}
```

Example 2: Allow a user to list only the objects in his or her home directory in the corporate bucket

This example builds on the previous example that gives Bob a home directory. To give Bob the ability to list the objects in his home directory, he needs access to `ListBucket`. However, we want the results to include only objects in his home directory, and not everything in the bucket. To restrict his access that way, we use the policy condition key called `s3:prefix` with the value set to `home/bob/*`. This means that only objects with a prefix `home/bob/*` will be returned in the `ListBucket` response.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":"s3:ListBucket",
      "Resource":"arn:aws:s3:::my_corporate_bucket",
      "Condition":{
           "StringLike":{
              "s3:prefix":"home/bob/*"
              }
          }
      }
   ]
}
```

Example 3: Allow a group to have a shared directory in Amazon S3

In this example, we create a policy that gives the group access to the following directory in Amazon S3: my_corporate_bucket/share/marketing. The group is allowed to access only the specific Amazon S3 actions shown in the policy, and only with the objects in the specific directory.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":["s3:PutObject","s3:GetObject","s3:GetObjectVersion",
                 "s3:DeleteObject","s3:DeleteObjectVersion"],
      "Resource":"arn:aws:s3:::my_corporate_bucket/share/marketing/*"
      }
   ]
}
```

Example 4: Allow all your users to read objects in a portion of the corporate bucket

In this example we create a group called *AllUsers* and put all the AWS account's users in the group. We then attach a policy that gives the group access to `GetObject` and `GetObjectVersion`, but only for objects in the my_corporate_bucket/readonly directory.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":["s3:GetObject","s3:GetObjectVersion"],
      "Resource":"arn:aws:s3:::my_corporate_bucket/readonly/*"
      }
   ]
}
```

Example 5: Allow a partner to drop files into a specific portion of the corporate bucket

In this example, we create a group called WidgetCo that represents the partner company, then create a user for the specific person (or application) at the partner company who needs access, and then put the user in the group.

We then attach a policy that gives the group `PutObject` access to the following directory in the corporate bucket: my_corporate_bucket/uploads/widgetco.

We also want to prevent the WidgetCo group from doing anything else with the bucket, so we add a statement that denies permission to any Amazon S3 actions except `PutObject` on any Amazon S3 resource in the AWS account. This is only necessary if there's a broad policy in use elsewhere in your AWS account that gives users wide access to Amazon S3.

```
{
    "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/*"
    }]
}
```

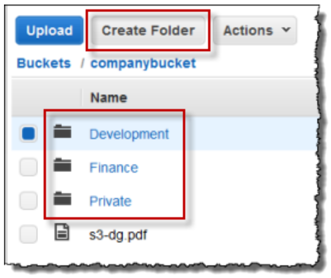
An Example: Using IAM policies to control access to your bucket

Topics

This walkthrough explains how user permissions work with Amazon S3. We will create a bucket with folders, and then we'll create Amazon IAM users in your AWS account and grant those users incremental permissions on your Amazon S3 bucket and the folders in it.

Background: Basics of Buckets and Folders

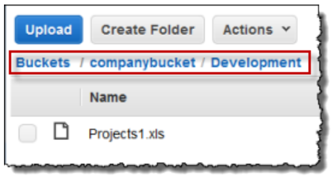
The Amazon S3 data model is a flat structure: you create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders; however, you can emulate a folder hierarchy. Tools such as the Amazon S3 Console can present a view of these logical folders and subfolders in your bucket, as shown here:



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 '/' delimiter.
- When you upload an object named **project1.pdf** in the **Development** folder, the console uploads the object and gives it the key **Development/project1.pdf**.

  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. In the console, when you double-click 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/project1.pdf`.

The console is using object keys to infer a logical hierarchy; Amazon S3 has no physical hierarchy, only buckets that contain objects in a flat file structure. When you create objects by 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 we will do in this walkthrough.

Before going into the walkthrough, you need to familiarize yourself with one more concept, the "root-level" bucket content. Suppose your `companybucket` bucket has the following objects:

Private/privDoc1.txt

Private/privDoc2.zip

Development/project1.xls

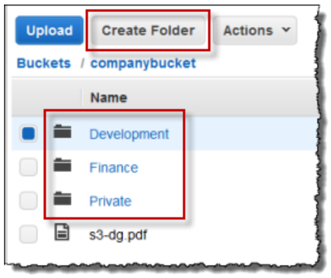
Development/project2.xls

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 click the bucket name in the Amazon S3 console, the root-level items appear as shown. 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 Example

The example for this walkthrough is as follows:

- You create a bucket and then add three folders (Private, Development, and Finance) to it.
- You have two users, Alice and Bob. You want Alice to access only the Development folder and Bob to access only the Finance folder, and you want to keep the Private folder content private. In the walkthrough, you manage access by creating AWS Identity and Access Management (IAM) users (we will use the same user names, Alice and Bob) and grant 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 will need some common permissions. So you will also create a group named Consultants and then add both Alice and Bob to the group. You will first grant permissions by attaching a group policy to the group. Then you will 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 will need to replace the bucket name with a name that you create.

Step 0: Preparing for the Walkthrough

In this example, you will use your AWS account credentials to create IAM users. Initially, these users have no permissions. You will incrementally grant these users permissions to perform specific Amazon S3 actions. To test these permissions, you will 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 into the AWS Management Console with your AWS account credentials, go to http://aws.amazon.com/console. An IAM user cannot sign in by 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.

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, click **IAM Dashboard** .
3. Under **AWS Account Alias**, 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.

For more information about IAM, go to The AWS Management Console Sign-in Page in *AWS Identity and Access Management Using IAM*.

Step 1: Create a Bucket

In this step, you will sign in to the Amazon S3 console with your AWS account credentials, create a bucket, add folders (Development, Finance, 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, go to Creating a Bucket in the *Amazon Simple Storage Service Console User Guide*.
3. Upload one document to the bucket.
   This exercise assumes 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, go to 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 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, go to Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.

Step 2: Create IAM Users and a Group

Now you use the IAM console to add two IAM users, Alice and Bob, to your AWS account. You will also create an administrative group named Consultants, and then you'll add both users to the group.

Caution
When you add users and a group, do not attach any policies that grant permissions to these users. At first, these users will not have any permissions. In the following sections, you will incrementally grant permissions. You must first ensure that you have assigned the passwords to these IAM users. You will use these user credentials to test Amazon S3 actions and verify that the permissions work as expected.

For step-by-instructions on creating a new IAM user, go to Adding a New User to Your AWS Account in *Using AWS Identity and Access Management*.

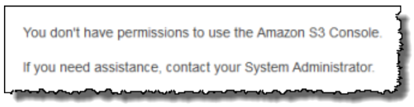
For step-by-step instructions on creating an administrative group, go to Creating an Admins Group section in the *Using AWS Identity and Access Management* guide.

Step 3: Verify that IAM Users Have No Permissions

If you are using two browsers, you can now use the second browser to sign into 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. 294)), sign into the AWS 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 you have no permissions.


You don't have permissions to use the Amazon S3 Console.

If you need assistance, contact your System Administrator.

Now, let's begin granting incremental permissions to the users. First, you will attach a group policy that grants permissions that both users must have.

Step 4: Grant Group-Level Permissions

We want all our 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.

Now we'll create a policy that grants these permissions and attach it to the group.

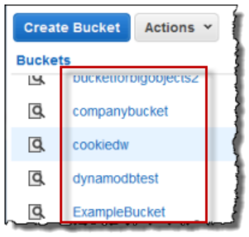
Step 4.1: Grant Permission to List All Buckets

Let's use the following policy to grant the users minimum permissions to enable them to list all buckets owned by the parent account.

```
{
  "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 permision 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 determine if specific permission is allowed or denied.

When you attach the preceding policy to a user or a group, you allow the user or group permission to obtain a list of buckets owned by the parent AWS account.



You attach policy documents to IAM users and groups in the IAM console. Because we want both our users to be able to list the buckets, let's attach the policy to the group using the following procedure.

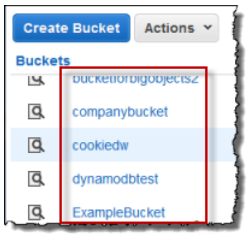
> **Note**
> You must be signed in with your AWS account credentials, not as an IAM user, to grant user permissions.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Attach the custom policy that you created earlier to the group.

   For step-by-step instructions, go to Managing IAM Policies in *Using AWS Identity and Access Management*.
3. Test the permission.

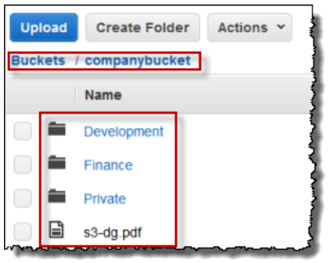
   1. Using the IAM user sign-in link (see To provide a sign-in link for IAM users (p. 294)), sign into the AWS console using any one of IAM user credentials. .
   2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.

      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

Now let's allow all users to list the root-level `companybucket` bucket items. When a user clicks the company bucket in the Amazon S3 console, he or she will be able to see the root-level items in the bucket.



Remember, we are using `companybucket` for illustration. You will use the name of the bucket that you created for this exercise.

To understand what request the console sends to Amazon S3 when you click a bucket name, the response Amazon S3 returns, and how the console interprets the response, it is necessary to take a little deep dive.

When you click a bucket name, the console sends the GET Bucket (List Objects) request to Amazon S3. This request includes the following parameters:

- `prefix` parameter with an empty string as its value.
- `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=
```

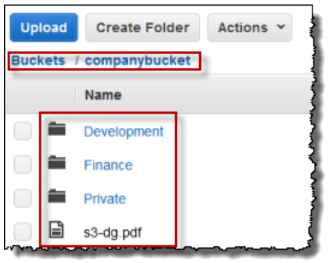
Amazon S3 returns a response that includes the following `<ListBucketResult/>` element:

```
<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` does not contain the `'/'` delimiter, and Amazon S3 returns the key in the `<Contents/>` element. However, all other keys in our 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.



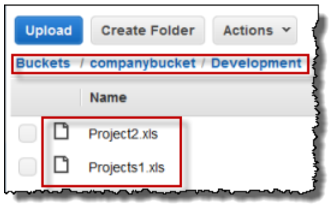
Now, if Bob or Alice double-clicks the **Development** folder, the console sends the GET Bucket (List Objects) request to Amazon S3 with the `prefix` and the `delimiter` parameters set to the following values:

- `prefix` parameter with value `Development/`.
- `delimiter` parameter with `'/'` 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, let's 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, we add a condition that users must specify an empty `prefix` in the request—that is, they are not allowed to double-click any of our root-level folders. Finally, we will add a condition to require folder-style access by requiring user requests to include the `delimiter` parameter with value `'/'`.

```
{
  "Sid": "AllowRootLevelListingOfCompanyBucket",
  "Action": ["s3:ListBucket"],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::companybucket"],
  "Condition":{
                "StringEquals":{"s3:prefix":[""]},
                "StringEquals":{"s3:delimiter:["/"]}
              }
}
```

When you use the Amazon S3 console, note that when you click a bucket, 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 group policy with the following custom policy, which also allows the `s3:ListBucket` action.

   For step-by-step instructions, go to Managing IAM Policies.

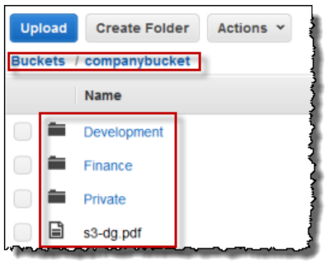
   ```
   {
     "Statement": [
       {
   ```

```
     "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequired
ForListBucket",
      "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":[""]},
                    "StringEquals":{"s3:delimiter":["/"]}
                 }
    }
  ]
}
```

3. Test the updated permissions.

   1. Using the IAM user sign-in link (see To provide a sign-in link for IAM users (p. 294)), sign in to the AWS Management Console.

      Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
   2. Click the bucket that you created for this exercise, and the console will now show the root-level bucket items. If you click any folders in the bucket, you will not be able to see the folder content, because you have not yet granted those permissions.



This test succeeds when users use the Amazon S3 console because when you click a bucket in 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 4.4: 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 cannot do much. Let's grant user-specific permissions, as follows:

- Permit Alice to get and put objects in the Development folder.

- Permit 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 we grant additional permissions to Alice so 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 attach a policy to the user that grants permission for the `s3:ListBucket` action on the `companybucket` bucket, provided the request includes the prefix `Development/`.

```
{
  "Statement": [
    {
      "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition":{  "StringLike":{"s3:prefix":["Development/*"] }
       }
    }
  ]
}
```

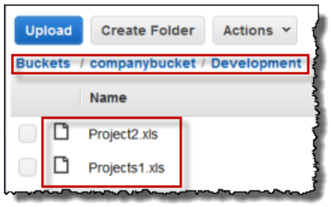
1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.
2. Attach the preceding policy to the user Alice.

   For step-by-step instructions, go to Managing Policies (AWS Management Console) in the *Using AWS Identity and Access Management* guide.
3. Test the chnage to Alice's permissions:

   a. Using the IAM user sign in link (see To provide a sign-in link for IAM users (p. 294)), sign in to the AWS Management Console.
   b. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
   c. In the Amazon S3 console, verify that Alice can see the list of objects in the `Development/` folder in the bucket.

      When the user clicks 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 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. Attach the following policy to the user Alice.

```
{
   "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/*"]
      }
   ]
}
```

   For step-by-step instructions, go to Managing Policies (AWS Management Console) in the *Using AWS Identity and Access Management* guide.
3. Test the updated policy:

1. Using the IAM user sign-in link (see To provide a sign-in link for IAM users (p. 294)), sign into the AWS Management Console.
2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
3. In 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 tighen 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 `Developerment/*` 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 }
    }
}
```

Note that there are two conditional expressions in the `Condition` block. The result of these conditional expressions is combined by using the logical OR. If either condition is true, the result of the combined condition is true.

- 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 will evaluate to true, and so the entire `Condition` will evaluate to true. You must allow an empty string as value of the `prefix` parameter. You recall from the preceding disussion, 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. 297).
- The `StringNotLike` conditional expression ensures that if the value of the `prefix` parameter is specified and is not `Developement/*`, the request will fail.

The updated user policy is given below. Follow the steps in the preceding section and replace the policy attached to user Alice with this updated policy.

```
{
   "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 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. 301).

Step 7: Secure the Private Folder

In this example, you have only two users. You granted all the minimum rquired 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, we don't want any of the users in this example to access the content of the Private folder. How do you ensure 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 follow 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": "ExplictDenyAccessToPrivateFolderToEveryoneInTheGroup",
  "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. In the console, if Bob or Alice double-clicks 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 group policy with the following updated policy, which includes the preceding deny statements. Now, none of the users in the group will be able to access the Private folder in your bucket.

```
{
  "Statement": [
    {
      "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequired
ForListBucket",
      "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": "ExplictDenyAccessToPrivateFolderToEveryoneInTheGroup",
      "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/"]}
       }
    }
  ]
}
```

Cleanup

In order to clean up, go to the IAM console and remove the users Alice and Bob. For step-by-step instructions, go to Deleting a User from Your AWS Account in the *Using AWS Identity and Access Management* 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 .

Using Bucket Policies

Topics

The following sections explain how to set and manage bucket policies on Amazon S3 resources.

Writing Bucket Policies

Bucket policies define access rights for Amazon S3 resources. Only a bucket owner can write bucket policies. A bucket owner can write a bucket policy to:

- Allow/deny bucket-level permissions.
- Deny permission on any objects in the bucket. Because the bucket owner is fiscally responsible for the bucket, the owner can write a bucket policy to deny permissions on any objects in a bucket.
- Grant permission on objects in the bucket only if the bucket owner is the object owner. For objects owned by other accounts the object owner must manage permissions using ACLs.

The policy itself is written in JSON and uses the access policy language. To learn about the details of the access policy language and how to write a bucket policy, see The access policy language (p. 441).

AWS Policy Generator Tool

You can use the AWS Policy Generator tool to create a bucket policy for your Amazon S3 bucket. You can then use the generated document to set your bucket policy using the Amazon S3 console, a number of third party tools or via your application. To use the policy generation tool, go to AWS Policy Generator.

Setting Bucket Policies on a Bucket

To set a policy on a bucket, you use the `PUT Bucket` operation on the *`policy`* sub-resource and you include the bucket policy in the body of the request. The following request, for example, allows two users (`1111-2222-3333`, `4444-5555-6666`) access execute a `GET` request (`s3:GetObject*`) for objects in `mybucket` (`arn:aws:s3:::mybucket/*",`):

```
PUT /?policy HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Tue, 04 Apr 2010 20:34:56 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:VGhpcyBSAMPLEBieSBlbHZpbmc=

{
"Version":"2008-10-17",
"Id":"aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect":"Allow",
        "Sid":"1",
        "Principal" : {
            "AWS":["1111-2222-3333","4444-5555-6666"]
        },
        "Action":["s3:GetObject*"],
        "Resource":"arn:aws:s3:::mybucket/*"
    }
 ]
}
```

Note
The *`Resource`* value must include the bucket name.

To attach a policy to a bucket, you must be the bucket owner. The bucket owner by default has permissions to attach bucket policies to their buckets using `PUT Bucket policy`. If the bucket already has a policy, the one in this request completely replaces it

For more information, go to PUT Bucket policy in the *Amazon S3 API Reference*.

Returning the Bucket Policies on a Bucket

To return the bucket policy on a specified bucket, use the `GET Bucket` operation with the *`policy`* sub-resource. The following request returns the bucket policy for the bucket, `mybucket.s3.amazonaws.com`:

```
GET ?policy HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:VGhpcyBSAMPLEBieSBlbHZpbmc=
```

The bucket owner by default has permissions to retrieve bucket policies using `GET Bucket policy`.

For more information, go to GET Bucket policy in the *Amazon S3 API Reference*.

Deleting Bucket Policies on a Bucket

To delete a policy associated with a bucket, use the `DELETE Bucket` operation with the *`policy`* sub-resource. The following request deletes the bucket policy associated with `mybucket`:

```
DELETE /?policy HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Tue, 04 Apr 2010 20:34:56 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:VGhpcyBSAMPLEeSBlbHZpbmc=
```

To use the delete operation, you must have *`DeletePolicy`* permissions on the specified bucket and be the bucket owner. The bucket owner by default has permissions to delete bucket policies.

For more information, go to DELETE Bucket policy in the *Amazon S3 API Reference*.

Example Cases for Amazon S3 Bucket Policies

This section gives a few examples of typical use cases for bucket policies.

Note
You can use the AWS Policy Generator tool to create a bucket policy for your Amazon S3 bucket. You can then use the generated document to set your bucket policy using the Amazon S3 console, a number of third party tools or via your application. To use the policy generation tool, go to AWS Policy Generator.

Granting Permissions to Multiple Accounts with Added Restrictions

The following example policy grants PutObject, and PutObjectAcl permissions to multiple accounts and requires that the public-read canned acl is included.

```
{
  "Version":"2008-10-17",
  "Statement":[{
 "Sid":"AddCannedAcl",
        "Effect":"Allow",
   "Principal": {
            "AWS":
["arn:aws:iam::111122223333:root","arn:aws:iam::444455556666:root"]
         },
   "Action":["s3:PutObject","s3:PutObjectAcl"
      ],
      "Resource":["arn:aws:s3:::bucket/*"
      ],
      "Condition":{
        "StringEquals":{
          "s3:x-amz-acl":["public-read"]
        }
      }
    }
  ]
}
```

Granting Permission to an Anonymous User

The following example policy grants permissions to anonymous users.

```
{
  "Version":"2008-10-17",
  "Statement":[{
 "Sid":"AddPerm",
        "Effect":"Allow",
   "Principal": {
            "AWS": "*"
         },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::bucket/*"
      ]
    }
  ]
}
```

Restricting Access to Specific IP Addresses

This statement grants permissions to any user to perform any S3 action 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 192.168.143.* range of allowed IP addresses with one exception, 192.168.143.188.

Note that the `IPAddress` and `NotIpAddress` values specified in the condition uses CIDR notation described in RFC 2632. For more information, go to http://www.rfc-editor.org/rfc/rfc4632.txt.

```
{
    "Version": "2008-10-17",
    "Id": "S3PolicyId1",
    "Statement": [
        {
            "Sid": "IPAllow",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::bucket/*",
            "Condition" : {
                "IpAddress" : {
                    "aws:SourceIp": "192.168.143.0/24"
                },
                "NotIpAddress" : {
                    "aws:SourceIp": "192.168.143.188/32"
                }
            }
        }
    ]
}
```

Restricting Access to Specific HTTP Referer

The following example policy restricts access based on HTTP Referer.

```
{
  "Version":"2008-10-17",
  "Id":"http referer policy example",
  "Statement":[
    {
      "Sid":"Allow get requests referred by www.mysite.com and mysite.com",
      "Effect":"Allow",
      "Principal":"*",
      "Action":"s3:GetObject",
      "Resource":"arn:aws:s3:::example-bucket/*",
      "Condition":{
        "StringLike":{
          "aws:Referer":[
            "http://www.mysite.com/*",
            "http://mysite.com/*"
          ]
        }
      }
    }
  ]
}
```

Granting Permissions to Enable Log Delivery to an S3 Bucket

The following example policy enables log delivery to your Amazon S3 bucket. The account specified in the following policy is the Log Delivery group. You must use the ARN specified in this policy because it identifies the Log Delivery group. For more information, see Setting Up Server Access Logging (p. 428).

```
{
 "Version":"2008-10-17",
 "Id":"LogPolicy",
 "Statement":[{
   "Sid":"Enables the log delivery group to publish logs to your bucket ",
   "Effect":"Allow",
   "Principal":{
    "AWS":"arn:aws:iam::111122223333:root"
   },
   "Action":["s3:GetBucketAcl",
    "s3:GetObjectAcl",
    "s3:PutObject"
   ],
   "Resource":["arn:aws:s3:::example-bucket",
    "arn:aws:s3:::example-bucket/*"
   ]
  }
 ]
}
```

Granting Permission, Using Canonical ID, to a CloudFront Origin Identify

The following example bucket policy grants a CloudFront Origin Identity permission to GET all objects in your Amazon S3 bucket. The CloudFront Origin Identity is used to enable CloudFront's private content feature. The policy uses the CanonicalUser prefix, instead of AWS, to specify a Canonical User ID. To learn more about CloudFront's 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.

```
{
 "Version":"2008-10-17",
 "Id":"PolicyForCloudFrontPrivateContent",
 "Statement":[{
   "Sid":" Grant a CloudFront Origin Identity access to support private content",

    "Effect":"Allow",
    "Principal":{
    "CanonicalUser":"79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be"
    },
    "Action":"s3:GetObject",
    "Resource":"arn:aws:s3:::example-bucket/*"
  }
 ]
}
```

Adding Bucket Policy to Require MFA Authentication

Amazon S3 supports MFA-protected API access, a feature that can enforce AWS Multi-Factor Authentication for access to your S3 resources. AWS 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 a 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 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 were 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 action on the `/taxdocuments` folder in the `examplebucket` bucket if the request is not MFA authenticated. To learn more about MFA authentication, go to Configuring MFA-Protected API Access.

```
{
   "Version": "2008-10-17",
   "Id": "123",
   "Statement": [
      {
         "Sid": "",
         "Effect": "Deny",
         "Principal": { "AWS": "*" },
         "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` action 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": "2008-10-17",
   "Id": "123",
   "Statement": [
      {
         "Sid": "",
         "Effect": "Deny",
         "Principal": { "AWS": "*"},
         "Action": "s3:**",
         "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
         "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
      },
      {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {"AWS": "*" },
         "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 action if the `aws:MultiFactorAuthAge` key value indicates that the temporary session was created more than an hour ago (3,600 seconds).

```
{
   "Version": "2008-10-17",
   "Id": "123",
   "Statement": [
      {
         "Sid": "",
         "Effect": "Deny",
         "Principal": { "AWS": "*" },
         "Action": "s3:**",
         "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
         "Condition": {"Null": {"aws:MultiFactorAuthAge": true }
         }
      },
      {
         "Sid": "",
         "Effect": "Deny",
         "Principal": { "AWS": "*"},
         "Action": "s3:**",
         "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
         "Condition": {"NumericGreaterThan": {"aws:MultiFactorAuthAge": 3600 }
 }
      },
      {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {"AWS": "*"},
         "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 other AWS accounts 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 access to the bucket owner identified by the email address (xyz@amazon.com) full control.

The policy requires the `PutObject` request to include the `x-amz-grant-full-control` request header. For more information, go to PUT Object.

```
{
   "Version":"2008-10-17",
   "Statement":[
      {
         "Sid":"111",
         "Effect":"Allow",
         "Principal":{
            "AWS":"1111111111"
         },
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::examplebucket/*"
      },
      {
         "Sid":"112",
         "Effect":"Deny",
         "Principal":{
            "AWS":"1111111111"
         },
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::examplebucket/*",
         "Condition":{
            "StringNotEquals":{
                "s3:x-amz-grant-full-control":[
                   "emailAddress=xyz@amazon.com"
                ]
             }
          }
       }
    ]
}
```

How to Use Resources, Principals, Operations, and Conditions in Bucket Policies

Topics

Specifying Amazon S3 Resources in Bucket Policies

You can refer to buckets and objects in bucket policies. Amazon S3 policies use the Amazon Resource Name (ARN) format for specifying them, as follows:

```
arn:aws:s3:::[resourcename]
```

The resource name is the fully qualified name of a bucket or object that the user is requesting access to. For buckets, the resource name is `bucketname`, where *`bucketname`* is the name of the bucket. For objects, the format for the resource's name is `bucketname/keyname`, where *`bucketname`* is the name of the bucket and *`keyname`* is the full name of the object. For example, if you have a bucket called “Ooyala” and an object with the name `shared/developer/settings.conf`, the resource name for the bucket would be Ooyala; for the object it would be `Ooyala/shared/developer/settings.conf`.

Specifying Principals in Bucket Policies

The `Principal` is one or more people who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). The AWS account ID can belong to either an AWS Account or an IAM User. You can specify multiple principals, or a wildcard (*) to indicate all possible users. You can view your account ID by logging in to your AWS account at http://aws.amazon.com and clicking **Account Activity** in the **Accounts** tab.

Instead of specifying an AWS account ID you can specify a Canonical User ID when granting permission to an AWS Account. You can view your Canonical User ID by logging in to your AWS account at http://aws.amazon.com and, clicking **Security Credentials** in the **Accounts** tab. You can also grant a CloudFront Origin Access Identify using the Canonical User ID associated with that identify. To learn more about CloudFront's support for serving private content, go to Serving Private Content topic in Amazon CloudFront Developer Guide. You must specify the Canonical User ID for your CloudFront distribution's origin identity, not your AWS Account.

In JSON, you use `"AWS":` as a prefix for the principal's AWS account ID and the `"CanonicalUser":` prefix for the principal’s AWS Canonical User ID.

> **Note**
> When you grant other AWS accounts access to your AWS 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, go to Enabling Cross-Account Access in Using Identity and Access Management.

Amazon S3 Actions

The following list shows the format for the Amazon S3 actions that you can reference in a policy.

Actions Related to Objects

- `s3:GetObject` (covers REST GET Object, REST HEAD Object, REST GET Object torrent, SOAP `GetObject`, and SOAP `GetObjectExtended`)
- `s3:GetObjectVersion` (covers REST GET Object, REST HEAD Object, REST GET Object torrent, SOAP `GetObject`, and SOAP `GetObjectExtended`)
- `s3:PutObject` (covers the REST PUT Object, REST POST Object, REST Initiate Multipart Upload, REST Upload Part, REST Complete Multipart Upload, SOAP `PutObject`, and SOAP `PutObjectInline`)
- `s3:GetObjectAcl`
- `s3:GetObjectVersionAcl`
- `s3:PutObjectAcl`
- `s3:PutObjectVersionAcl`
- `s3:DeleteObject`
- `s3:DeleteObjectVersion`
- `s3:ListMultipartUploadParts`
- `s3:AbortMultipartUpload`
- `s3:GetObjectTorrent`
- `s3:GetObjectVersionTorrent`
- `s3:RestoreObject`

Actions Related to Buckets

- `s3:CreateBucket`
- `s3:DeleteBucket`
- `s3:ListBucket`
- `s3:ListBucketVersions`
- `s3:ListAllMyBuckets` (covers REST GET Service and SOAP `ListAllMyBuckets`)
- `s3:ListBucketMultipartUploads`

Actions Related to Bucket Sub-Resources

- `s3:GetBucketAcl`
- `s3:PutBucketAcl`
- `s3:GetBucketVersioning`
- `s3:PutBucketVersioning`
- `s3:GetBucketRequesterPays`
- `s3:PutBucketRequesterPays`

- `s3:GetBucketLocation`
- `s3:PutBucketPolicy`
- `s3:GetBucketPolicy`
- `s3:PutBucketNotification`
- `s3:GetBucketNotification`
- `s3:GetBucketLogging`
- `s3:PutBucketLogging`
- `s3:GetLifecycleConfiguration`
- `s3:PutLifecycleConfiguration`

You can delete objects by explicitly calling the DELETE Object API or configure its lifecycle (see Object Expiration (p. 114)) to enable Amazon S3 to remove them for you. If you want to block users or accounts from removing or deleting objects from your bucket you must deny them `s3:DeleteObject`, `s3:DeleteObjectVersion` and `s3:PutLifecycleConfiguration` actions.

Available AWS Keys in Amazon S3 Policies

AWS provides a set of common keys supported by all AWS products that adopt the access policy language for access control. These keys are:

- `aws:CurrentTime`–Key to use with date conditions to restrict access based on request time. For date/time conditions (see Date Conditions (p. 456))
- `aws:MultiFactorAuthAge`–Key that provides a numeric value indicating how long ago (in seconds) the temporary credential for Multi-Factor Authentication (MFA) validation, included in the request, was created.
- `aws:SecureTransport`–Boolean representing whether the request was sent using SSL (see Boolean Conditions (p. 456))
- `aws:SourceIp`–The requesters IP address, for use with IP address conditions (see IP Address (p. 457))
- `aws:UserAgent`–Information about the requesters client application, for use with string conditions (see String Conditions (p. 455))
- `aws:EpochTime`–Number of seconds since epoch.
- `aws:Referer`–Same as the HTTP *`referer`* field.

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

Note
If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

Each AWS service that uses the access policy language might also provide service-specific keys. For a list of service-specific keys you can use, see Special Information for Amazon S3 Policies (p. 459).

Amazon S3 also has action-specific policy keys. They're grouped by resource type and applicable action in the following tables. Some policy keys are applicable to more than one resource type or action.

Important
IAM cannot evaluate a policy for validity within Amazon S3. If you specify an invalid key/action combination in the policy, IAM doesn't throw an error when you upload the policy to IAM. Also, you will not receive an error message from Amazon S3. Amazon S3 can determine only that the policy doesn't apply because it cannot fulfill the conditions. However, if you use a policy condition in an unexpected way (for example, you use a string field as a numeric comparison), Amazon S3 will throw an exception on the request and access will be denied.

Unless otherwise noted, each key is for use with the access policy language's string conditions (for more information, see String Conditions (p. 455)).

Bucket Keys in Amazon S3 Policies

The following table shows the keys related to buckets that can be in Amazon S3 policies.

Action	Applicable Keys	Description
`s3:CreateBucket`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:LocationConstraint`	Specifies the Region where the bucket will be created. Valid values are `us-west-1` (for Northern California) or `EU` (for Ireland). Do not specify a value for US Standard. Example value: `us-west-1`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Bucket. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).

Action	Applicable Keys	Description
`s3:ListBucket`	`s3:prefix`	Limits the response to objects that begin with the specified prefix. Use this to allow or deny access to objects that begin with the prefix. Example value: `home`
	`s3:delimiter`	The character you use to group objects. Example value: `/`
	`s3:max-keys`	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions (for more information, see Numeric Conditions (p. 455)). Example value: `100`
`s3:ListBucketVersions`	`s3:prefix`	Header that lets you limit the response to include only keys that begin with the specified prefix. Example value: `home`
	`s3:delimiter`	The character you use to group objects. Example value: `/`
	`s3:max-keys`	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions (for more information, see Numeric Conditions (p. 455)). Example value: `100`

Action	Applicable Keys	Description
`s3:PutBucketAcl`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Bucket acl. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).

Object Keys in Amazon S3 Policies

The following list shows the keys related to objects that can be in Amazon S3 policies.

Action	Applicable Keys	Description
`s3:PutObject`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-copy-source`	The header that specifies the name of the source bucket and key name of the source object, separated by a slash (/). Used when copying an object. Example value: `/bucketname/keyname`
	`s3:x-amz-grant-read, s3:x-amz-grant-write, s3:x-amz-grant-read-acp, s3:x-amz-grant-write-acp , s3:x-amz-grant-full -control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Object. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).
	`s3:x-amz-server-side -encryption`	Allow the specific action only if `x-amz-server-side-encryption` header is present in the request and its value matches the specified condition. with Valid values: AES256 Example value: AES256
	`s3:x-amz-metadata-di rective`	

Action	Applicable Keys	Description
		The header that specifies whether the metadata is copied from the source object or replaced with metadata provided in the request. If copied, the metadata, except for the version ID, remains unchanged. Otherwise, all original metadata is replaced by the metadata you specify. Used when copying an object. Valid values: `COPY` | `REPLACE`. The default is `COPY`. Example value: `REPLACE`
`s3:PutObjectAcl`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Object acl. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).
`s3:GetObjectVersion`	`s3:VersionId`	The version ID of the object being retrieved. Example value: `Upfdndhfd8438MNFDN93 jdnJFkdmqnh893`
`s3:GetObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being retrieved. Example value: `Upfdndhfd8438MNFDN93 jdnJFkdmqnh893`

Action	Applicable Keys	Description
`s3:PutObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being PUT. Example value: `Upfdndhfd8438MNFDN93 jdnJFkdmqnh893`
	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read, s3:x-amz-grant-write, s3:x-amz-grant-read-acp, s3:x-amz-grant-write-acp, s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers headers with specific values. For more information about the API and the request headers, go to PUT Object acl. For an example policy that uses these condition keys, see Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control (p. 313).
`s3:DeleteObjectVersion`	`s3:VersionId`	The version ID of the object being deleted. Example value: `Upfdndhfd8438MNFDN93 jdnJFkdmqnh893`

Using ACLs

Topics

Access Control List (ACL) Overview

Topics

Amazon S3 Access Control Lists (ACL) 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 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 as shown in the following sample bucket ACL (the default object ACL has the same structure).

```
<?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 identifying the owner via the AWS account's canonical user ID. 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, 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 by the email address or the canonical user ID. However, if you provide an email in your grant request, Amazon S3 finds the canonical user ID for that account and adds it to the ACL. The resulting ACLs will always contain the canonical user ID for the AWS account, not the AWS account's email address.

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 Amazon AWS accounts. Access permission to this group allows any Amazon AWS account to access the resource. However, all requests must be signed (authenticated).
- **All Users group—**Represented by `http://acs.amazonaws.com/groups/global/AllUsers`.
  Access permission to this group allows anyone to access the resource. The requests can be signed (authenticated) or unsigned (anonymous). Unsigned requests omit the Authentication header in the request.
- **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 Server Access Logging (p. 420)) 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.

Note
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, go to Enabling Cross-Account Access in Using Identity and Access Management.

Finding a Canonical User ID

The canonical user ID is associated with your AWS account. You can find this ID as follows:

To find the canonical user ID

1. Go to Security Credentials.
2. If you are not already logged in, you are prompted to sign in to **Amazon Web Services**.
3. Go to the **Account Identifier** section for the canonical user ID associated with your AWS account.

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 a permission, there is a grant entry in the ACL with the AWS account's canonical user ID.

What Permissions Can I Grant?

The following table lists the set of permissions Amazon S3 supports in an ACL. The table lists the permission and describes what they mean in the context of object and bucket permissions.

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
`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

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.

```
<?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 a 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 `FULL_CONTROL`. No one else has access rights (default).
`public-read`	Bucket and object	Owner gets `FULL_CONTROL`. The `AllUsers` group (see Who Is a Grantee? (p. 324)) gets `READ` access.
`public-read-write`	Bucket and object	Owner gets `FULL_CONTROL`. The `AllUsers` group gets `READ` and `WRITE` access. Granting this on a bucket is generally not recommended.
`authenticated-read`	Bucket and object	Owner gets `FULL_CONTROL`. The `AuthenticatedUsers` group gets `READ` access.
`bucket-owner-read`	Object	Object owner gets `FULL_CONTROL`. Bucket owner gets `READ` 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 `FULL_CONTROL` over the object. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.
`log-delivery-write`	Bucket	The `LogDelivery` group gets `WRITE` and `READ_ACP` permissions on the bucket. For more information on logs, see (Server Access Logging (p. 420)).

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 API provide you with 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 a existing resource, you can set the ACL either in the request header or in the body.

For more information, see Managing ACLs (p. 327).

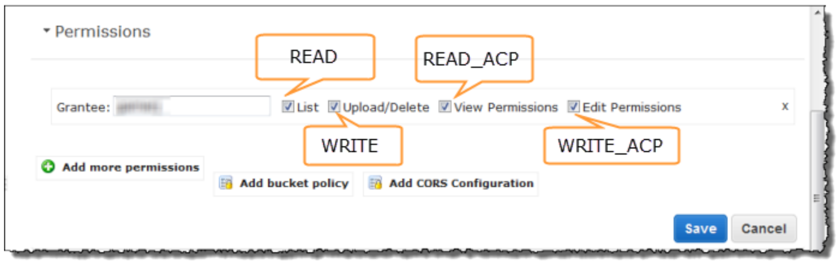
Managing ACLs

Topics

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 use 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. The **Properties** pane includes the following **Permissions** tab.



The tab shows the ACL for your selected resource. That is, it shows the list of grants found in the resource ACL. For each grant, it shows the grantee and a set of check boxes showing the permissions granted. The permission names in the console are different than the ACL permission names (see What Permissions Can I Grant? (p. 324)). The preceding illustration shows the mapping between the two.

The preceding illustration shows a grantee with `FULL_CONTROL` permissions; note that all the check boxes are selected. All the UI components shown, except the **Add bucket policy** link, relate to the ACL-based permissions. The UI allows you to add or remove permissions. To add permissions, click **Add more permissions**, and to delete a permission, highlight the line and click **X** to the right of it. When you are done updating permissions, click Save to update the ACL. The console sends the necessary request to Amazon S3 to update the ACL on the specific resource.

Managing ACLs Using the AWS SDK for Java

Setting an ACL when Creating a Resource

When creating a resource (buckets and objects), you can grant permisions (see Access Control List (ACL) Overview (p. 323)) by adding an `AccessControlList` in your request. For each permission, you explicitly specify the grantee and the permission.

For example, the following Java code snippet sends a `PutObject` request to upload an object. In the request, the code snippet specifies permissions to two AWS accounts and the Amazon S3 `AllUsers` group. The `PutObject` call includes the object data in the request body and the ACL grants in the request headers (see PUT Object).

```
String bucketName     = "bucket-name";
String keyName        = "object-key";
String uploadFileName = "file-name";

AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
         S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

AccessControlList acl = new AccessControlList();
acl.grantPermission(new Canonic
alGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),
 Permission.ReadAcp);
acl.grantPermission(GroupGrantee.AllUsers, Permission.Read);
acl.grantPermission(new EmailAddressGrantee("user@email.com"), Permis
sion.WriteAcp);

File file = new File(uploadFileName);
s3client.putObject(new PutObjectRequest(bucketName, keyName, file).withAccessCon
trolList(acl));
```

For more information about uploading objects, see Working with Amazon S3 Objects (p. 100).

In the preceding code snippet, in granting each permission you explicitly identified a grantee and a permission. Alternatively, you can specify a canned (predefined) ACL (see Canned ACL (p. 326)) in your request when creating a resource. The following Java code snippet creates a bucket and specifies a `LogDeliveryWrite` canned ACL in the request to grant write permission to the Amazon S3 `LogDelivery` group.

```
String bucketName     = "bucket-name";
AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
         S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

s3client.createBucket(new CreateBucketRequest (bucketName).withCannedAcl(CannedAc
cessControlList.LogDeliveryWrite));
```

For information about the underlying REST API, go to PUT Bucket.

Updating ACL on an Existing Resource

You can set ACL on an existing object or a bucket. You create an instance of the `AccessControlList` class and grant permissions and call the appropriate set ACL method. The following Java code snippet calls the `setObjectAcl` method to set ACL on an existing object.

```
String bucketName     = "bucket-name";
String keyName        = "object-key";

AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
         S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

AccessControlList acl = new AccessControlList();
acl.grantPermission(new Canonic
alGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),
 Permission.ReadAcp);
acl.grantPermission(GroupGrantee.AuthenticatedUsers, Permission.Read);
acl.grantPermission(new EmailAddressGrantee("user@email.com"), Permis
sion.WriteAcp);
Owner owner = new Owner();
owner.setId("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beExample");
owner.setDisplayName("display-name");
acl.setOwner(owner);

s3client.setObjectAcl(bucketName, keyName, acl);
```

Note
In the preceding code snippet, you can optionally read an existing ACL first, by calling the `getObjectAcl` method, add new grants to it, and then set the revised ACL on the resource.

Instead of granting permissions by explicitly specifying grantees and permissions explicitly, you can also specify a canned ACL in your request. The following Java code snippet sets the ACL on an existing object. In the request, the snippet specifies the canned ACL `AuthenticatedRead` to grant read access to the Amazon S3 `Authenticated Users` group.

```
String bucketName     = "bucket-name";
String keyName        = "object-key";

AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
         S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

s3client.setObjectAcl(bucketName, keyName, CannedAccessControlList.Authentic
atedRead);
```

An Example

The following Java code example first creates a bucket. In the create request, it specifies a `public-read` canned ACL. It then retrieves the ACL in an `AccessControlList` instance, clears grants, and adds new grants to the `AccessControlList`. Finally, it saves the updated `AccessControlList`, that is, it replaces the bucket ACL subresource.

The following Java code example performs the following tasks:

- Create a bucket. In the request, it specifies a `log-delivery-write` canned ACL, granting write permission to the `LogDelivery` Amazon S3 group.
- Read the ACL on the bucket.

- Clear existing permissions and add the new permission to the ACL.
- Call `setBucketAcl` to add the new ACL to the bucket.

Note

To test the following code example, you must update the code and provide your credentials, and also provide the canonical user ID and email address of the accounts that you want to grant permissions to.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.CanonicalGrantee;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
import com.amazonaws.services.s3.model.Grant;
import com.amazonaws.services.s3.model.GroupGrantee;
import com.amazonaws.services.s3.model.Permission;
import com.amazonaws.services.s3.model.Region;


public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));

        Collection<Grant> grantCollection = new ArrayList<Grant>();
        try {
            // 1. Create bucket with Canned ACL.
            CreateBucketRequest createBucketRequest =
             new CreateBucketRequest(bucketName, Region.US_Standard).with
CannedAcl(CannedAccessControlList.LogDeliveryWrite);

            Bucket resp = s3Client.createBucket(createBucketRequest);

            // 2. Update ACL on the existing bucket.
            AccessControlList bucketAcl = s3Client.getBucketAcl(bucketName);


            // (Optional) delete all grants.
            bucketAcl.getGrants().clear();

            // Add grant - owner.
            Grant grant0 = new Grant(
              new Canonic
alGrantee("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beExample"),
              Permission.FullControl);
            grantCollection.add(grant0);

            // Add grant using canonical user id.
            Grant grant1 = new Grant(
              new Canonic
alGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),

              Permission.Write);
            grantCollection.add(grant1);

            // Grant LogDelivery group permission to write to the bucket.
            Grant grant3 = new Grant(GroupGrantee.LogDelivery,
                               Permission.Write);
            grantCollection.add(grant3);

           bucketAcl.getGrants().addAll(grantCollection);

            // Save (replace) ACL.
            s3Client.setBucketAcl(bucketName, bucketAcl);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
              " means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means"+

              " the client encountered " +
                    "a serious internal problem while trying to " +
                    "communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Managing ACLs Using the AWS SDK for .NET

Setting ACL when Creating a Resource

When creating a resource (buckets and objects), you can grant permissions by specifying a collection of Grants (see Access Control List (ACL) Overview (p. 323)) in your request. For each Grant, you create an `S3Grant` object explicitly specifying the grantee and the permission.

For example, the following C# code snippet sends a `PutObject` request to upload an object. In the request, the code snippet specifies permissions to two AWS account and the Amazon S3 `AllUsers` group. The `PutObject` call includes the object data in the request body and the ACL grants in the request headers (see PUT Object).

```
AmazonS3 client;
string bucketName;
string keyName;

var grantee1 = new S3Grantee().WithCanonic
alUser("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example",
"display-name");
var grantee2 = new S3Grantee().WithEmailAddress("user@email.com");
var grantee3 = new S3Grantee().WithURI("http://acs.amazonaws.com/groups/glob
al/AllUsers");

S3Grant grant1 = new S3Grant().WithGrantee(grantee1).WithPermission(S3Permis
sion.READ_ACP);
S3Grant grant2 = new S3Grant().WithGrantee(grantee2).WithPermission(S3Permis
sion.WRITE_ACP);
S3Grant grant3 = new S3Grant().WithGrantee(grantee3).WithPermission(S3Permis
sion.READ);

// 1. Simple object put.
PutObjectRequest request = new PutObjectRequest();
request.WithContentBody("Object data for simple put.")
    .WithBucketName(bucketName)
    .WithKey(keyName)
    .WithGrants(grant1, grant2, grant3);

S3Response response = client.PutObject(request);
```

For more information about uploading objects, see Working with Amazon S3 Objects (p. 100).

In the preceding code snippet, for each `S3Grant` you explicitly identify a grantee and permission. Alternatively, you can specify a canned (predefined) ACL (see Canned ACL (p. 326)) in your request when creating a resource. The following C# code snippet creates an object and specifies an `AuthenticatedRead` canned ACL in the request to grant read permission to the Amazon S3 `AuthenticatedUsers` group.

```
AmazonS3 client;
string bucketName;

PutObjectRequest request = new PutObjectRequest();
request.WithBucketName(bucketName)
       .WithCannedACL(S3CannedACL.AuthenticatedRead);

client.PutObject(request).Dispose();
```

For information about the underlying REST API, go to PUT Bucket.

Updating ACL on an Existing Resource

You can set an ACL on an existing object or a bucket by calling the `AmazonS3.SetAcl` method. You create an instance of the `S3AccessControlList` class with a list of ACL grants and include the list in the `SetAcl` request. The following C# code snippet sends a `SetACL` request to set ACL on an existing object.

```
var grantee1 = new S3Grantee().WithCanonic
alUser("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example",
"display-name");
```

```
var grantee2 = new S3Grantee().WithEmailAddress("user@email.com");
var grantee3 = new S3Grantee().WithURI("http://acs.amazonaws.com/groups/glob
al/AllUsers");

S3Grant grant1 = new S3Grant().WithGrantee(grantee1).WithPermission(S3Permis
sion.READ_ACP);
S3Grant grant2 = new S3Grant().WithGrantee(grantee2).WithPermission(S3Permis
sion.WRITE_ACP);
S3Grant grant3 = new S3Grant().WithGrantee(grantee3).WithPermission(S3Permis
sion.READ);

S3AccessControlList aclList = new S3AccessControlList();
aclList.Grants.Add(grant1);
aclList.Grants.Add(grant2);
aclList.Grants.Add(grant3);
Owner aclOwner = new Owner()
  .WithId("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beexample")
  .WithDisplayName("owner-display-name");
aclList.Owner = aclOwner;

SetACLRequest req = new SetACLRequest();
req.WithBucketName(bucketName)
   .WithKey(keyName)
   .WithCannedACL(S3CannedACL.AuthenticatedRead);

SetACLResponse setACLResponse = client.SetACL(req);
```

Note
In the preceding code snippet, you can optionally read an existing ACL first, using the `AmazonS3.GetAcl` method, add new grants to it, and then set the revised ACL on the resource.

Instead of creating `S3Grant` objects and specifying grantee and permission explicitly, you can also specify a canned ACL in your request. The following C# code snippet sets an ACL on an existing object. In the request, the snippet specifies an `AuthenticatedRead` canned ACL to grant read access to the Amazon S3 `Autenticated Users` group.

```
AmazonS3 client;
string bucketName;
string keyName;

SetACLRequest req = new SetACLRequest();
req.WithBucketName(bucketName)
   .WithKey(keyName)
   .WithCannedACL(S3CannedACL.AuthenticatedRead);

SetACLResponse setACLResponse = client.SetACL(req);
```

An Example

The following C# code example performs the following tasks:

- Create a bucket. In the request, it specifies a `log-delivery-write` canned ACL, granting write permission to the `LogDelivery` Amazon S3 group.
- Read the ACL on the bucket.
- Clear existing permissions and add new the permission to the ACL.
- Call `SetACL` request to add the new ACL to the bucket.

```
using System;
using System.Collections.Specialized;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;

namespace s3.amazon.com.docsamples
{
  class S3Sample
  {
    static string bucketName = "*** Provide bucket name ***";
    static AmazonS3 client;

    public static void Main(string[] args)
    {
      PutBucketRequest request = new PutBucketRequest();

      NameValueCollection appConfig = ConfigurationManager.AppSettings;

      string accessKeyID = appConfig["AWSAccessKey"];
      string secretAccessKeyID = appConfig["AWSSecretKey"];
      try
      {
        using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
            accessKeyID, secretAccessKeyID))
        {
          // Add bucket (specify canned ACL).
          AddBucketWithCannedACL();

          // Get ACL on a bucket.
          GetBucketACL(bucketName);

          // Add (replace) ACL on a bucket.
          AddACLToExistingBucket();
        }
      }
      catch (AmazonS3Exception amazonS3Exception)
      {
        if (amazonS3Exception.ErrorCode != null &&
            (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
            ||
            amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
        {
          Console.WriteLine("Check the provided AWS Credentials.");
          Console.WriteLine(
              "For service sign up go to http://aws.amazon.com/s3");
        }
        else
        {
          Console.WriteLine(
              "Error occurred. Message:'{0}' when writing an object"
              , amazonS3Exception.Message);
        }
      }
      catch (Exception e)
      {
        Console.WriteLine(e.Message);
      }
      Console.WriteLine("Press any key to continue...");
      Console.ReadKey();
    }

    static void AddBucketWithCannedACL()
    {
       PutBucketRequest request = new PutBucketRequest();
      request.WithBucketName(bucketName)
          .WithBucketRegion(S3Region.US);
      // Add canned acl.
      request.AddHeaders(AmazonS3Util.CreateHeaderEntry(
                                 "x-amz-acl", "log-delivery-write"));
      client.PutBucket(request).Dispose();
    }

    static void GetBucketACL(string bucketName)
    {
      GetACLRequest request = new GetACLRequest();
      request.WithBucketName(bucketName);

      GetACLResponse response = client.GetACL(request);
      S3AccessControlList accessControlList =
                            response.AccessControlList;
      response.Dispose();
    }

    static void AddACLToExistingBucket()
    {
       GetACLRequest getRequest = new GetACLRequest();
      getRequest.BucketName = bucketName;

      GetACLResponse getResponse = client.GetACL(getRequest);
      S3AccessControlList acl = getResponse.AccessControlList;
      getResponse.Dispose();

      // Clear existing grants.
      acl.Grants.Clear();

      // Add grants. First, reset owner's full permission
      // (previous clear statement removed all permissions).
      S3Grantee grantee0 = new S3Grantee();
      grantee0.WithCanonicalUser(acl.Owner.Id, acl.Owner.DisplayName);
      acl.AddGrant(grantee0, S3Permission.FULL_CONTROL);

      // Grant permission using email.
      S3Grantee grantee1 = new S3Grantee();
      grantee1.EmailAddress = "user@email.com";
      acl.AddGrant(grantee1, S3Permission.WRITE_ACP);

      // Grant permission using Canonical ID.
      S3Grantee grantee2 = new S3Grantee();
      Amazon.S3.Model.Tuple<string, string> t =
          new Amazon.S3.Model.Tuple<string, string>
              ("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Ex
ample", "display-name");
      grantee2.CanonicalUser = t;
      acl.AddGrant(grantee2, S3Permission.WRITE);
      // Grant permission to the LogDelivery group.
      S3Grantee grantee3 = new S3Grantee();
      grantee3.URI = "http://acs.amazonaws.com/groups/s3/LogDelivery";
      acl.AddGrant(grantee3, S3Permission.WRITE);

      // Now update the ACL.
      SetACLRequest request = new SetACLRequest();
      request.BucketName = bucketName;
      request.ACL = acl;

      SetACLResponse response = client.SetACL(request);
      response.Dispose();

      // Get and print the updated ACL XML.
      Console.WriteLine(client.GetACL(new GetACLRequest()
          .WithBucketName(bucketName)).ResponseXml);
    }
  }
}
```

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

When to Use ACLs vs. Bucket Policies

ACLs provide a coarse-grain permission model, where you simply grant access permissions to buckets or objects. Bucket policies, on the other hand, provide fine-grain control over the permissions you are granting. For example, you can write a policy granting users access to a bucket or an object, provided the user sends the request from a specific IP address, or the request arrives after a specific date and time. Depending on your needs, you can use one or both of these permission models. However, there are specific use cases where ACL's may be the most appropriate:

- **There is only a Bucket Policy (no Object Policy)—**There are times when you will need to grant a wide variety of permissions on each object in your bucket. For example, if you grant write permission on your bucket, others can add objects to your bucket, which you don't have permission to access. These new object owners must explicitly grant permissions on these objects before enabling others to access them.
- **Bucket Policies are Limited to 20 Kilobytes in Size—**If you have a large number of objects and users, your bucket policy could reach the 20K size limit. In this case, you should consider using ACLs for additional grants.

Amazon S3 supports both ACLs and bucket policies. If you already use ACLs, there is no need to change. In simpler scenarios, ACLs might provide the appropriate level of permissions for your use case. For example, when granting permissions to a smaller number of grantees, using ACLs might be adequate.

Using ACLs and Bucket Policies Together

When you have ACLs and bucket policies assigned to buckets, Amazon S3 evaluates the existing Amazon S3 ACLs as well as the bucket policy when determining an account's access permissions to an Amazon S3 resource. If an account has access to resources that an ACL or policy specifies, they are able to access the requested resource.

With existing Amazon S3 ACLs, a grant always provides access to a bucket or object. When using policies, a deny always overrides a grant.

Note
Bucket policies have their own set of rules when prioritizing grants and denies. For more information, see Evaluation Logic (p. 446).

You can migrate existing Amazon S3 ACLs to policies.

To migrate ACLs to bucket policies

1	Associate a policy with a specific user, group, or bucket.
2	Add a grant to the policy for each Amazon S3 resource that the user or group has been granted access to in the ACL

Once completed, you can begin to manage your account permissions with policies instead of ACLs.

Relationship Between Actions and Permissions

Policies can allow or deny certain actions. ACLs can grant certain permissions. The actions allowed or denied by policies are a superset of the permissions that you can grant by ACLs. The relationship between actions and permissions is summarized in the following sections.

Object ACL Permissions

- **READ**—Granting *READ* permission in an object ACL allows the *s3:GetObject*, *s3:GetObjectVersion*, and *s3:GetObjectTorrent* actions to be performed on that object.
- **READ_ACP**—Granting *READ_ACP* permission in an object ACL allows the *s3:GetObjectAcl* and *s3:GetObjectVersionAcl* actions to be performed on that object.
- **WRITE_ACP**—Granting *WRITE_ACP* permission in an object ACL allows the *s3:PutObjectAcl* and *s3:PutObjectVersionAcl* actions to be performed on that object.
- **FULL_CONTROL**—Granting *FULL_CONTROL* permission in an object ACL is equivalent to granting *READ*, *READ_ACP*, and *WRITE_ACP* permission.

Bucket ACL Permissions

- **READ**—Granting *READ* permission in a bucket ACL allows the *s3:ListBucket*, *s3:ListBucketVersions*, and *s3:ListBucketMultipartUploads* actions to be performed on that bucket.
- **WRITE**—Granting *WRITE* permission in a bucket ACL allows the *s3:PutObject* and *s3:DeleteObject* actions to be performed on any object in that bucket. In addition, when the grantee is the bucket owner,

granting *WRITE* permission in a bucket ACL allows the *s3:DeleteObjectVersion* action to be performed on any version in that bucket.

- **READ_ACP—**Granting *READ_ACP* permission in a bucket ACL allows the *s3:GetBucketAcl* action to be performed on that bucket.
- **WRITE_ACP—**Granting *WRITE_ACP permission* in a bucket ACL allows the *s3:PutBucketAcl* action to be performed on that bucket.
- **FULL_CONTROL—**Granting *FULL_CONTROL* permission in a bucket ACL is equivalent to granting *READ*, *WRITE*, *READ_ACP*, and *WRITE_ACP* permission.

For more information on the actions that can be allowed or denied by policies, see Writing Bucket Policies (p. 306).

Using Query String Authentication

Query string authentication is useful for giving HTTP or browser access to resources that would normally require authentication. The signature in the query string secures the request. Query string authentication requests require an expiration date. You can specify any future expiration time in epoch or UNIX time (number of seconds since January 1, 1970).

Using Query String Authentication

1	Create a query.
2	Specify an expiration time for the query.
3	Sign it with your signature.
4	Place the data in an HTTP request.
5	Distribute the request to a user or embed the request in a web page

For example, a query URL is similar to the following example.

```
http://quotes.s3.amazonaws.com/nelson?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Ex
pires=1177363698&Signature=vjSAMPLENmGa%2ByT272YEAiv4%3D
```

For information on how to sign requests, see Query String Request Authentication Alternative (p. 57)

Data Protection

Topics

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 ensure data durability, Amazon S3 `PUT` and `PUT Object copy` operations synchronously store your data across multiple facilities before returning `SUCCESS`. Once stored, Amazon S3 maintains the durability of your objects by quickly detecting and repairing any lost redundancy.

Amazon S3 also regularly verifies the integrity of data stored using checksums. If Amazon S3 detects data corruption, it is repaired using redundant data. In addition, Amazon S3 calculates checksums on all network traffic to detect corruption of data packets when storing or retrieving data.

Amazon S3's standard storage is:

- 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 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.

Using Data Encryption

Topics

Encryption provides added security for your object data stored in your buckets in Amazon S3. You can encrypt data on your client-side and upload the encrypted data to Amazon S3. In this case, you manage encryption process, the encryption keys, and related tools. Optionally, you might want to use the server-side encryption feature in which Amazon S3 encrypts your object data before saving it on disks in its data centers and decrypts it when you download the objects, freeing you from the tasks of managing encryption, encryption keys, and related tools.

Using Server-Side Encryption

Topics

Server-side encryption is about data encryption at rest, that is, Amazon S3 encrypts your data 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. Amazon S3 manages encryption and decryption for you. For example, if you share your objects using a pre-signed URL, the pre-signed URL works the same way for both encrypted and unencrypted objects.

In client-side encryption, you manage encryption/decryption of your data, the encryption keys, and related tools. Server-side encryption is an alternative to client-side encryption in which Amazon S3 manages the encryption of you data freeing you from the tasks of managing encryption and encryption keys.

Amazon S3 Server Side Encryption employs strong multi-factor encryption. Amazon S3 encrypts each object with a unique key. As an additional safeguard, it encrypts the key itself with a master key that it regularly rotate. Amazon S3 Server Side Encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256), to encrypt your data.

You can specify data encryption at the object level. When you upload an object, you can explicitly specify in your request if you want Amazon S3 to save your object data encrypted. Server-side encryption is optional. Your bucket might contain both encrypted and unencrypted objects. Amazon S3 supports bucket policy 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.

```
{
   "Version":"2008-10-17",
   "Id":"PutObjPolicy",
   "Statement":[{
         "Sid":"DenyUnEncryptedObjectUploads",
         "Effect":"Deny",
         "Principal":{
            "AWS":"*"
         },
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::YourBucket/*",
         "Condition":{
            "StringNotEquals":{
               "s3:x-amz-server-side-encryption":"AES256"
            }
         }
      }
   ]
}
```

Server-side encryption encrypts only the object data. Any object metadata is not encrypted.

API Support for Server-Side Encryption

The object creation REST APIs (see Specifying Server-Side Encryption Using REST API (p. 347)) provide a request header, `x-amz-server-side-encryption` that you can use to request server-side encryption. The AWS SDKs also provide wrapper APIs for you 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 using the AWS SDK for Java to upload an object, you can use the `ObjectMetadata` property of the `PutObjectRequest` to set the `x-amz-server-side-encryption` request header (see Specifying Server-Side Encryption Using REST API (p. 347)). When you call the `PutObject` method of the AmazonS3 client as shown in the following Java code sample Amazon S3 encrypts and saves the data.

```
File file = new File(uploadFileName);
PutObjectRequest putRequest = new PutObjectRequest(
                                      bucketName, keyName, file);

// Request server-side encryption.
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setServerSideEncryption(
                     ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);
putRequest.setMetadata(objectMetadata);

PutObjectResult response = s3client.putObject(putRequest);
System.out.println("Uploaded object encryption status is " +
                  response.getServerSideEncryption());
```

In response, Amazon S3 returns the encryption algorithm used for encrypting your object data, which you can check using the `getServerSideEncryption` method.

For a working sample that shows how to upload an object, see Upload an Object Using the AWS SDK for Java (p. 157). For server-side encryption, add the `ObjectMetadata` property to your request.

When uploading large objects using multipart upload API, you can request server-side encryption for the object that you are uploading.

- When using the low-level multipart upload API (see Upload a File (p. 173)) to upload a large object, you can specify server-side encryption when you initiate the multipart upload. That is, you add the `ObjectMetadata` property by calling the `InitiateMultipartUploadRequest.setObjectMetadata` method.
- When using the high-level multipart upload API (see Using the High-Level Java API for Multipart Upload (p. 168)), the `TransferManager` class provides methods to upload objects. You can call any of the upload methods that take `ObjectMetadata` as a parameter.

Determining Encryption Algorithm Used

To determine the encryption state of an existing object you can retrieve the object metadata as shown in the following Java code sample.

```
GetObjectMetadataRequest request2 =
                new GetObjectMetadataRequest(bucketName, keyName);

ObjectMetadata metadata = s3client.getObjectMetadata(request2);

System.out.println("Encryption algorithm used: " +
            metadata.getServerSideEncryption());
```

If server-side encryption is not used for the object that is stored in Amazon S3, the method returns a null.

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, you make a copy of the object and delete the source object. Note that, by default, the copy API will not encrypt the target, unless you explicitly request server-side encryption. You can request the encryption of the target object by using the `ObjectMetadata` property to specify server-side encryption in the `CopyObjectRequest` as shown in the following Java code sample.

```
CopyObjectRequest copyObjRequest = new CopyObjectRequest(
sourceBucket, sourceKey, targetBucket, targetKey);

// Request server-side encryption.
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setServerSideEncryption(
          ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);

copyObjRequest.setNewObjectMetadata(objectMetadata);

CopyObjectResult response =  s3client.copyObject(copyObjRequest);
System.out.println("Copied object encryption status is " +
                  response.getServerSideEncryption());
```

For a working sample of how to copy an object, see Copy an Object Using the AWS SDK for Java (p. 209). You can specify server-side encryption in the `CopyObjectRequest`. object as shown in the preceding code sample.

Specifying Server-Side Encryption Using the AWS SDK for .NET

When using the AWS SDK for .NET to upload an object, you can use the `WithServerSideEncryptionMethod` property of the `PutObjectRequest` to set the `x-amz-server-side-encryption` request header (see Specifying Server-Side Encryption Using REST API (p. 347)). When you call the `PutObject` method of the AmazonS3 client as shown in the following C# code sample, Amazon S3 encrypts and saves the data.

```
static AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

PutObjectRequest request = new PutObjectRequest();
```

```
request.WithContentBody("Object data for simple put.")
    .WithBucketName(bucketName)
    .WithKey(keyName)
    .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256);

S3Response response = client.PutObject(request);

// Check the response header to determine if the object is encrypted.
ServerSideEncryptionMethod destinationObjectEncryptionStatus = response.Server
SideEncryptionMethod;
```

In response, Amazon S3 returns the encryption algorithm that is used to encrypt your object data, which you can check using the `ServerSideEncryptionMethod` property.

For a working sample of how to upload an object, see Upload an Object Using the AWS SDK for .NET (p. 158). For server-side encryption, set the `ServerSideEncryptionMethod` property by calling the `WithServerSideEncryptionMethod` method.

To upload large objects using the multipart upload API, you can specify server-side encryption for the objects that you are uploading.

- When using the low-level multipart upload API (see Using the Low-Level .NET API for Multipart Upload (p. 188)) to upload a large object, you can specify server-side encryption in your InitiateMultipartUpload request. That is, you set the `ServerSideEncryptionMethod` property to your `InitiateMultipartUploadRequest` by calling the `WithServerSideEncryptionMethod` method.
- When using the high-level multipart upload API (see Using the High-Level .NET API for Multipart Upload (p. 178)), the `TransferUtility` class provides methods (`Upload` and `UploadDirectory`) to upload objects. In this case, you can request server-side encryption using the `TransferUtilityUploadRequest` and the `TransferUtilityUploadDirectoryRequest` objects.

Determining Encryption Algorithm Used

To determine the encryption state of an existing object you can retrieve the object metadata as shown in the following C# code sample.

```
AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

ServerSideEncryptionMethod objectEncryption;

GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest()
                                               .WithBucketName(bucketName)
                                               .WithKey(keyName);

objectEncryption = client.GetObjectMetadata(metadataRequest)
                                  .ServerSideEncryptionMethod;
```

The encryption algorithm is specified with an enum. If the stored object is not encrypted (default behavior), then the `ServerSideEncryptionMethod` property of the object will default to `None`.

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, you can make a copy of the object and delete the source object. Note that by default the copy API will not encrypt the target, unless you explicitly request server-side encryption of the destination object. The following C# code sample makes a copy of an object. The request explicitly specifies server-side encryption for the destination object.

```
AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

CopyObjectResponse response = client.CopyObject(new CopyObjectRequest()
            .WithSourceBucket(sourceBucketName)
            .WithSourceKey(sourceObjetKey)
            .WithDestinationBucket(targetBucketName)
            .WithDestinationKey(targetObjectKey)
            .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256)
);
// Check the response header to determine if the object is encrypted.
ServerSideEncryptionMethod destinationObjectEncryptionStatus = response.Server
SideEncryptionMethod;
```

For a working sample of how to copy an object, see Copy an Object Using the AWS SDK for .NET (p. 210). You can specify server-side encryption in the `CopyObjectRequest`. object as shown in the preceding code sample.

Specifying Server-Side Encryption Using the AWS SDK for PHP

When using the AWS SDK for PHP to upload an object, you can use the `AmazonS3::create_object()` method to upload an object. To add the `x-amz-server-side-encryption` request header (see Specifying Server-Side Encryption Using REST API (p. 347)) to your request, specify the `array` parameter provided by this method. In the `array` parameter, set the `encryption` key with value `AES256` as shown in the following PHP code sample.

```
// Instantiate the class.
$s3 = new AmazonS3();

$response = $s3->create_object(
    $bucket,
    $keyname1,
     array('fileUpload' => $filepath,
           'encryption' => 'AES256')
    );
print_r($response);
```

In response, Amazon S3 returns the `x-amz-server-side-encryption` header with the value of the encryption algorithm used to encrypt your object data.

For a working sample of how to upload an object, see Upload an Object Using the AWS SDK for PHP (p. 162).

To upload large objects using the multipart upload API, you can specify server-side encryption for the objects that you are uploading.

- When using the low-level multipart upload API (see Using the Low-Level PHP API for Multipart Upload (p. 195)), you can specify server-side encryption when you call the `AmazonS3::initiate_multipart_upload()` method. You add the `array` parameter and set the `encryption` key with the value of the encryption algorithm `AES256`.
- When using the high-level multipart upload API (see Using the High-Level PHP API for Multipart Upload (p. 194)), `AmazonS3::create_mpu_object()`, the API does not support the specification of server-side encryption.

Determining Encryption Algorithm Used

To determine the encryption state of an existing object, retrieve the object metadata by calling the `get_object_headers` method as shown in the following PHP code sample. For the object that is encrypted on the server-side, Amazon S3 returns the `x-amz-server-side-encryption` header with the value of the encryption algorithm used.

```
$s3 = new AmazonS3();

$response = $s3->get_object_headers($bucket, $keyname);

header('Content-Type: text/plain; charset=utf-8');
print_r($response);


if ($response->isOK())
{
    echo 'Object headers retrieved!';
}
```

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. Note that by default the copy API will not encrypt the target, unless you explicitly request server-side encryption of the destination object. You must add the optional array parameter and set the encryption. The following PHP code sample makes a copy of an object. The request explicitly specifies server-side encryption for the destination object.

```
$sourcebucket = '*** Source bucket ***';
$sourcekeyname = '*** Source object key ***';

$targetbucket = '*** Target bucket ***';
$targetkeyname = '*** Target object key ***';

// Instantiate the class.
$s3 = new AmazonS3();

// Copy the object.
$response = $s3->copy_object(
        array( // Source.
           'bucket' => $sourcebucket,
           'filename' => $sourcekeyname
        ),
        array( // Target.
           'bucket' => $targetbucket,
           'filename' => $targetkeyname
           //'encryption' => 'AES256'
        ),
        array( // Optional parameters.
        'encryption' => 'AES256'
        )
);
```

For a working sample of how to copy an object, see Copy an Object Using the AWS SDK for PHP (p. 213).

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 at rest encrypted by specifying an options hash `server_side_encryption` in the `#write` instance method. When you read the object back, it is automatically decrypted.

The following Ruby script sample demonstrates how to specify that a file uploaded to Amazon S3 be encrypted at rest.

```
# Upload a file and set server-side encryption.
key_name = File.basename(file_name)
s3.buckets[bucket_name].objects[key_name].write(:file => file_name, :serv
er_side_encryption => :aes256)
```

For a working sample that shows how to upload an object, see Upload an Object Using the AWS SDK for Ruby (p. 165).

Determining Encryption Algorithm Used

To check the encryption algorithm that is used for encrypting an object data at rest, use the `#server_side_encryption` method of the `S3Object` instance. The following code sample demonstrates how to determine the encryption state of an existing object.

```
# Determine server-side encryption of an object.
enc = s3.buckets[bucket_name].objects[key_name].server_side_encryption
enc_state = (enc != nil) ? enc : "not set"
puts "Encryption of #{key_name} is #{enc_state}."
```

If server-side encryption is not used for the object that is stored in Amazon S3, the method returns a 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. The Ruby API `S3Object` class has `#copy_from` and `#copy_to` methods that you can use to copy objects. Note that, by default, the copy methods will 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 sample. The code sample demonstrates how to use the `#copy_to` method.

```
s3 = AWS::S3.new

# Upload a file and set server-side encryption.
bucket1 = s3.buckets[source_bucket]
bucket2 = s3.buckets[target_bucket]
obj1 = bucket1.objects[source_key]
obj2 = bucket2.objects[target_key]

obj1.copy_to(obj2, :server_side_encryption => :aes256)
```

For a working sample of how to copy an object, see Copy an Object Using the AWS SDK for Ruby (p. 216).

Specifying Server-Side Encryption Using 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 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

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 how to upload an object, go to Uploading Objects into Amazon S3.

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 will be encrypted. For an example of how to copy an object using the console, go to Copying an Object. The console also allows you to update properties of one or more objects. For example, you can select one or more objects and select server-side encryption

Using Client-Side Encryption

Overview

Instead of using Amazon S3's server-side encryption, you also have the option of encrypting your data before sending it to Amazon S3. You can build your own library that encrypts your objects data on the client side before uploading it to Amazon S3. Optionally, you can use the AWS SDK for Java, which you can use to automatically encrypt your data before uploading it to Amazon S3.

Currently, only the AWS SDK for Java supports client-side encryption.

Important
Your private encryption keys and your unencrypted data are never sent to AWS; therefore, it is important that you safely manage your encryption keys. If you lose your encryption keys, you won't be able to unencrypt your data.

The AWS SDK for Java uses a process called *envelope encryption*. In envelope encryption, you provide your encryption key to the Amazon S3 encryption client and the client takes care of the rest of the process. The process works like this:

1. The Amazon S3 encryption client generates a one-time-use symmetric key (called the *envelope symmetric key*) that the client uses to encrypt your data.
2. The client encrypts the envelope symmetric key using your private encryption key.
3. The client then uploads the encrypted envelope key along with your encrypted data to Amazon S3. By default, the encrypted envelope key is stored as object metadata, `x-amz-meta-x-amz-key`, in Amazon S3.

Retrieving and decrypting the client-side encrypted data from Amazon S3 is the reverse of the encryption flow above:

1. The client retrieves your encrypted data from Amazon S3 along with the encrypted envelope key.
2. The client then decrypts the encrypted envelope key using your private encryption key.
3. The client decrypts your data using the envelope key.

Your private encryption key that the client uses in the envelope encryption process can be an asymmetric key pair (composed of public and private keys) or it can be a symmetric key. We recommend using asymmetric keys for the increased level of security that they provide in the envelope encryption process.

For more information about the envelope encryption process, see the Client-Side Data Encryption with the AWS SDK for Java and Amazon S3 article.

Specifying Client-Side Encryption Using the AWS SDK for Java

To encrypt and decrypt objects client-side for upload to Amazon S3, you can use the `AmazonS3EncryptionClient` class in the AWS SDK for Java. This class provides similar functionality as the non-encrypted `AmazonS3Client` class so upgrading existing code to use the encryption client is straightforward.

The `AmazonS3EncryptionClient` class requires that you initialize it with an `EncryptionMaterials` instance which describes your encryption keys (either asymmetric or symmetric) to use.

Note
If you get a cipher encryption error message when you use the encryption API for the first time, then 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.

Uploading and Retrieving a Client-Side Encrypted Object

1	Construct a new `AWSCredentials` object by providing your security credentials.

2	Construct a new `EncryptionMaterials` object by providing asymmetric key pair.
3	Use the `AWSCredentials` object and the `EncryptionMaterials` object to create a new `AmazonS3EncryptionClient` instance.
4	Upload the object using the `putObject` method and retrieve the object using the `getObject` method. The client performs the necessary encryption and decryption.

The following Java code sample demonstrates the preceding tasks.

```
// Specify the asymmetric key pair to use.
KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance("RSA");
keyGenerator.initialize(1024, new SecureRandom());
KeyPair myKeyPair = keyGenerator.generateKeyPair();

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(myKeyPair);
AmazonS3EncryptionClient encryptedS3client =
    new AmazonS3EncryptionClient(credentials, encryptionMaterials);

// Upload the object.
encryptedS3client.putObject(new PutObjectRequest(bucketName, key, createSample
File()));

// Retrieve the object.
S3Object downloadedObject = encryptedS3client.getObject(bucketName, key);
```

Example

The following Java code example creates an asymmetric key pair and uses the key pair to upload an encrypted sample file to Amazon S3 and then retrieves the encrypted sample file. You must update the following sample and provide the correct credential information and bucket name.

Note
For demonstration, the following example creates an asymmetric key that is used for the duration the program execution. However, in a real world application, you should already have a key to provide to the client.

```
import java.io.*;
import java.security.*;
import java.security.spec.*;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3EncryptionClient;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

public class S3ClientSideEncryptionTest {
    public static void main(String[] args)
    throws IOException, NoSuchAlgorithmException, InvalidKeySpecException {

        String myAccessKeyId = "***Provide Access Key ID***";
        String mySecretKey = "***Provide Secret Key***";
        String bucketName = "***Provide bucket name***";
        String key = "test.txt";
        String encryptionAlgorithm = "RSA";

       KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance(encryption
Algorithm);
  keyGenerator.initialize(1024, new SecureRandom());
  KeyPair myKeyPair = keyGenerator.generateKeyPair();

        // Construct an instance of AmazonS3EncryptionClient.
        AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId,
mySecretKey);
        EncryptionMaterials encryptionMaterials = new EncryptionMateri
als(myKeyPair);
        AmazonS3EncryptionClient encryptedS3client =
                            new AmazonS3EncryptionClient(credentials, encryp
tionMaterials);

        // Encrypt and upload a sample file to Amazon S3.
        encryptedS3client.putObject(new PutObjectRequest(bucketName, key, cre
ateSampleFile()));

        // Retrieve the file from Amazon S3 and decrypt.
        S3Object downloadedObject = encryptedS3client.getObject(bucketName,
key);
    }

    private static File createSampleFile() throws IOException {
        File file = File.createTempFile("encryptiontest", ".txt");
        file.deleteOnExit();
        Writer writer = new OutputStreamWriter(new FileOutputStream(file));
        writer.write(new java.util.Date().toString()+"\n");
        writer.write("abcdefghijklmnopqrstuvwxyz\n");
        writer.write("01234567890112345678901234\n");
        writer.write("!@#$%^&*()-=[]{};':',.<>/?\n");
        writer.close();

        return file;
    }
}
```

Customizing Your Client-Side Encryption Environment

Client-side encryption using the AWS SDK for Java gives you several options for customizing your encryption/decryption process. This topic discusses how to configure common options and what they mean.

You can configure your client-side encryption environment by choosing the appropriate constructor of the `AmazonS3EncryptionClient` class. The constructor has several signatures and can take a combination of the following four arguments: `AWSCredentials`, `ClientConfiguration`, `EncryptionMaterials`, and `CryptoConfiguration`. The `AWSCredentials` and `ClientConfiguration` arguments are the same arguments that are used in the non-encrypted Amazon S3 client `AmazonS3Client` and are not discussed here. The remaining two arguments, `EncryptionMaterials` and `CryptoConfiguration` allow you to customize different aspects of your client-side encryption and are discussed below. For more information about the encryption client constructors, see the AWS Java SDK documentation.

Specifying Encryption Materials

To use Amazon S3 client-side encryption, you must specify a key that is used in the encryption process, for both the upload and retrieval of encrypted data. You must keep this key secure.

Important
Your private encryption keys and your unencrypted data are never sent to AWS; therefore, if you lose your encryption keys, you won't be able to unencrypt your data.

You can use the `EncryptionMaterials` class to indicate to the Amazon S3 encryption client what key you want to use for encryption. You can specify either an asymmetric or symmetric key to use; however, it is recommended that you use an asymmetric key whenever possible. An asymmetric key is a key pair that consists of a public key and a private key.

The following sample code demonstrates how to create a new symmetric key.

```
// Create a new symmetric key.
KeyGenerator symKeyGenerator = KeyGenerator.getInstance("DES");
SecretKey symKey = symKeyGenerator.generateKey();

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);

// Create an EncryptionMaterials object using the symmetric key.
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(symKey);

// Create a new encryption client.
AmazonS3EncryptionClient encryptedS3 = new AmazonS3EncryptionClient(credentials,
 encryptionMaterials);
```

Specifying Encryption Metadata Storage Location

When the Amazon S3 client (using the `AmazonS3EncryptionClient` class) encrypts data and uploads it to Amazon S3, the encrypted envelope symmetric key is also stored in S3. By default, the encrypted key is stored as user-defined object metadata. After you upload an encrypted object, you can view its properties and see the additional metadata name-value pairs related to encryption. For example, the key name **x-amz-meta-x-amz-key** and key value equal to the envelope key are set on an client-side encrypted object uploaded to Amazon S3.

Optionally, you can also choose to store encryption metadata as an *instruction* file stored at the same location as the encrypted file. The instruction file will have the same key name as the encrypted data file but with the extension ".instruction" appended. You should use an instruction file when the strength of your encryption key results in a symmetric key that is too big for the object metadata. Metadata should be less than 2 KB. Encryption metadata is either stored as object metadata or an instruction file, but not both.

The following code sample demonstrates how to store encryption metadata in an instruction file using the `CryptoConfiguration` class and the appropriate constructor of the `AmazonS3EncryptionClient` class.

```
// Load the asymmetric key pair.
KeyPair myKeyPair = S3ClientSideEncryptionTest.LoadKeyPair(pathToKey, encrytion
Algorithm);

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(myKeyPair);

// Create an instance of the CryptoConfiguration class to specify using the
instruction file.
CryptoConfiguration cryptoConfig =
    new CryptoConfiguration().withStorageMode(CryptoStorageMode.InstructionFile);

// Use the CryptoConfiguration instance in the encryption client constructor.
AmazonS3EncryptionClient encryptedS3 =
 new AmazonS3EncryptionClient(credentials, encryptionMaterials, cryptoConfig);

// Upload the object.
encryptedS3.putObject(new PutObjectRequest(bucketName, key, createSampleFile()));


// Retrieve the object.
S3Object downloadedObject = encryptedS3.getObject(bucketName, key);
```

Specifying A Crypto Provider

The default cryptography provider used in Amazon S3 client-side encryption client is the Java Cryptography Extension (JCE). Optionally, you can specify a different cryptography provider implementation to use to encrypt and decrypt data.

The following example sample code demonstrates using the Bouncy Castle provider.

```
// Create a new instance of the provider.
Provider bcProvider = new BouncyCastleProvider();

// Load the asymmetric key pair.
KeyPair myKeyPair = S3ClientSideEncryptionTest.LoadKeyPair(pathToKey, encrytion
Algorithm);

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(myKeyPair);

// Create an instance of the CryptoConfiguration class to specify a crypto
provider.
CryptoConfiguration cryptoConfig =
    new CryptoConfiguration().withCryptoProvider(bcProvider);

// Use the CryptoConfiguration instance in the encryption client constructor.
AmazonS3EncryptionClient encryptedS3 =
 new AmazonS3EncryptionClient(credentials, encryptionMaterials, cryptoConfig);

// Upload the object.
encryptedS3.putObject(new PutObjectRequest(bucketName, key, createSampleFile()));


// Retrieve the object.
S3Object downloadedObject = encryptedS3.getObject(bucketName, key);
```

Using Reduced Redundancy Storage

Topics

The Amazon S3 Reduced Redundancy Storage (RRS) option provides a less durable, highly available storage option that is designed to provide 99.99% durability of objects over a given year.

Amazon S3 stores objects according to their storage class, which Amazon S3 assigns to an object when it is written to Amazon S3. The default storage class is `STANDARD`. You can assign objects a specific storage class (`STANDARD` or `REDUCED_REDUNDANCY`) only when writing the objects or when copying objects stored in Amazon S3. RRS is specified per object, at the time the object is written."

For a general overview of this feature, see Reduced Redundancy Storage (p. 7).

Setting the Storage Class of an Object You Upload

To set the storage class of an object you upload to RRS, you set *`x-amz-storage-class`* to `REDUCED_REDUNDANCY` in a `PUT` request.

How to Set the Storage Class of an Object You're Uploading to RRS

- Create a `PUT Object` request setting the *`x-amz-storage-class`* request header to `REDUCED_REDUNDANCY`.
  You must have the correct permissions on the bucket to perform the `PUT` operation. The default value for the storage class is `STANDARD` (for regular Amazon S3 storage).

  The following example sets the storage class of `my-image.jpg` to RRS.

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: image/jpeg
Content-Length: 11434
Expect: 100-continue
x-amz-storage-class: REDUCED_REDUNDANCY
```

Changing the Storage Class of an Object in Amazon S3

Topics

You can also change the storage class of an object that is already stored in Amazon S3 by copying it to the same key name in the same bucket. To do that, you use the following request headers in a `PUT Object copy` request:

- *`x-amz-metadata-directive`* set to `COPY`
- *`x-amz-storage-class`* set to `STANDARD` or `REDUCED_REDUNDANCY`

Important
To optimize the execution of the copy request, do not change any of the other metadata in the `PUT Object copy` request. If you need to change metadata other than the storage class, set *`x-amz-metadata-directive`* to `REPLACE` for better performance.

How to Rewrite the Storage Class of an Object in Amazon S3

- Create a `PUT Object copy` request and set the *`x-amz-storage-class`* request header to `REDUCED_REDUNDANCY` (for RRS) or `STANDARD` (for regular Amazon S3 storage), and make the target name the same as the source name.

  You must have the correct permissions on the bucket to perform the copy operation.

  The following example sets the storage class of `my-image.jpg` to RRS.

```
PUT /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
x-amz-copy-source: /bucket/my-image.jpg
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
x-amz-storage-class: REDUCED_REDUNDANCY
x-amz-metadata-directive: COPY
```

  The following example sets the storage class of `my-image.jpg` to standard.

```
PUT /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
```

```
x-amz-copy-source: /bucket/my-image.jpg
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
x-amz-storage-class: STANDARD
x-amz-metadata-directive: COPY
```

Note
If you copy an RSS object and fail to include the *`x-amz-storage-class`* request header, the storage class of the target object defaults to `STANDARD`.

It is not possible to change the storage class of a specific version of an object. When you copy it, Amazon S3 gives it a new version ID.

Note
When an object is written in a copy request, the entire object is rewritten in order to apply the new storage class.

For more information about versioning, see Using Versioning (p. 355).

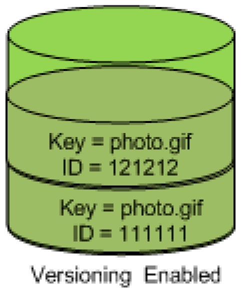
Return Code For Lost Data

If Amazon S3 detects that an object has been lost, any subsequent `GET`,or `HEAD` operations, or `PUT Object copy` operation that uses the lost object as the source object, will result in a `405 Method Not Allowed` error. Once an object is marked lost, Amazon S3 will never be able to recover the object. In this situation, you can either delete the key, or upload a copy of the object."

Using Versioning

Topics

Versioning is a means of keeping multiple variants of an object in the same bucket. 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).



You might enable versioning to prevent objects from being deleted or overwritten by mistake, or to archive objects so that you can retrieve previous versions of them.

Note
The SOAP API does not support versioning.

Enabling a Bucket's Versioning State

Topics

Buckets can be in one of three states: unversioned (the default), versioning-enabled, or versioning-suspended. Once you version enable a bucket, it can never return to an unversioned state. You can, however, suspend versioning on that bucket.

Only the bucket owner can configure the versioning state of a 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.

MFA Delete

You can add another layer of security by configuring a bucket to enable MFA (Multi-Factor Authentication) Delete. By enabling MFA Delete on your Amazon S3 bucket, you can only change the versioning state of your bucket or permanently delete an object version when you provide two forms of authentication together:

- Your AWS account credentials
- The concatenation of a valid serial number, a space, and the six-digit code displayed on an approved authentication device

To use MFA Delete, you must purchase a third-party device that automatically updates the authentication code displayed on it, as shown.

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 Bucket Policy to Require MFA Authentication (p. 311).

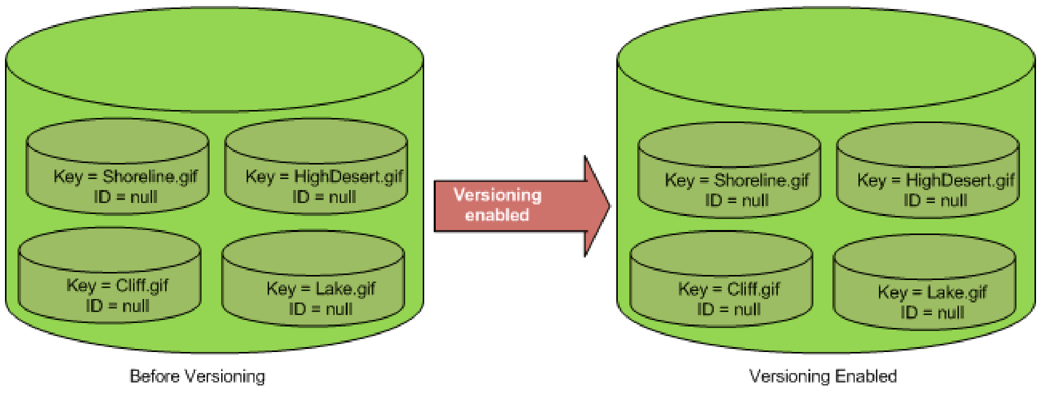
For more information on how to purchase and activate an authentication device, go to http://aws.amazon.com/mfa/.

For more information about configuring a bucket to enable MFA delete, see Configuring a Bucket with MFA Delete (p. 358).

Enabling Versioning

Topics

Objects stored in your bucket before you set the versioning state have a version ID of `null`. When you enable versioning the objects in your bucket do not change, as shown in the following figure.



What changes is how Amazon S3 handles the objects in future requests in that bucket.

To enable object versioning

1. In a `PUT Bucket` request, include the *`versioning`* sub-resource.
2. In the body of the request, use `Enabled` for the value of the *`Status`* request element.

Example Enabling Versioning

The following request enables versioning on the bucket, *bucketName*.

```
PUT /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

Configuring a Bucket with MFA Delete

You have the option of enabling MFA Delete at the same time you specify the versioning state of the bucket. Once you configure a bucket so that it is MFA Delete enabled, all future requests to change the versioning state or delete a version require the request header `x-amz-mfa:` *`[SerialNumber] [AuthenticationCode]`*. Note the space between *`[SerialNumber]`* and *`[AuthenticationCode]`*. Requests that include *`x-amz-mfa`* must use HTTPS.

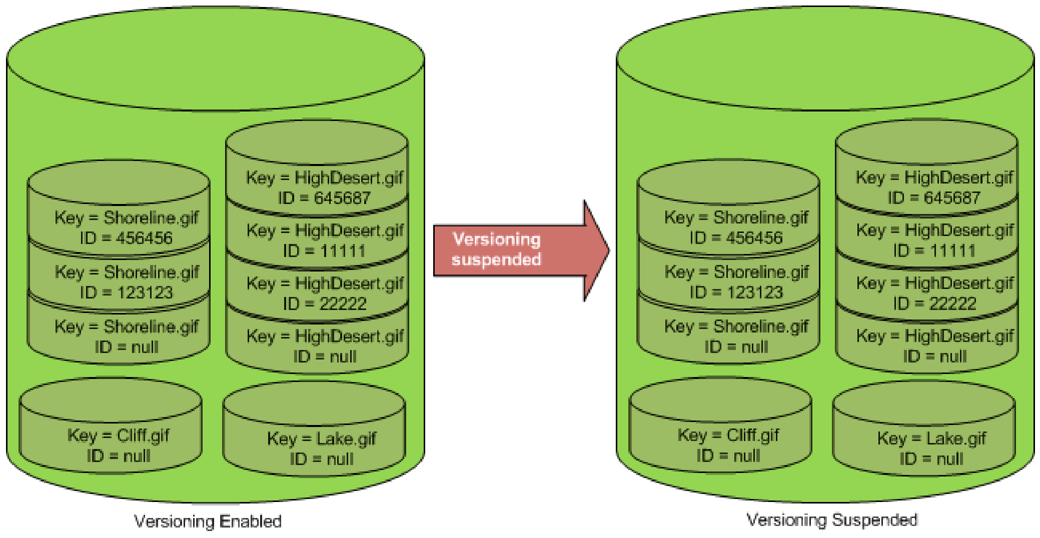
Example Enabling Versioning and MFA Delete

The following request enables versioning and MFA Delete on *bucketName*.

```
PUT /?versioning HTTPS/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124
x-amz-mfa: 20899872 301749

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
  <MfaDelete>Enabled</MfaDelete>
</VersioningConfiguration>
```

Suspending Versioning

The bucket owner can suspend versioning to stop accruing object versions. When you suspend versioning, the objects in your bucket do not change, as shown in the following figure.



What changes is how Amazon S3 handles objects in future requests. For a more detailed explanation of the effects of suspending versioning, see Working with Versioning-Suspended Buckets (p. 372).

To suspend object versioning

1. In the header of a `PUT Bucket` request, include the *`versioning`* sub-resource.
2. In the body of the request, use `Suspended` for the value of the *`Status`* request element.

Example Suspending Versioning

The following request suspends versioning on *`bucketName`*.

```
PUT /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
</VersioningConfiguration>
```

If you configured a bucket to be MFA Delete enabled, you must include the *`x-amz-mfa`* request header and the *`MfaDelete`* request element in the request to change the bucket's versioning state.

Example Suspending Versioning Using MFA Delete

The following request suspends versioning on *bucketName* that is configured with MFA Delete.

```
PUT /?versioning HTTPS/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
x-amz-mfa: 20899872 301749
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
  <MfaDelete>Enabled</MfaDelete>
</VersioningConfiguration>
```

Note that requests that include *x-amz-mfa* must use HTTPS.

Determining the Versioning State

You can use the *versioning* sub-resource to retrieve the versioning state of a bucket.

To retrieve the versioning state of a bucket

1. In the header of a `GET Bucket` request, include the *versioning* sub-resource.

```
GET /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
```

2. In the response, check the value of the response element *Status*. There are three possible results:
   - If Versioning on a bucket is enabled, the response is:

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

   - If Versioning on a bucket is suspended, the response is:

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
</VersioningConfiguration>
```

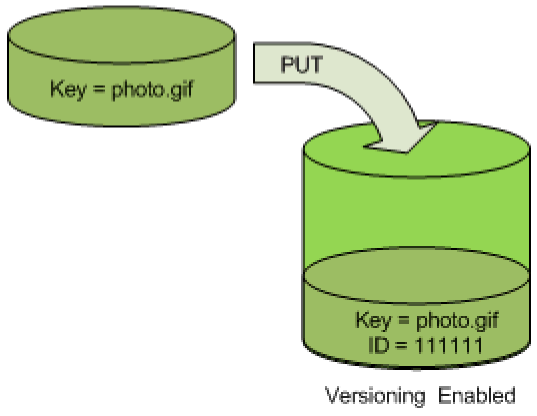
   - If Versioning on a bucket has never been enabled, the response doesn't retrieve a *status* element:

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

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.



Adding Objects to Versioning-Enabled Buckets

1	Enable versioning on a bucket using a `PUT Bucket versioning` request. For more information, go to PUT Bucket versioning.
2	Send a `PUT`, `POST`, or `COPY` 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-versionid`* 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.

Listing the Objects in a Versioning-Enabled Bucket

Topics

To list all of the versions of all of the objects in a bucket, you use the *`versions`* sub-resource in a `GET Bucket` request. Amazon S3 can only retrieve a maximum of 1000 objects, and each object version counts fully as an object. Therefore, if a bucket contains two keys (e.g. `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`* sub-resource.

```
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 Buckets

Topics

There might be times when you either want to retrieve a subset of all object versions in a bucket, or the number of object versions 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 must use the request parameters for GET Bucket. For more information, go to GET Bucket.

Retrieving All Versions of a Key

You can retrieve all versions of an object using the *`versions`* sub-resource and the *`prefix`* request parameter using the following process. For more information about *`prefix`*, go to GET Bucket.

Retrieving All Versions of a Key

1	Set the *`prefix`* parameter to the key of the object you want to retrieve.
2	Send a `GET Bucket` request using the *`versions`* sub-resource and *`prefix`*. `GET /?versions&prefix=objectName HTTP/1.1`

Example Retrieving Objects Using 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, go to GET Bucket.

Retrieving Additional Object Versions after Exceeding Max-Keys

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`*, go to GET Bucket.

Retrieving Additional Responses that Satisfy the Original GET Request

1	Set the value of *`key-marker`* to the key returned in *`NextKeyMarker`* in the previous response.
2	Set the value of *`version-id-marker`* to the version ID returned in *`NextVersionIdMarker`* in the previous response.
3	Send a `GET Bucket versions` request using *`key-marker`* and *`version-id-marker`*.

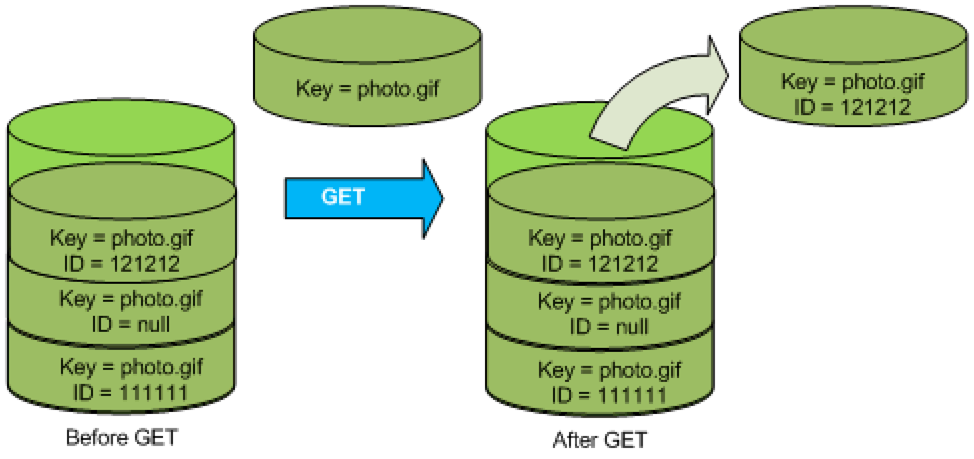
Example Retrieving Objects Starting with a Specified Key and Version ID

```
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=
```

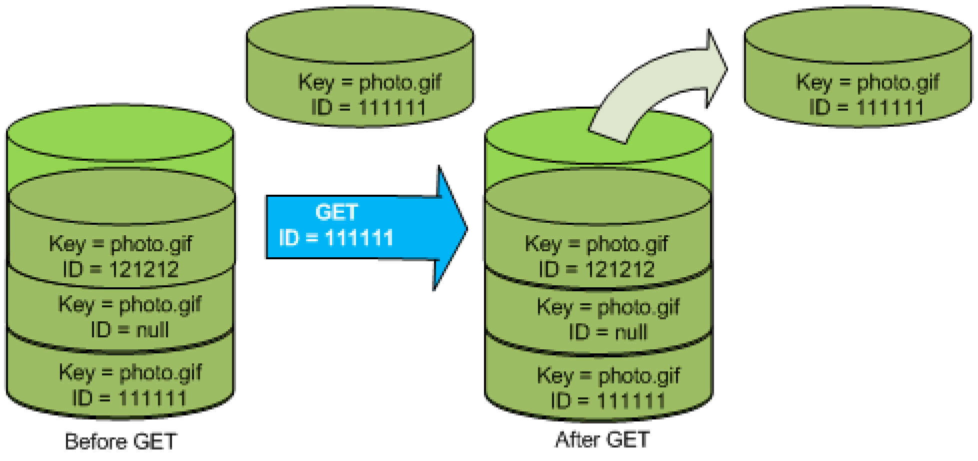
Retrieving Object Versions

Topics

A simple `GET` request retrieves the latest version of an object. The following figure shows how `GET` returns the latest 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 latest).



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=
```

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 3HL4kqCxf3vjVBH40Nrjfkd of `my-image.jpg`.

```
HEAD /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfkd 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: 3HL4kqtJlcpXroDTDmjVBH40Nrjfkd
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
```

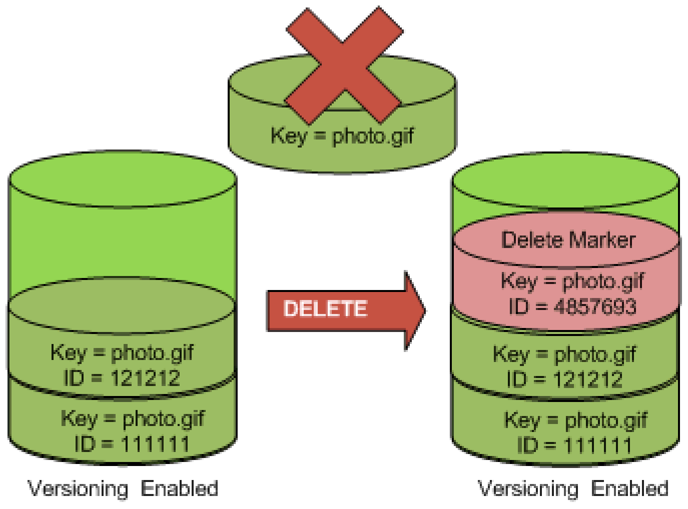
Deleting Object Versions

Topics

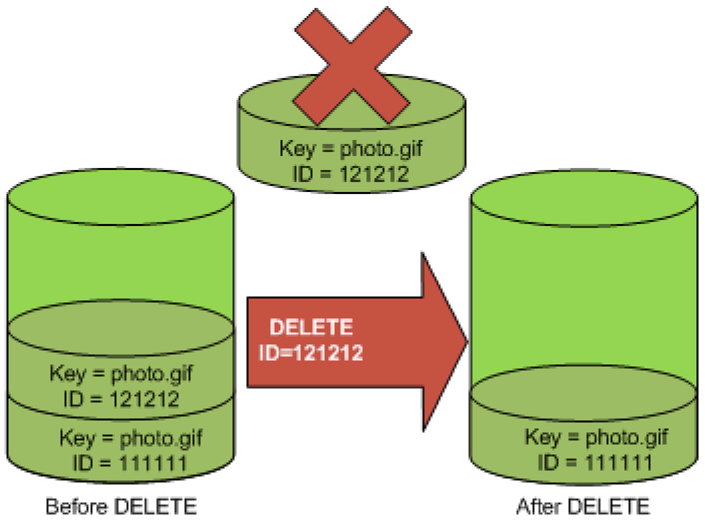
A `DELETE` request has the following use cases:

- When versioning is enabled, a simple `DELETE` cannot permanently delete an object.
  Instead, Amazon S3 inserts a delete marker in the bucket and that becomes the latest version of the object with a new ID. When you try to `GET` an object whose latest 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.
- To permanently delete versioned objects, you must use `DELETE Object versionId`.

The following figure shows that a simple `DELETE` does not actually remove the specified object. Instead, Amazon S3 inserts a delete marker.



The following figure shows that deleting a specified object version permanently removes that object.



To a delete a specific version of an object

- In a `DELETE` request 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
```

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. 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.

Requests that include *`x-amz-mfa`* must use HTTPS.

For more information about authentication devices, go to http://aws.amazon.com/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.

```
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=
```

Note the space between *[SerialNumber]* and *[AuthenticationCode]*. For more information, go to DELETE Object .

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 is like any other object (it has a key and version ID) except that it:

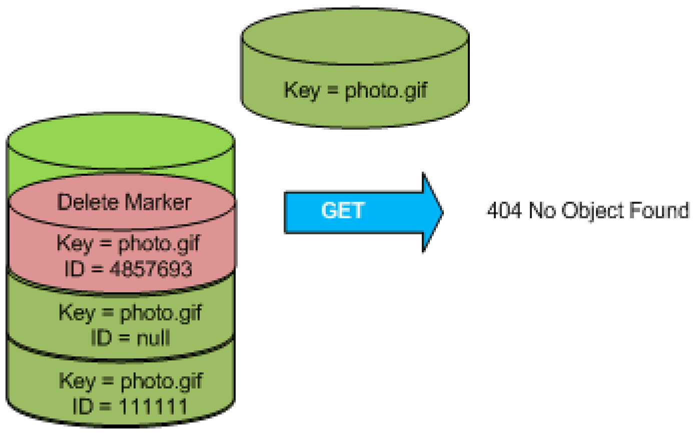
- Does not have data associated with it
- Is not associated with an access control list (ACL) value
- Does not retrieve anything from a `GET` request because it has no data; you get a 404 error
- The only operation you can use on a delete marker is `DELETE` and only the bucket owner can issue such a request

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 latest version of the object. (The object's key becomes the key of the Delete Marker.) If you try to get an object and its latest version is a delete marker, Amazon S3 responds with:

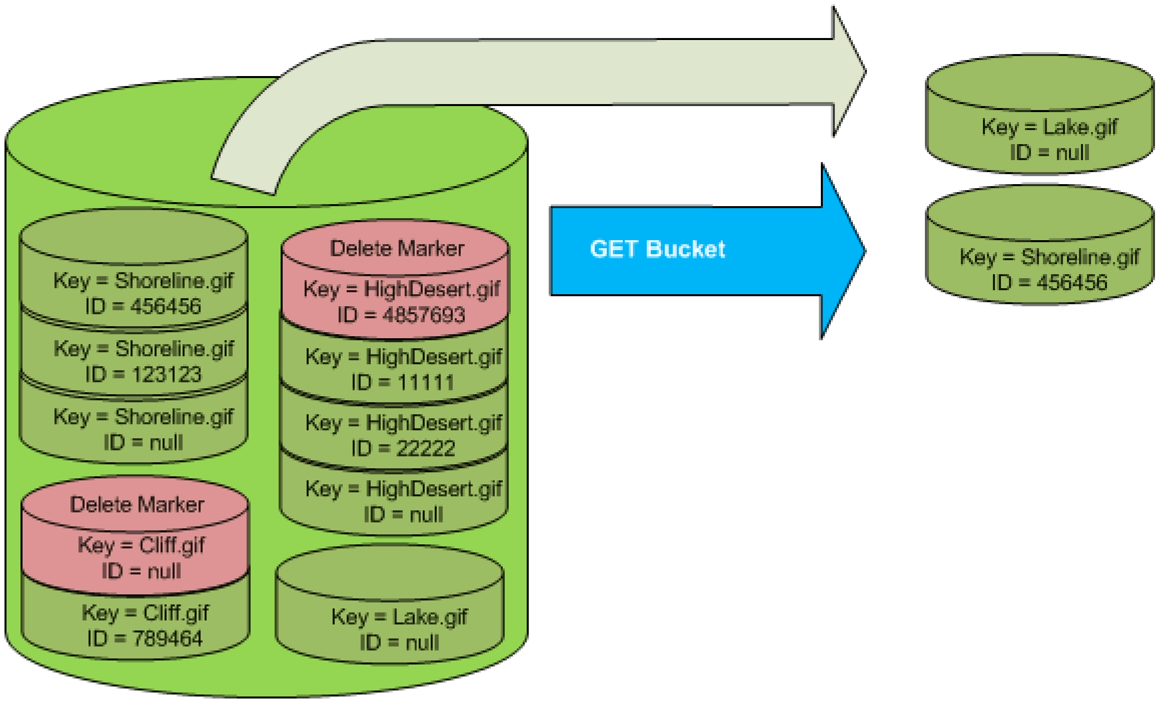
- A 404 (Object not found) error
- A response header, x-amz-delete-marker: true

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 latest version is a delete marker, returns a 404 No Object Found error.



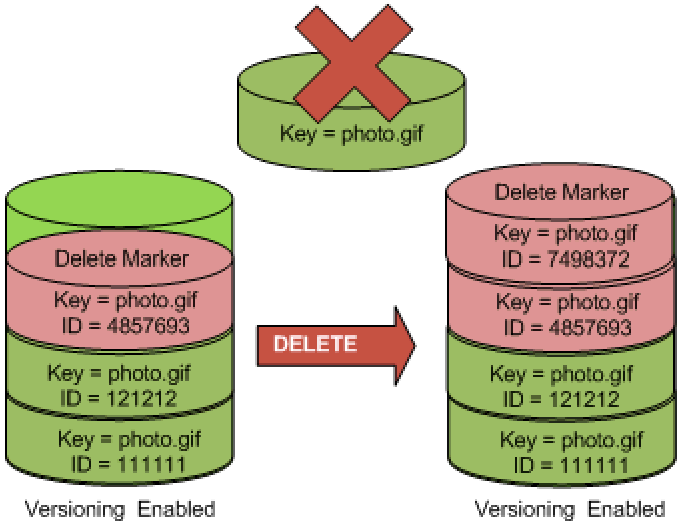
The only way to list delete markers (and other versions of an object) is by using the *versions* sub-resource in a `GET Bucket versions` request. A simple `GET` does not retrieve delete marker objects. The following figure shows that a `GET Bucket` request does not return objects whose latest version is a delete marker.



Removing Delete Markers

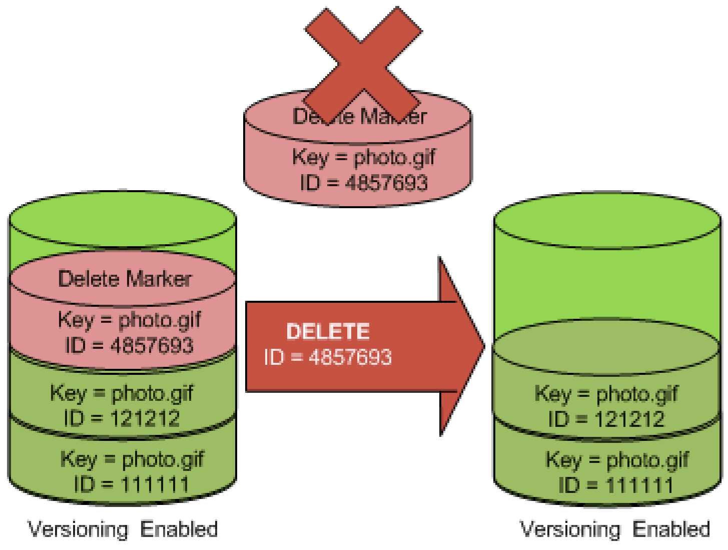
To delete a delete marker, you must specify its version ID in a `DELETE Object versionId` request. If you use `DELETE` to delete a delete marker (without specifying the version ID of the delete marker), Amazon S3 does not delete the delete marker, but instead, inserts another delete marker.

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 latest 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
```

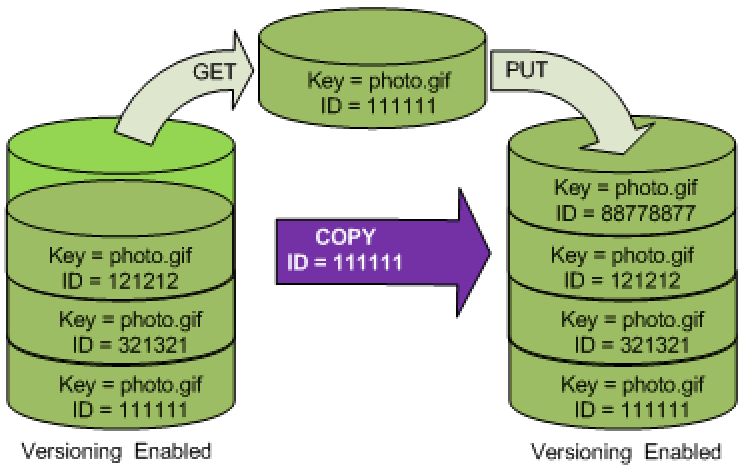
Note
Delete markers accrue a nominal charge for storage in Amazon S3. Delete markers are equal to the size of your key name, e.g., a Delete Marker for the key "photo.gif" adds 9 bytes of storage to your bucket.

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:

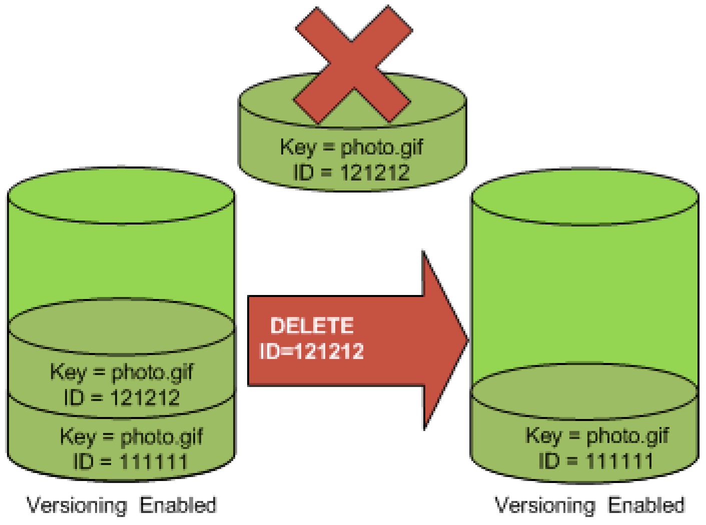
- Copy a previous version of the object into the same bucket
  The copied object becomes the latest version of that object and all object versions are preserved.
- Permanently delete the latest version of the object
  When you delete the latest object version you, in effect, turn the previous version into the latest version of that object.

Because all object versions are preserved, you can make any earlier version the latest 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 latest 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 latest version (121212) of an object, which leaves the previous version (111111) as the latest object.



A subsequent `GET` will retrieve version 111111.

Versioned Object Permissions and ACLs

Permissions are set at the version level. Each version has its own object owner—whoever `PUT` the version. 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 Authentication and Access Control (p. 277).

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, 3HL4kqtJvjVBH40Nrjfkd.

```
PUT /my-image.jpg?acl&versionId=3HL4kqtJvjVBH40Nrjfkd 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 latest 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, go to GET Object acl.

Working with Versioning-Suspended Buckets

Topics

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.

Note

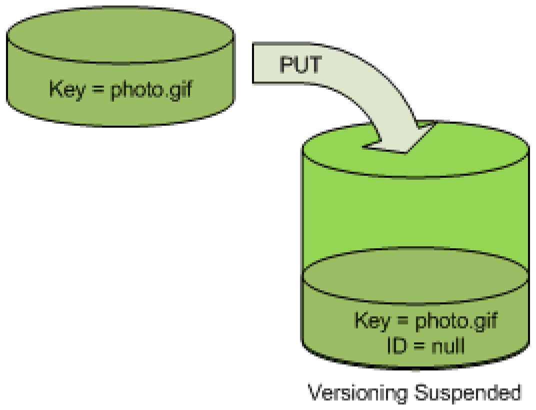
It is possible to have versioned objects in a versioning-suspended bucket if the objects were added when the bucket was versioning-enabled.

The following sections describe the behavior of buckets and objects when versioning is suspended.

Adding Objects to Versioning-Suspended Buckets

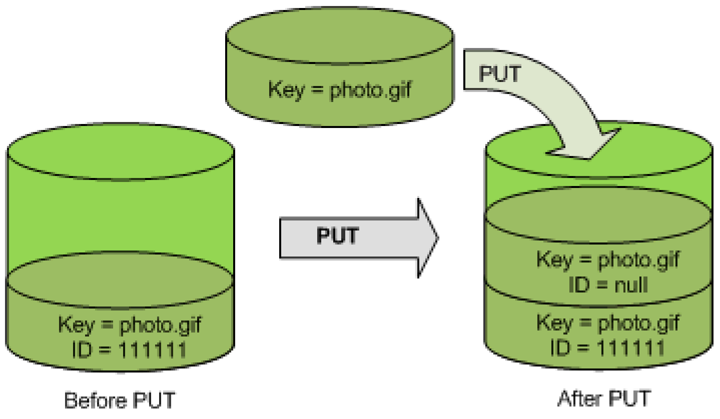
Once you suspend versioning on a bucket, Amazon S3 automatically adds a `null` version ID to every subsequent object stored thereafter (using `PUT`, `POST`, or `COPY`) in that bucket.

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.

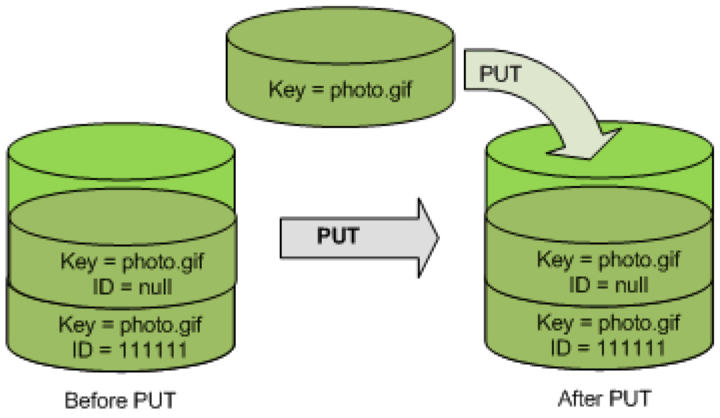


If a null version is already in the bucket and you add another object with the same key, the added object overwrites the original null version.

If there are versioned objects in the bucket, the version you `PUT` becomes the latest 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.



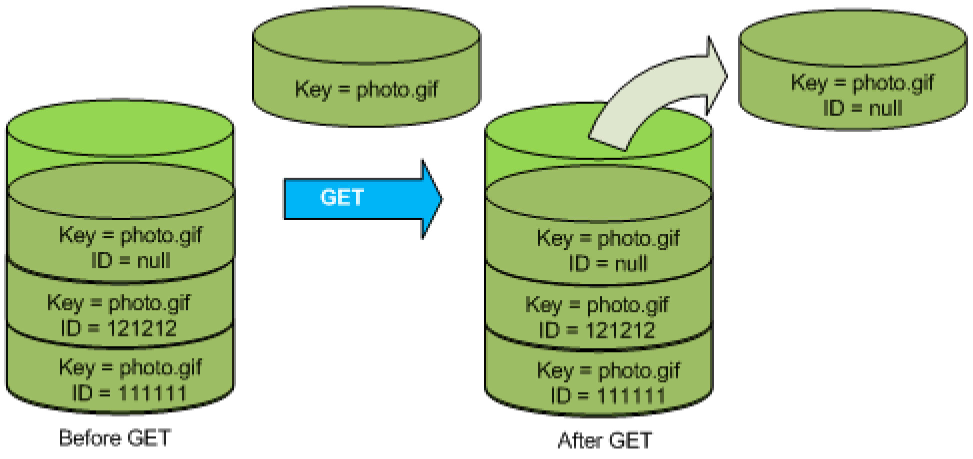
Note that although the key and version ID (`null`) of null version are the same before and after the `PUT`, the contents of the null version originally stored in the bucket is replaced by the contents of the object `PUT` into the bucket.

Adding Objects to Version-Suspended Buckets

1	Suspend versioning on a bucket using a `PUT Bucket versioning` request. For more information, go to PUT Bucket versioning.
2	Send a `PUT`, `POST`, or `COPY` request to store an object in the bucket.

Retrieving Objects from Versioning-Suspended Buckets

A `GET Object` request returns the latest version of an object whether you've enabled versioning on a bucket or not. The following figure shows how a simple `GET` returns the latest version of an object.



Example Retrieving the Latest Version

The following request retrieves the latest version of `photo.gif`.

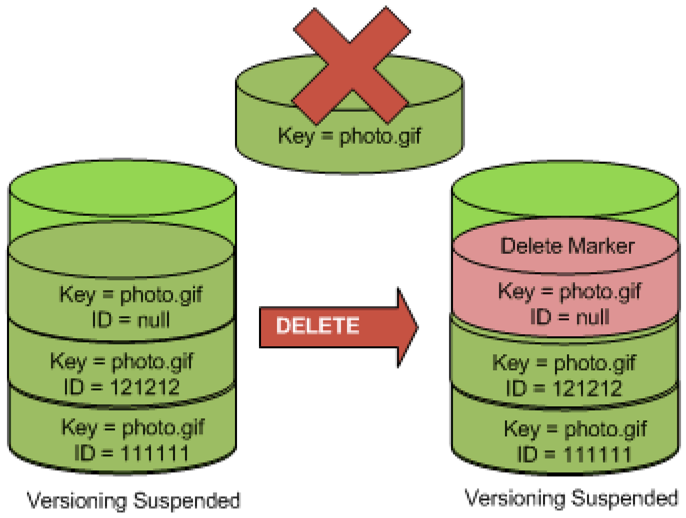
```
GET /photo.gif HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Deleting Objects from Versioning-Suspended Buckets

If versioning is suspended, a `DELETE` request:

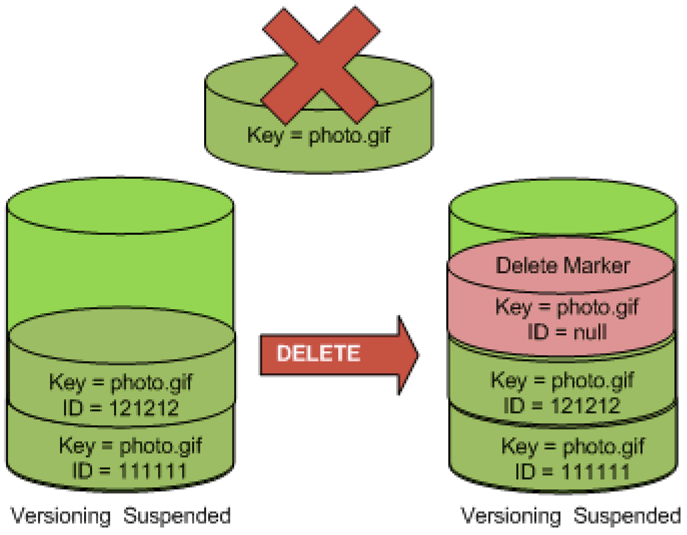
- Can only remove an object whose version ID is `null`
  Doesn't remove anything if there isn't a null version of the object in the bucket.
- Inserts a delete marker into the bucket

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`.

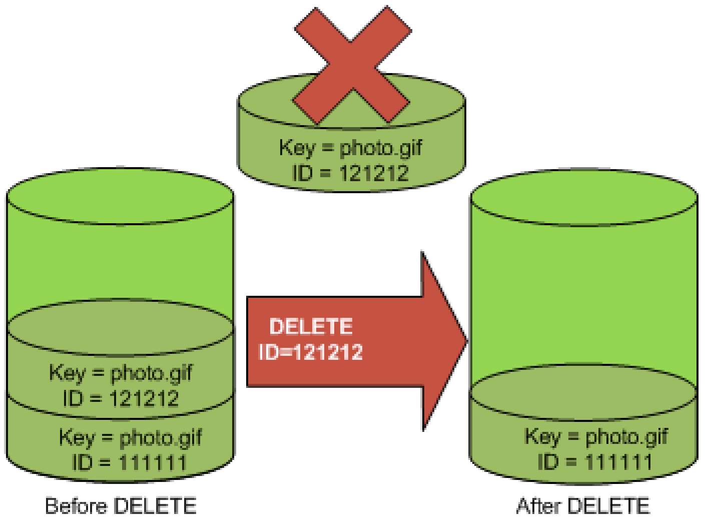


Remember that a delete marker doesn't have content so you lose the content of the null version when a delete marker replaces it.

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.



Hosting a Static Website on Amazon S3

Topics

You can host a static website on Amazon S3. On a static website, individual web pages include static content. They may 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.

To host your static website, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. The website is then available at the region-specific website endpoint of the bucket:

```
<bucket-name>.s3-website-<AWS-region>.amazonaws.com
```

For a list of region specific website endpoints for Amazon S3, see Website Endpoints (p. 378). For example, suppose you create a bucket called `examplebucket` in the US East 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-east-1.amazonaws.com/
  ```

- This URL requests the photo.jpg object, which is stored at the root level in the bucket.

  ```
  http://examplebucket.s3-website-us-east-1.amazonaws.com/photo.jpg
  ```

- This URL requests the `docs/doc1.html` object in your bucket.

  ```
  http://examplebucket.s3-website-us-east-1.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, in conjunction 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. 392).

To configure a bucket for website hosting, you add website configuration to the bucket. For more information, see Configure a Bucket for Website Hosting (p. 379).

Website Endpoints

Topics

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 endpoints, see Request Endpoints (p. 13).

The general form of an Amazon S3 website endpoint is as follows:

```
bucket-name-s3-website-region.amazonaws.com
```

For example, if your bucket is named `example-bucket` and it resides in the US Standard region, the website is available at the following Amazon S3 website endpoint:

```
http://example-bucket.s3-website-us-east-1.amazonaws.com/
```

The following table lists Amazon S3 regions and the corresponding website endpoints.

Region	Website endpoint
US Standard	***bucket-name*-s3-website-us-east-1.amazonaws.com**
US West (Oregon) Region	***bucket-name*-s3-website-us-west-2.amazonaws.com**
US West (Northern California) Region	***bucket-name*-s3-website-us-west-1.amazonaws.com**
EU (Ireland) Region	***bucket-name*-s3-website-eu-west-1.amazonaws.com**
Asia Pacific (Singapore) Region	***bucket-name*-s3-website-ap-southeast-1.amazonaws.com**
Asia Pacific (Sydney) Region	***bucket-name*-s3-website-ap-southeast-2.amazonaws.com**
Asia Pacific (Tokyo) Region	***bucket-name*-s3-website-ap-northeast-1.amazonaws.com**

Region	Website endpoint
South America (Sao Paulo) Region	***bucket-name*-s3-website-sa-east-1.amazonaws.com**

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 or DevPay buckets do not allow access through the website endpoint. Any request to such a bucket will receive a `403 Access Denied` response. For more information, see Requester Pays Buckets (p. 94).

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` will be routed to `www.example-bucket.com.s3-website-<region>.amazonaws.com`. For more information, see Virtual Hosting of Buckets (p. 59).

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
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.
Error message handling	Returns an XML-formatted error response.	Returns an HTML document.
Access control	Supports both public and private content.	Supports only publicly readable content.
Redirection support	Not applicable	Supports both object-level and bucket-level redirects.

Configure a Bucket for Website Hosting

Topics

Overview

To configure a bucket for static website hosting, you add a website configuration to your bucket. The configuration includes the following information:

- Index document

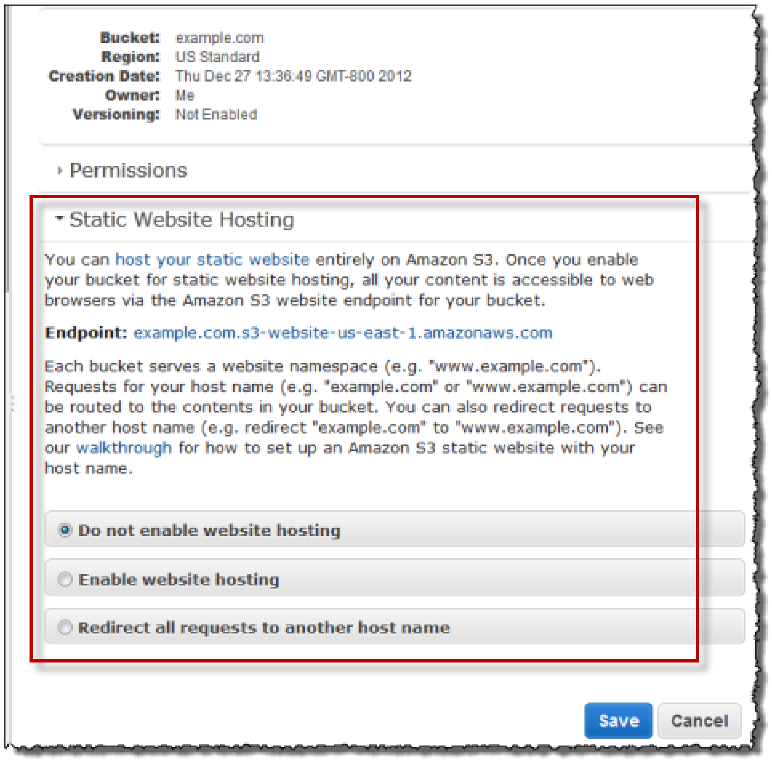
  When you type a URL such as http://example.com you are not requesting a specific page. In this case the web server serves a default page, for the directory where the requested website content is stored. This default page is referred as index document, and most of the time it is named index.html. When you configure a bucket for website hosting, you must specify an index document. Amazon S3 returns this index document when requests are made to the root domain or any of the subfolders. For more information, see Index Documents and Folders (p. 386).
- Error document

  If an error occurs, Amazon S3 returns an HTML error document. For 4XX class errors, you can optionally provide your own custom error document, in which you can provide additional guidance to your users. For more information, see Custom Error Document Support (p. 387).
- Redirects all requests

  If you 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`, maintain website content in only one bucket, say, `example.com`, and configure the other bucket to redirect all requests to the `example.com` bucket.
- Advanced conditional redirects

  You can conditionally route requests according to specific object key names or prefixes in the request, or according to the response code. 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. Suppose that you want to make a folder unavailable. You can add a routing rule to redirect the request to another page, which explains why the folder is no longer available. You can also add a routing rule to handle an error condition by routing requests that return the error to another domain, where the error will be processed.

You can manage your buckets website configuration using the Amazon S3 console. The bucket **Properties** panel in the console enables you to specify the website configuration.



Bucket: example.com
Region: US Standard
Creation Date: Thu Dec 27 13:36:49 GMT-800 2012
Owner: Me
Versioning: Not Enabled

Permissions

Static Website Hosting

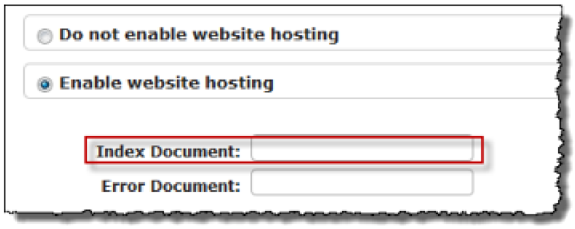
You can host your static website entirely on Amazon S3. Once you enable your bucket for static website hosting, all your content is accessible to web browsers via the Amazon S3 website endpoint for your bucket.

Endpoint: example.com.s3-website-us-east-1.amazonaws.com

Each bucket serves a website namespace (e.g. "www.example.com"). Requests for your host name (e.g. "example.com" or "www.example.com") can be routed to the contents in your bucket. You can also redirect requests to another host name (e.g. redirect "example.com" to "www.example.com"). See our walkthrough for how to set up an Amazon S3 static website with your host name.

[✓ Do not enable website hosting]
Enable website hosting
Redirect all requests to another host name

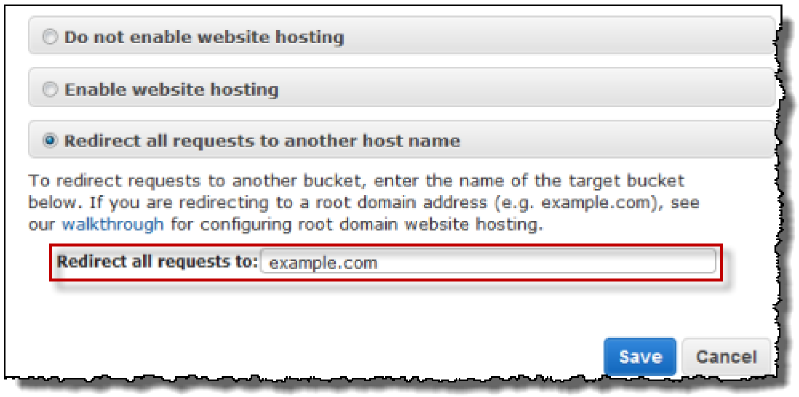
To host a static website on Amazon S3, you need only provide the name of the index document.



Do not enable website hosting
[✓ Enable website hosting]

Index Document:
Error Document:

To redirect all requests to the bucket's website endpoint to another host, you only need to provide host name.

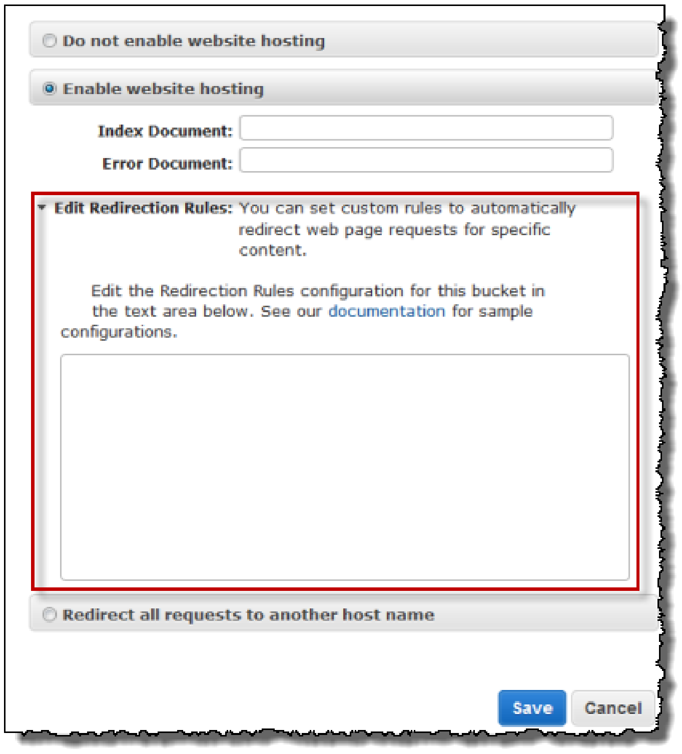


Do not enable website hosting
Enable website hosting
[✓ Redirect all requests to another host name]

To redirect requests to another bucket, enter the name of the target bucket below. If you are redirecting to a root domain address (e.g. example.com), see our walkthrough for configuring root domain website hosting.

Redirect all requests to: example.com

However, when configuring bucket for website hosting, you can optionally specify advanced redirection rules.



You can set custom rules to automatically redirect web page requests for specific content.

Edit the Redirection Rules configuration for this bucket in the text area below. See our documentation for sample configurations.

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 a 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 has at least one child element.

<Redirect> =
   <Redirect>
```

```
      [ <HostName>...</HostName> ]
      [ <Protocol>...</Protocol> ]
      [ <ReplaceKeyPrefixWith>...</ReplaceKeyPrefixWith>  ]
      [ <ReplaceKeyWith>...</ReplaceKeyWith> ]
      [ <HttpRedirectCode>...</HttpRedirectCode> ]
   </Redirect>
    Note: <Redirect> must has 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	A rule that identifies a condition and the redirect that is applied when the condition is met. Condition: A *RoutingRules*, container must contain at least one routing rule.
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	The object key name prefix from which requests will be redirected. *KeyPrefixEquals* is required if `HttpErrorCodeReturnedEquals` is not specified. If both *KeyPrefixEquals* and `HttpErrorCodeReturnedEquals` are specified, both must be true for the condition to be met.
HttpErrorCodeReturnedEquals	The HTTP error code that must match for the the redirect to apply. In the event of an error, if the error code meets this value, then specified redirect applies. *HttpErrorCodeReturnedEquals* is required if `KeyPrefixEquals` is not specified. If both *KeyPrefixEquals* and `HttpErrorCodeReturnedEquals` are specified, both must be true for the condition to be met.
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 elements: `Protocol`, `HostName`, `ReplaceKeyPrefixWith`, `ReplaceKeywith` or `HttpRedirectCode`.
Protocol	The protocol, http or https, to be used in the Location header that is returned in the response. *Protocol* is not required if one of it's siblings is supplied.
HostName	The host name to be used in the Location header that is returned in the response. *HostName* is not required if one of it's siblings is supplied.

Name	Description
ReplaceKeyPrefixWith	The object key name prefix that will replace the value of `KeyPrefixEquals` in the redirect request. *`ReplaceKeyPrefixWith`* is not required if one of it's siblings is supplied. It can be supplied only if `ReplaceKeyWith` is not supplied.
ReplaceKeyWith	The object key to be used in the Location header that is returned in the response. *`ReplaceKeyWith`* is not required if one of it's siblings is supplied. It can be supplied only if `ReplaceKeyPrefixWith` is not supplied.
HttpRedirectCode	The HTTP redirect code to be used in the Location header that is returned in the response. *`HttpRedirectCode`* is not required if one of it's siblings is supplied.

The following are some of the examples:

Example 1: Redirect after renaming a key prefix

Suppose your bucket contained the following objects:

index.html

docs/article1.html

docs/article2.html

Now you decided to rename the folder from `docs/` to `documents/`. After you make this change, you will need to redirect requests for prefix `/docs` to `documents/`. For example, request for `docs/article1.html` will need to 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 you delete the `images/` folder (that is, you delete all objects with 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>
</WebsiteConfiguration>
```

Example 3: Redirect for an HTTP error

Suppose that when a requested object is not found, you want to redirect requests to an Amazon EC2 instance. You can add a redirection rule so that when an HTTP status code 404 (Not Found) is returned the site visitor is redirected to an EC2 instance that will handle 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 a HTTP 404 error, the request is redirected to a page `report-404/ExamplePage.html` on the specified EC2 instance. If there is no routing rule and the HTTP error 404 occurs, the error document 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>
</WebsiteConfiguration>
```

Index Document Support

An index document is a webpage that is returned 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, you should provide the name of the index document. You must 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 will return `index.html`.

```
http://example-bucket.s3-website-region.amazonaws.com/
http://example-bucket.s3-website-region.amazonaws.com
```

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`

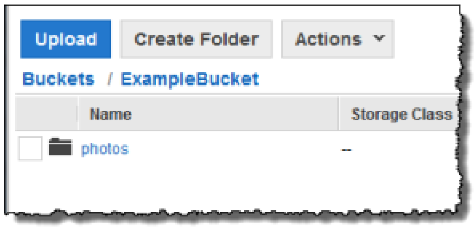
Although these are stored with no physical hierarchical organization, you can infer the following logical folder structure from the key names.

`sample1.jpg` object is at the root of the bucket

`sample2.jpg` object is in the `photos/2006/Jan` subfolder, and

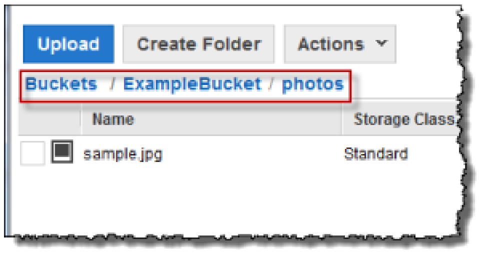
`sample3.jpg` object is in `photos/2006/Feb` subfolder.

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.



	Name	Storage Class
	photos	--

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`.



	Name	Storage Class
	sample.jpg	Standard

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/
```

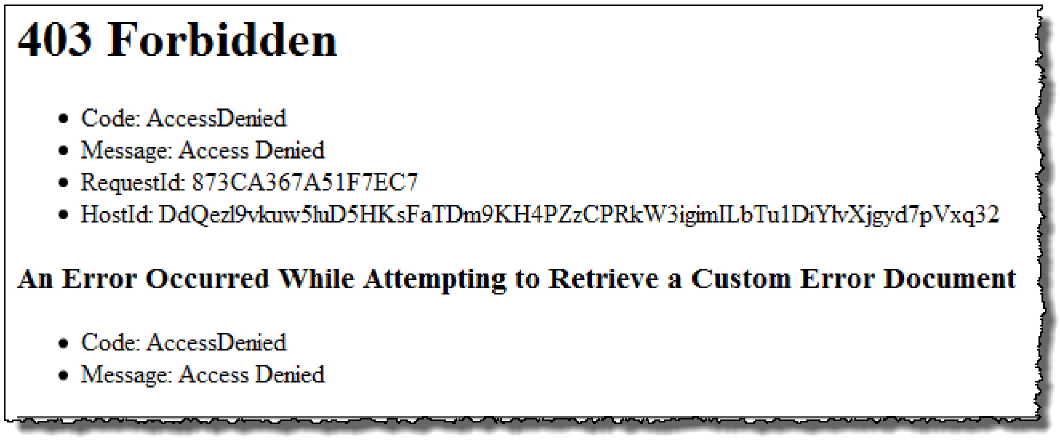
However, if you exclude the trailing slash from the preceding URL, Amazon S3 first looks for an object `photos` in the bucket. If the `photos` object is not found, then it searches for an index document, `photos/index.html`. If that document is found, Amazon S3 returns a `302 Found` message and points 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.

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 (`http://s3-website-<region>.amazonaws.com/`), Amazon S3 returns a **301 Moved Permanently** response and redirects those requests to `http://aws.amazon.com/s3`.
302 Found	When Amazon S3 receives a request for a key `x`, `http://<bucket>.s3-website-<region>.amazonaws.com/x`, without a trailing slash, it first looks for the object with the keyname `x`. If the object is not found, Amazon S3 determines that the request is for subfolder `x` and redirects the request by adding a slash at the end, and returns **302 Found**.
304 Not Modified	Amazon S3 users request headers `If-Modified-Since`, `If-Unmodified-Since`, `If-Match` and/or `If-None-Match` 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: • Amazon S3 determines the website URL refers to an object key that does not exist • Amazon infers the request is for an index document that does not exist • A bucket specified in the URL does not exist • A bucket specified in the URL exists, however, it is not configured as a website For information on how Amazon S3 interprets the URL as a request for an object or an index document, see Index Document Support (p. 385).
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 as shown in the following sample HTML returned for **403 Forbidden** response.


403 Forbidden

- Code: AccessDenied
- Message: Access Denied
- RequestId: 873CA367A51F7EC7
- HostId: DdQezl9vkuw5luD5HKsFaTDm9KH4PZzCPRkW3igimILbTu1DiYlvXjgyd7pVxq32

An Error Occurred While Attempting to Retrieve a Custom Error Document

- Code: AccessDenied
- Message: Access Denied

You can optionally provide a custom error document with a user-friendly error message and with 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 have configured your website with a custom error document, Amazon S3 returns that error document. However, note that when an error occurs, some browsers display their own error message, ignoring the error document Amazon S3 returns. For example, when an HTTP 404 Not Found error occurs, Chrome might display its own error ignoring the error document that Amazon S3 returns.

Configuring a Web Page 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. 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 Metadata (p. 101).

A bucket that 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 does not redirect the page request. It returns the requested object.

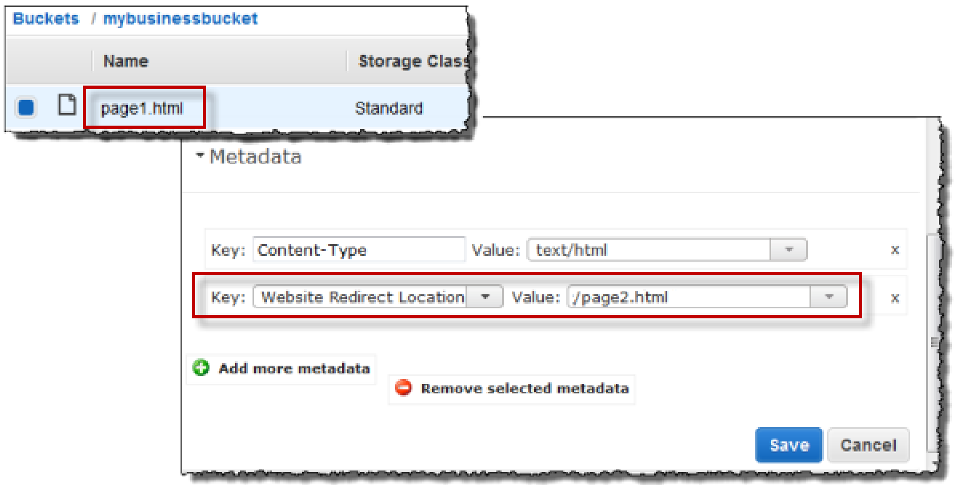
For more information about the endpoints, see Key Differences Between the Amazon Website and the REST API Endpoint (p. 379).

You can set a page redirect from the Amazon S3 console or by using the Amazon S3 REST API

Page Redirect Support 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 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, under **Properties** for `page1.html`, click the **Metadata** tab. 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`.

- 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, and then specify `Website Redirect Location` for page1.html in the upload process. For information about uploading an object, go to Uploading Objects into Amazon S3 in the *Amazon S3 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 any requests will be 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.

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 so, you write a bucket policy that grants everyone `s3:GetObject` permission. On the website endpoint, if a user requests an object that does not exist, Amazon S3 returns HTTP response code `404 (Not Found)`. If the object exists but you have not granted read permission on the object, the website endpoint returns HTTP response code `403 (Access Denied)`. The user can use the response code to infer if a specific object existsor not. If you do not 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 on bucket policies, see Using Bucket Policies (p. 306).

```
{
  "Version":"2008-10-17",
  "Statement":[{
 "Sid":"PublicReadGetObject",
        "Effect":"Allow",
   "Principal": {
            "AWS": "*"
         },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::example-bucket/*"
      ]
    }
  ]
}
```

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, you grant READ permission to the AllUsers group as shown in the following grant element. You add this grant element to the object ACL. For information on managing ACLs, see Using ACLs (p. 323).

```
<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>
```

Example Walkthroughs - Hosting Websites On Amazon S3

Topics

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 Amazon S3 bucket website endpoint, and server 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. In the following procedure, you will use the AWS Management Console to perform the necessary tasks:

1. Create an Amazon S3 bucket and configure it as a website.
2. Add a bucket policy that make the bucket content public.

   The content that you serve at the website endpoint must be publicly readable. You can grant the necessary permissions by adding a bucket policy or using Access Control List (ACL). Here we describe adding a bucket policy.
3. Upload an index document.
4. Test your website using the Amazon S3 bucket website endpoint.

To create a bucket and configure 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, go to Create a bucket in *Amazon Simple Storage Service Console User Guide*.
   For bucket naming guidelines, see Bucket Restrictions and Limitations (p. 82). If you have your registered domain name, for additional information about bucket naming, see Customizing Amazon S3 URLs with CNAMEs (p. 62) and Customizing Amazon S3 URLs with CNAMEs (p. 62).
3. Open the bucket **Properties** panel, click **Static Website Hosting** and do the following:
   1. Select the **Enable website hosting**.
   2. In the **Index Document** box, add the name of your index document. This name is typically index.html.
   3. Click **Save** to save the website configuration.
   4. Note down the **Endpoint**.

      This is the Amazon S3 provided website endpoint for your bucket. You will use this endpoint in the following steps to test your website.

To add a bucket policy that makes your bucket content publicly available

1. In bucket **Properties** panel, click the **Permissions**.
2. Click **Add Bucket Policy**.
3. Copy the following bucket policy, and then paste it in the Bucket Policy Editor.

```
{
  "Version":"2008-10-17",
  "Statement":[{
 "Sid":"PublicReadForGetBucketObjects",
        "Effect":"Allow",
   "Principal": {
            "AWS": "*"
         },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::example-bucket/*"
      ]
    }
  ]
}
```

4. In the policy, replace *example-bucket* with the name of your bucket.
5. Click **Save**.

To upload an index document

1. Create a document. The file name must be same as the name that you provided for the index document earlier.
2. Using the console, upload the index document to your bucket.
   For step-by-step instructions, go to Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.

Test your website.

- Enter the following URL in the browser, replacing example-bucket with the name of your bucket and *website-region* with the name of the region where you deployed your bucket. For information about region names, see Website Endpoints (p. 378)).

```
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 server the content from the website you created. You might also want to use Amazon S3's root domain support to serve requests for both the `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. 392).

Example: Setting Up a Static Website Using a Custom Domain

Topics

Suppose you want to host your static website on Amazon S3. You have 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 now want to host on Amazon S3 or you are starting from scratch, this example will help you host websites on Amazon S3.

Before You Begin

As you walk through the steps in this example, note that you will work with the following services:

Domain registrar of your choice– If you do not already have a registered domain name, such as `example.com`, you will need to create and register one with a registrar of your choice. You can typically register a domain for a small yearly fee. For procedural information about registering a domain name, see the web site of the registrar

Amazon S3– You will use Amazon S3 to create buckets, upload a sample website page, configure permissions so everyone can see the content, and then configure the buckets for website hosting. In this example, because you want to allow requests for both `http://www.example.com` and `http://example.com`, you will create two buckets; however, you will host content in only one bucket. You will configure the other Amazon S3 bucket to redirect requests to the bucket that hosts the content.

Amazon Route 53– You will configure Amazon Route 53 as your DNS provider. You will create a hosted zone in Amazon Route 53 for your domain and configure applicable DNS records. If you are switching from an existing DNS provider, you will need to ensure that you have transferred all of the DNS records for your domain.

As you walk through this example, a basic familiarity with domains, Domain Name System (DNS), CNAME records, and A records would be helpful. A detailed explanation of these concepts is beyond the scope of this guide, but your domain registrar should provide any basic information that you need.

Note
All the steps in this example use `example.com` as a domain name. You will need to replace this domain name with the one you registered.

Step 1: Register a Domain

If you already have a registered domain, you can skip this step. If you are new to hosting a website, your first step is to register a domain, such as `example.com`, with a registrar of your choice.

After you have chosen a registrar, you will register your domain name according to the instructions at the registrar's website. For a list of registrar web sites that you can use to register your domain name, go to ICANN.org.

When you have a registered domain name, your next task is 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

In this example, to support requests from both the root domain such as `example.com` and subdomain such as `www.exmaple.com`, you will create two buckets. One bucket will contain the content and you will configure the other bucket to redirect requests. You perform the following tasks in Amazon S3 console to create and configure your website:

1. Create two buckets.
2. Configure these buckets for website hosting.
3. Test the Amazon S3 provided bucket website endpoint.

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 will support requests from both `example.com` and `www.example.com`.

In this step, you will sign in to the Amazon S3 console with your AWS account credentials and create the following two buckets.

- *`example.com`*
- `www.`*`example.com`*

Note
To create the buckets for this example, follow these steps. As you walk through this example, substitute the domain name that you registered for *`example.com`*.

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. One of the bucket names should include the subdomain www, For instance, *`example.com`* and `www.`*`example.com`*.
   For step-by-step instructions, go to Creating a 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. Note that you can store content in either bucket. For this example you will host content in *`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 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 will 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, go to Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
4. Configure permissions for your objects to make them publicly accessible.

Attach the following bucket policy to the *example.com* bucket substituting the name of your bucket for e*xample.com*. For step-by-step instructions to attach a bucket policy, go to Editing Bucket Permissions in the *Amazon Simple Storage Service Console User Guide*.

```
{
  "Version":"2008-10-17",
  "Statement":[{
 "Sid":"AddPerm",
        "Effect":"Allow",
   "Principal": {
            "AWS": "*"
         },
      "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 will 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 and both visitors who specify “www” in their browsers and visitors that only specify the root domain will both be routed to the same website content in your “example.com” bucket.

Step 2.2: Configure Buckets for Website Hosting

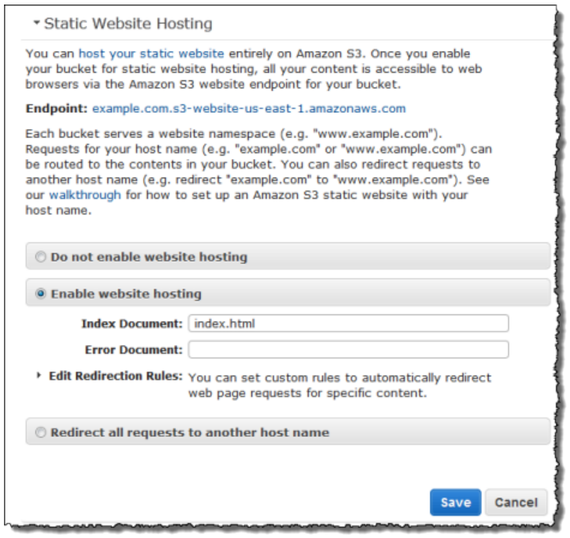
When you configure a bucket for website hosting, you can access the website using the Amazon S3 assigned bucket website endpoint.

In this step, you will configure both buckets for website hosting. First, you will configure *example.com* as a website and then you'll configure `www.`*example.com* to redirect all requests to the *example.com* bucket.

To configure *example.com* bucket for website hosting

1. Configure *example.com* bucket for website hosting. In the **Index Document** box, type the name that you gave your index page.

   For step-by-step-instructions, go to Managing Bucket Website Configuration in the *Amazon Simple Storage Service Console User Guide*. Make a note of the URL for the website endpoint. You will need it later.



▾ Static Website Hosting

You can host your static website entirely on Amazon S3. Once you enable your bucket for static website hosting, all your content is accessible to web browsers via the Amazon S3 website endpoint for your bucket.

Endpoint: example.com.s3-website-us-east-1.amazonaws.com

Each bucket serves a website namespace (e.g. "www.example.com"). Requests for your host name (e.g. "example.com" or "www.example.com") can be routed to the contents in your bucket. You can also redirect requests to another host name (e.g. redirect "example.com" to "www.example.com"). See our walkthrough for how to set up an Amazon S3 static website with your host name.

( ) Do not enable website hosting

(●) Enable website hosting

Index Document: index.html

Error Document:

▸ Edit Redirection Rules: You can set custom rules to automatically redirect web page requests for specific content.

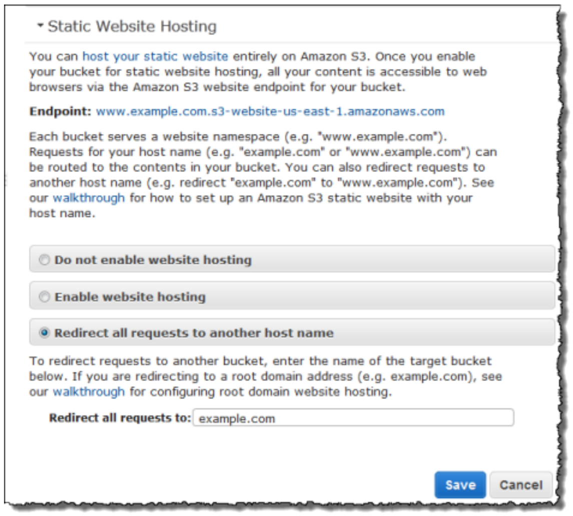
( ) Redirect all requests to another host name

2. To test the website, enter the **Endpoint** URL in your browser.

   Your browser will display the index document page. Next, you will configure `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, right-click `www.`*`example.com`* and then click **Properties**.
2. Under **Static Website Hosting**, click **Redirect all requests to another host name**. In the **Redirect all requests** box, type *`example.com`*.



▾ Static Website Hosting

You can host your static website entirely on Amazon S3. Once you enable your bucket for static website hosting, all your content is accessible to web browsers via the Amazon S3 website endpoint for your bucket.

Endpoint: www.example.com.s3-website-us-east-1.amazonaws.com

Each bucket serves a website namespace (e.g. "www.example.com"). Requests for your host name (e.g. "example.com" or "www.example.com") can be routed to the contents in your bucket. You can also redirect requests to another host name (e.g. redirect "example.com" to "www.example.com"). See our walkthrough for how to set up an Amazon S3 static website with your host name.

( ) Do not enable website hosting

( ) Enable website hosting

(●) Redirect all requests to another host name

To redirect requests to another bucket, enter the name of the target bucket below. If you are redirecting to a root domain address (e.g. example.com), see our walkthrough for configuring root domain website hosting.

Redirect all requests to: example.com

3. To test the website, enter the **Endpoint** URL in your browser.

   Your request will be redirected and the browser will display the index document for *`example.com`*.

The following Amazon S3 bucket website endpoints are accessible to any internet user:

`http://`*`example.com`*`.s3-website-us-east-1.amazonaws.com`

`http://www.`*`example.com`*`.s3-website-us-east-1.amazonaws.com`

Now you will do additional configuration to serve requests from the domain you registered in the preceding step. For example, if you registered a domain example.com, you want to serve requests from the following URLs :

`http://example.com`

`http://www.example.com`

In the next step, we will use Amazon Route 53 to enable customers to use the URLs above to navigate to your site.

Step 3: Create and Configure Amazon Route 53 Hosted Zone

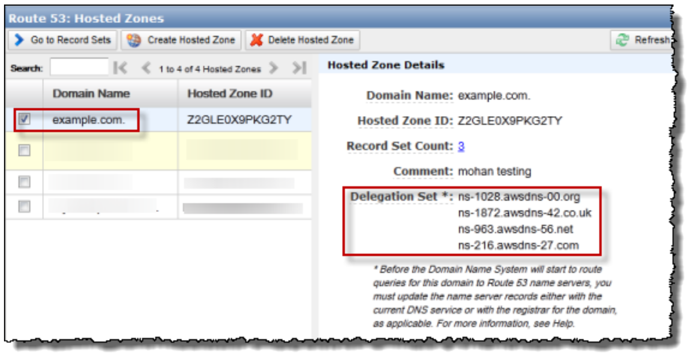
Now you will configure Amazon Route 53 as your Domain Name System (DNS) provider. You must use Amazon Route 53 if you want to serve content from your root domain, such as `example.com`. You will create a hosted zone, which holds the DNS records associated with your domain:

- Alias record– Maps the domain `example.com` to the Amazon S3 website endpoint (for example, s3-website-us-east-1.amazonaws.com).
- CNAME record– Maps the domain `www.example.com.com` to the corresponding Amazon S3 bucket website endpoint. Any requests to `http://www.example.com` will be routed to the Amazon S3 bucket website endpoint.

Step 3.1: Create a Hosted Zone for Your Domain

Go to the Amazon Route 53 console at https://console.aws.amazon.com/route53 and then create a hosted zone for your domain. For step-by-step instructions, go to Creating a Hosted Zone in the *Amazon Route 53 Developer Guide*.

The following example shows the hosted zone created for the `example.com` domain. Write down the Route 53 name servers (NS) for this domain. You will need them later.



	Domain Name	Hosted Zone ID
✓	example.com.	Z2GLE0X9PKG2TY

Hosted Zone Details

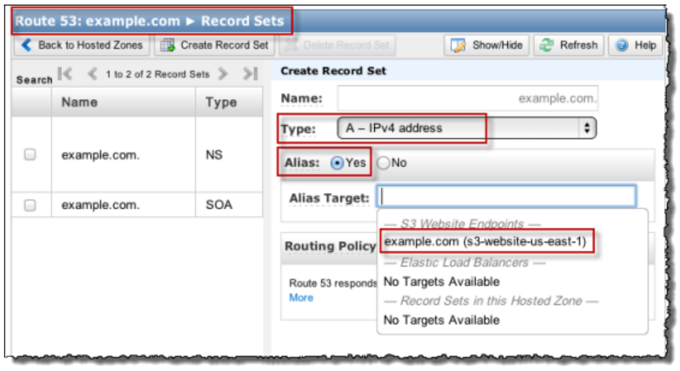
Domain Name: example.com.
Hosted Zone ID: Z2GLE0X9PKG2TY
Record Set Count: 3
Comment: mohan testing
Delegation Set *: ns-1028.awsdns-00.org
ns-1872.awsdns-42.co.uk
ns-963.awsdns-56.net
ns-216.awsdns-27.com

* Before the Domain Name System will start to route queries for this domain to Route 53 name servers, you must update the name server records either with the current DNS service or with the registrar for the domain, as applicable. For more information, see Help.

Step 3.2: Add an Alias Record to the Hosted Zone

The alias record that you add to the hosted zone for your domain will map `example.com` to the corresponding Amazon S3 bucket. Instead of using an IP address, the alias record uses the Amazon S3 website endpoint. Amazon Route 53 maintains a mapping between the alias record and the IP address where the Amazon S3 bucket resides.

For step-by-step instructions, go to Creating an Alias Resource Record Set in the *Amazon Route 53 Developer Guide*.



Name	Type
example.com.	NS
example.com.	SOA

Create Record Set

Name: example.com.

Type: A – IPv4 address

Alias: [✓ Yes] No

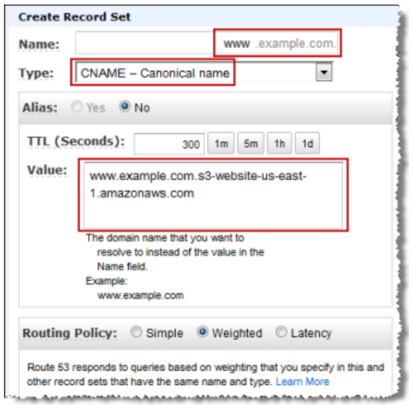
Alias Target:

— S3 Website Endpoints —
example.com (s3-website-us-east-1)
— Elastic Load Balancers —
No Targets Available
— Record Sets in this Hosted Zone —
No Targets Available

To enable this hosted zone, you must use Amazon Route 53 as the DNS server for your domain `example.com`. Before you switch, if you are moving an existing website to Amazon S3, you must transfer DNS records associated with your domain `example.com` to the hosted zone that you created in Amazon Route 53 for your domain. If you are creating a new website, you can go directly to step 4.

Step 3.3: Add a CNAME Record to the Hosted Zone

The CNAME record that you add routes requests for `www.example.com` to the endpoint for the corresponding Amazon S3 bucket. For step-by-step instructions, go to Creating, Changing, and Deleting Resource Record Sets in the *Amazon Route 53 Developer Guide*. The CNAME value shown in the following example is the host name that you specified in the Amazon S3 console.



Create Record Set

Name: www .example.com.

Type: CNAME – Canonical name

Alias: Yes [✓ No]

TTL (Seconds): 300

Value: www.example.com.s3-website-us-east-1.amazonaws.com

The domain name that you want to resolve to instead of the value in the Name field.
Example:
www.example.com

Routing Policy: Simple [✓ Weighted] Latency

Route 53 responds to queries based on weighting that you specify in this and other record sets that have the same name and type. Learn More

Step 3.4: Transfer Other DNS Records from Your Current DNS Provider to Route 53

Before you switch to Amazon Route 53 as your DNS provider, you must transfer any remaining DNS records from your current DNS provider, including MX records, CNAME records, and A records, to Amazon Route 53. You don't need to transfer the following records:

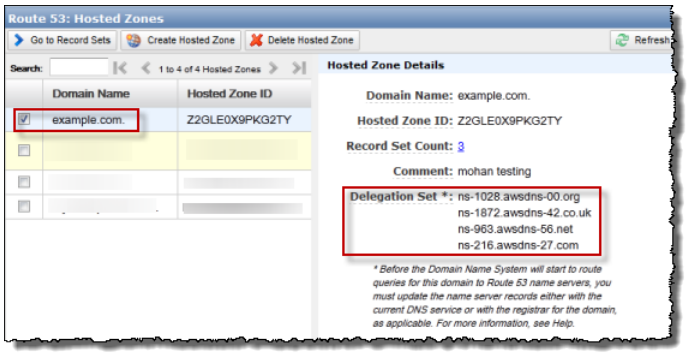
- NS records– Instead of transferring these, you replace their values with the name server values that are provided by Amazon Route 53.
- SOA record– Amazon Route 53 provides this record in the hosted zone with a default value.

Migrating required DNS records is a critical step to ensure the continued availability of all the existing services hosted under the domain name.

Step 4: Switch to Amazon Route 53 as Your DNS Provider

To switch to Amazon Route 53 as your DNS provider, you must go to your current DNS provider and update the name server (NS) record to use the name servers in your delegation set in Amazon Route 53.

Go to your DNS provider site and update the NS record with the delegation set values of the hosted zone as shown in the following Amazon Route 53 console screenshot. For more information, go to Updating Your DNS Service's Name Server Records in *Amazon Route 53 Developer Guide*.



	Domain Name	Hosted Zone ID
✓	example.com.	Z2GLE0X9PKG2TY

Hosted Zone Details

Domain Name: example.com.
Hosted Zone ID: Z2GLE0X9PKG2TY
Record Set Count: 3
Comment: mohan testing
Delegation Set *: ns-1028.awsdns-00.org
ns-1872.awsdns-42.co.uk
ns-963.awsdns-56.net
ns-216.awsdns-27.com

** Before the Domain Name System will start to route queries for this domain to Route 53 name servers, you must update the name server records either with the current DNS service or with the registrar for the domain, as applicable. For more information, see Help.*

When the transfer to Route 53 is complete, there are tools that you can use to verify the name server for your domain has indeed changed. On a Linux computer, you can use the `dig` DNS lookup utility. For example, this `dig` command:

```
dig +recurse +trace www.example.com any
```

returns the following output (only partial output is shown). The output shows the same four name servers the name servers on Amazon Route 53 hosted zone you created for `example.com` domain.

```
...
example.com.      172800  IN      NS      ns-9999.awsdns-99.com.
example.com.      172800  IN      NS      ns-9999.awsdns-99.org.
example.com.      172800  IN      NS      ns-9999.awsdns-99.co.uk.
example.com.      172800  IN      NS      ns-9999.awsdns-99.net.

www.example.com.  300     IN      CNAME   www.example.com.s3-website-us-east-
1.amazonaws.com.
...
```

Step 5: 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`*.

Setting Up Notification of Bucket Events

The Amazon S3 notifications feature enables you to configure a bucket so that Amazon S3 publishes a message to an Amazon Simple Notification Service (SNS) topic when Amazon S3 detects a key event on a bucket. Subscribers to this topic can have messages for bucket events delivered to an endpoint such as a web server, e-mail address, or an Amazon Simple Queue Service queue.

Note that Notification fees will be charged at the normal Amazon SNS rates. For details about pricing, go to http://aws.amazon.com/sns/#pricing. There are no additional Amazon S3 charges for publishing a message to an Amazon SNS topic beyond the request fee to enable and disable the feature. In order to receive notifications of a bucket's events, you must subscribe to the topic that has been configured for that bucket.

Currently, the *`s3:ReducedRedundancyLostObject`* event is the only event supported by Amazon S3. The *`s3:ReducedRedundancyLostObject`* event is triggered when Amazon S3 detects that it has lost all replicas of a Reduced Redundancy Storage (RRS) object and can no longer service requests for that object. Notification on this event can enable you to replace missing RRS objects proactively to ensure there is no customer impact.

Notifications are published to a specified Amazon SNS topic. Currently, only one topic can be configured for notifications. If the bucket owner and Amazon SNS topic owner are the same, the bucket owner has permission to publish notifications to the topic by default. Otherwise, the owner of the topic must create a policy to enable the bucket owner to publish to the topic. For more information about creating this policy, see Example Cases for Amazon SNS Access Control.

Important
The Amazon SNS topic specified for a bucket’s notification configuration must be in the same region as the bucket.

The following table lists the topic regions for the possible *`LocationContraint`* values for buckets.

LocationConstraint	Topic Region
Empty string	us-east-1
EU	eu-west-1
us-west-1	us-west-1

LocationConstraint	Topic Region
us-west-2	us-west-2
ap-northeast-1	ap-northeast-1
ap-southeast-1	ap-southeast-1
sa-east-1	sa-east-1

By default, only the bucket owner has permission to set and view the notifications on a bucket. However, the bucket owner can use a bucket policy to grant permission to other users to set this configuration using the *s3:PutBucketNotification* permission or view this configuration via the *s3:GetBucketNotification* permission. For more information about bucket policy, see Using Bucket Policies (p. 306).

Similar to other bucket settings, there is a sub-resource for each bucket enabling you to set or view that bucket's notification settings. To configure notifications on a bucket, use the bucket's *notification* sub-resource with the Amazon S3 REST API. For more information about the *notification* sub-resource, see Put Bucket notification and Get Bucket notification.

To set notifications for a bucket

- Use the PUT operation with a request body containing a *NotificationConfiguration* element that specifies the Amazon Resource Name (ARN) of the Amazon SNS topic, where the notification will be published, and what events will be reported.

```
PUT ?notification HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 02 June 2010 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
<NotificationConfiguration>
   <TopicConfiguration>
       <Topic>arn:aws:sns:us-east-1:111122223333:myTopic</Topic>
           <Event>s3:ReducedRedundancyLostObject</Event>
   </TopicConfiguration>
</NotificationConfiguration>
```

After you call the PUT operation to configure notifications on a bucket, Amazon S3 publishes a test notification to ensure that the topic exists and that the bucket owner has permission to publish to the specified topic. If the notification is successfully published to the SNS topic, the PUT operation updates the bucket configuration and returns a 200 OK response with a *x-amz-sns-test-message-id* header containing the message ID of the test notification sent to the topic. For information about PUT operation errors, see Put Bucket notification.

The message published for an event to a topic contains the fields listed in the following table.

Name	Description
Service	Amazon S3.
Event	Currently, the only supported event is *s3:ReducedRedundancyLostObject*.

Name	Description
Time	The time that the event was triggered.
Bucket	The name of the bucket.
Key	The object name.
VersionId	If versioning is enabled, the version ID. If versioning is not enabled, empty string.
RequestID	A unique ID that identifies which Amazon S3 operation published the notification.
HostID	A unique ID that identifies which host sent the message.

The following message is an example of a message publishing an *s3:ReducedRedundancyLostObject* event:

```
{
    "Service":"Amazon S3",
    "Event": "s3:ReducedRedundancyLostObject",
    "Time":"2010-02-11T23:48:22.000Z",
    "Bucket": "myphotos",
    "Key": "Home/2009/10/carvingpumpkins.jpg",
    "VersionId":"qfXHy5689N7n9mWVwanN_hIroMn_rzXl",
    "RequestId":"4442587FB7D0A2F9",
   "HostId":"fHZOHz+oQlKAFQ7RgVSklvcDNYI0v05gsOWWeJcyTTXWgGzI6CC5FZVICjD9bUzT"
}
```

The following message is an example of the test message that Amazon S3 sends when a bucket is enabled for notifications:

```
{
    "Service":"Amazon S3",
    "Event": "s3:TestEvent",
    "Time":"2010-02-11T23:48:22.000Z",
    "Bucket": "myphotos",
    "RequestId":"4442587FB7D0A2F9",
   "HostId":"fHZOHz+oQlKAFQ7RgVSklvcDNYI0v05gsOWWeJcyTTXWgGzI6CC5FZVICjD9bUzT"
}
```

To read the notification configuration for a bucket

- Use the GET operation, which returns a *NotificationConfiguration* element containing the Amazon SNS topic and events set for notification on the bucket.

```
GET ?notification HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 09 June 2010 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

To turn off notifications on a bucket

- Use the `PUT` operation with a request body containing an empty *`NotificationConfiguration`* element.

Request Routing

Topics

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

Overview

Amazon S3 uses the Domain Name System (DNS) to route requests to facilities that can process them. This system works very effectively. However, 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.

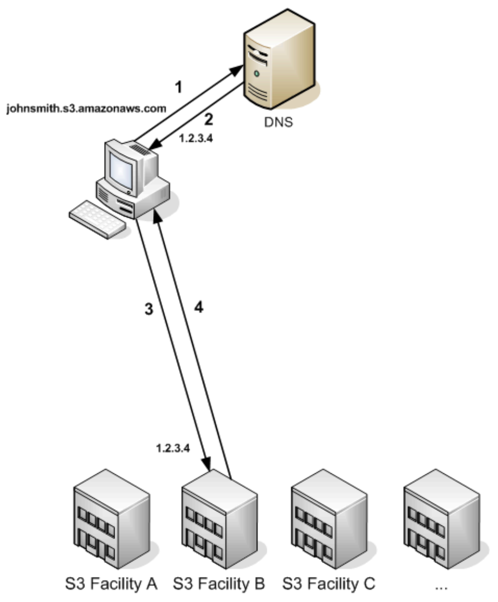
If a request is incorrectly formed, Amazon S3 uses permanent redirects to provide direction on how to perform the request correctly.

> **Important**
> Every Amazon S3 program must be designed to handle redirect responses. The only exception is for programs that work exclusively with buckets that were created without `<CreateBucketConfiguration>`. For more information on location constraints, see Buckets and Regions (p. 83).

DNS Routing

DNS routing routes requests to appropriate Amazon S3 facilities.

The following figure shows an example of DNS routing.



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 copy of the object.

Temporary Request Redirection

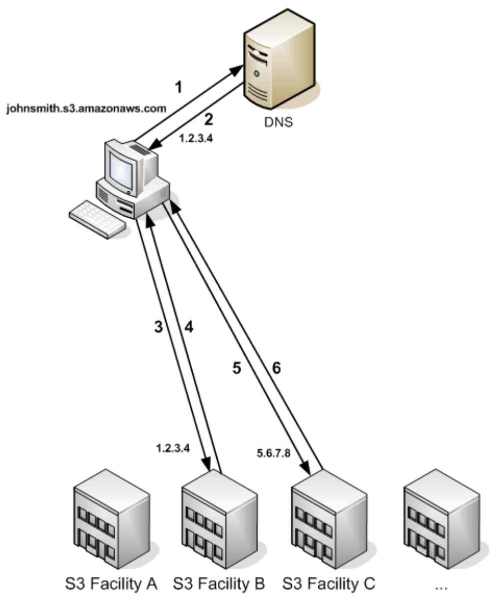
A temporary redirect is a type of error response that signals to the requester that he should resend his request to a different endpoint.

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 will receive a temporary redirect. After information about the bucket propagates through DNS, redirects will be rare.

Temporary redirects contain a URI to the correct facility which you can use to immediately resend the request.

> **Important**
> Do not reuse an endpoint provided by a previous redirect response. It might appear to work (even for long periods of time), but might provide unpredictable results and will eventually fail without notice.

The following figure shows an example of a temporary redirect.



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 Using CreateBucketConfiguration (p. 83).

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.

Example REST API Redirect

```
HTTP/1.1 307 Temporary Redirect
Location: http://johnsmith.s3-gztb4pa9sq.amazonaws.com/pho
tos/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>
```

Example SOAP API Redirect

```
<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.</Fault
string>
    <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.

Performance Optimization

Topics

Amazon S3 provides new features that support high performance networking. These include TCP window scaling and selective acknowledgements.

> **Note**
> For more information on high performance tuning, go to http://www.psc.edu/networking/projects/tcptune/.

TCP Window Scaling

TCP window scaling allows you to improve network throughput performance between your operating system and application layer and Amazon S3 by supporting window sizes larger than 64 KB. At the start of the TCP session, a client advertises its supported receive window WSCALE factor, and Amazon S3 responds with its supported receive window WSCALE factor for the upstream direction.

Although TCP window scaling can improve performance, it can be challenging to set correctly. Make sure to adjust settings at both the application and kernel level. For more information about TCP window scaling, refer to your operating system's documentation and go to RFC 1323.

TCP Selective Acknowledgement

TCP selective acknowledgement is designed to increase recovery time after a large number of packet losses. TCP selective acknowledgement is supported by most newer operating systems, but might have to be enabled. For more information about TCP selective acknowledgements, refer to the documentation that accompanied your operating system and go to RFC 2018.

Using BitTorrent with Amazon S3

Topics

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
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.

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.

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 Access Control (p. 277).

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.

Using Amazon DevPay with Amazon S3

Topics

Amazon DevPay enables you to charge customers for using your Amazon S3 product through Amazon's authentication and billing infrastructure. You can charge any amount for your product including usage charges (storage, transactions, and bandwidth), monthly fixed charges, and a one-time charge.

Once a month, Amazon bills your customers for you. AWS then deducts the fixed Amazon DevPay transaction fee and pays you the difference. AWS then separately charges you for the Amazon S3 usage costs incurred by your customers and the percentage-based Amazon DevPay fee.

If your customers do not pay their bills, AWS turns off access to Amazon S3 (and your product). AWS handles all payment processing.

Amazon S3 Customer Data Isolation

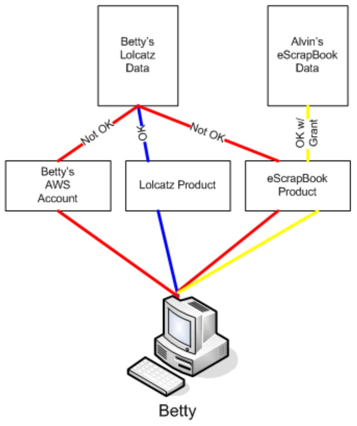
Amazon DevPay requests store and access data on behalf of the users of your product. The resources created by your application are owned by your users; unless you modify the ACL, you cannot read or modify the user's data.

Data stored by your product is isolated from other Amazon DevPay products and general Amazon S3 access. Customers that *store* data in Amazon S3 through your product can only *access* that data through your product. The data cannot be accessed through other Amazon DevPay products or through a personal AWS account.

Two users of a product can only access each other's data if your application explicitly grants access through the ACL.

Example

The following figure illustrates allowed, disallowed, and conditional (discretionary) data access.



Betty's access is limited as follows:

- She can access Lolcatz data through the Lolcatz product. If she attempts to access her Lolcatz data through another product or a personal AWS account, her requests will be denied.
- She can access Alvin's eScrapBook data through the eScrapBook product if access is explicitly granted.

Amazon DevPay Token Mechanism

To enable you to make requests on behalf of your customers and ensure that your customers are billed for use of your application, your application must send two tokens with each request: the product token and the user token.

The product token identifies your product; you must have one product token for each Amazon DevPay product that you provide. The user token identifies a user in relationship to your product; you must have a user token for each user/product combination. For example, if you provide two products and a user subscribes to each, you must obtain a separate user token for each product.

For information on obtaining product and user tokens, refer to the *Amazon DevPay Developer Guide.*

Amazon S3 and Amazon DevPay Authentication

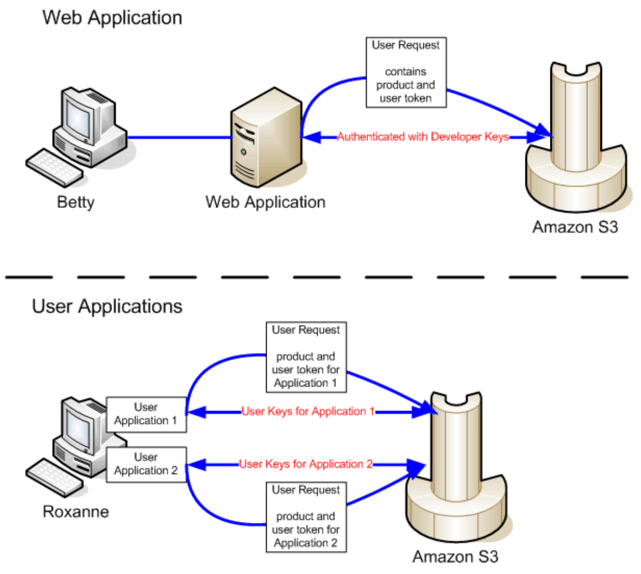
Although the token mechanism uniquely identifies a customer and product, it does not provide authentication.

Normally, your applications communicate directly with Amazon S3 using your Access Key ID and Secret Access Key. For Amazon DevPay, Amazon S3 authentication works a little differently.

If your Amazon DevPay product is a web application, you securely store the Secret Access Key on your servers and use the user token to specify the customer for which requests are being made.

However, if your Amazon S3 application is installed on your customers' computers, your application must obtain an Access Key ID and a Secret Access Key for each installation and must use those credentials when communicating with Amazon S3.

The following figure shows the differences between authentication for web applications and user applications.


Web Application
User Request contains product and user token
Authenticated with Developer Keys
Betty
Web Application
Amazon S3
User Applications
User Request product and user token for Application 1
User Application 1
User Keys for Application 1
User Application 2
User Keys for Application 2
User Request product and user token for Application 2
Roxanne
Amazon S3

Amazon S3 Bucket Limitation

Each of your customers can have up to 100 buckets for each Amazon DevPay product that you sell. For example, if a customer uses three of your products, the customer can have up to 300 buckets (100 * 3) plus any buckets outside of your Amazon DevPay products (i.e., buckets in Amazon DevPay products from other developers and the customer's personal AWS account).

Amazon S3 and Amazon DevPay Process

Following is a high-level overview of the Amazon DevPay process.

Launch Process

1	A customer signs up for your product through Amazon.
2	The customer receives an activation key.
3	The customer enters the activation key into your application.
4	Your application communicates with Amazon and obtains the user's token. If your application is installed on the user's computer, it also obtains an Access Key ID and Secret Access Key on behalf of the customer.
5	Your application provides the customer's token and the application product token when making Amazon S3 requests on behalf of the customer. If your application is installed on the customer's computer, it authenticates with the customer's credentials.
6	Amazon uses the customer's token and your product token to determine who to bill for the Amazon S3 usage.
7	Once a month, Amazon processes usage data and bills your customers according to the terms you defined.

8	AWS deducts the fixed Amazon DevPay transaction fee and pays you the difference. AWS then separately charges you for the Amazon S3 usage costs incurred by your customers and the percentage-based Amazon DevPay fee.

Additional Information

For information about using, setting up, and integrating with Amazon DevPay, go to Amazon DevPay *Developer Guide.*

Handling Errors

Topics

This section describes REST and SOAP errors and how to handle them.

The REST Error Response

Topics

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

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.

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.

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

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 http://developer.amazonwebservices.com/connect/forum.jspa?forumID=24 or sign up for AWS Premium Support http://aws.amazon.com/premiumsupport/.

Isolate Errors

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.

Server Access Logging

Topics

Important
This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the Amazon S3 Developer Forum.

An Amazon S3 bucket can be configured to create access log records for the requests made against it. An access log record contains details about the request such as the request type, the resource with which the request worked, and the time and date that the request was processed. Server access logs are useful for many applications, because they give bucket owners insight into the nature of requests made by clients not under their control.

By default, server access logs are not collected for a bucket. To learn how to enable server access logging, see Server Access Logging Configuration API (p. 421).

Once logging is enabled for a bucket, available log records are aggregated into log files and delivered to you via an Amazon S3 bucket of your choosing on an hourly basis. For a detailed description of this process, see Delivery of Server Access Logs (p. 423).

For information on how to interpret the contents of log files, see Server Access Log Format (p. 424).

To walk through the process of enabling logging for your bucket, see Setting Up Server Access Logging (p. 428).

Note
There is no extra charge for enabling the server access logging feature on an Amazon S3 bucket, however any log files the system delivers to you will accrue the usual charges for storage (you can delete the log files at any time). No data transfer charges will be assessed for log file delivery, but access to the delivered log files is charged for data transfer in the usual way.

Server Access Logging Configuration API

Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the Amazon S3 Developer Forum.

Each Amazon S3 bucket has an associated XML sub-resource that you can read and write in order to inspect or change the logging status for that bucket. The XML schema for the bucket logging status resource is common across SOAP and REST.

The `BucketLoggingStatus` element has the following structure.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
    <LoggingEnabled>
        <TargetBucket>mylogs</TargetBucket>
        <TargetPrefix>access_log-</TargetPrefix>
        <TargetGrants>
     <Grant>
         <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">
             <EmailAddress>email_address</EmailAddress>
         </Grantee>
         <Permission>permission</Permission>
     </Grant>
        </TargetGrants>
    </LoggingEnabled>
</BucketLoggingStatus>
```

Following is a list of elements that belong to the `BucketLoggingStatus` element.

- *`LoggingEnabled`*

  The presence of this element indicates that server access logging is enabled for the bucket. The absence of this element (and all nested elements) indicates that logging is disabled for the bucket.
- *`TargetBucket`*

  This element specifies the bucket where server access logs will be delivered. You can have your logs delivered to any bucket that you own, including the same bucket that is being logged. You can also configure multiple buckets to deliver their logs to the same target bucket. In this case you should choose a different `TargetPrefix` for each source bucket so that the delivered log files can be distinguished by key.

  Note

  The source and the target buckets must be in the same location. For more information about bucket location constraints, see Buckets and Regions (p. 83)
- *`TargetPrefix`*

  This element lets you specify a prefix for the keys that the delivered log files will be stored under. For information on how the key name for log files is constructed, see Delivery of Server Access Logs (p. 423).
- *`TargetGrants`*

The bucket owner is automatically granted FULL_CONTROL to all logs delivered to the bucket. This optional element enables you grant access to others. Any specified TargetGrants are added to the default ACL. For more information about ACLs, see Access Control Lists (p. 8).

To enable server access logging, Set or `PUT` a `BucketLoggingStatus` with a nested `LoggingEnabled` element. To disable server access logging, Set or `PUT` an empty `BucketLoggingStatus` element.

In REST, the address of the BucketLoggingStatus resource for a bucket 'mybucket' is `http://s3.amazonaws.com/`*`mybucket`*`?logging`. The `PUT` and `GET` methods are valid for this resource. For example, the following request fetches the `BucketLoggingStatus` resource for mybucket.

```
GET ?logging HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Wed, 25 Nov 2009 12:00:00 GMT
Authorization: AWS YOUR_AWS_ACCESS_KEY_ID:YOUR_SIGNATURE_HERE

HTTP/1.1 200 OK
Date: Wed, 25 Nov 2009 12:00:00 GMT
Connection: close
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>mybucketlogs</TargetBucket>
    <TargetPrefix>mybucket-access_log-/</TargetPrefix>
        <TargetGrants>
     <Grant>
         <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">
             <EmailAddress>user@company.com</EmailAddress>
         </Grantee>
         <Permission>READ</Permission>
     </Grant>
        </TargetGrants>

  </LoggingEnabled>
</BucketLoggingStatus>
```

In SOAP, you can work with BucketLoggingStatus resource using the SOAPSetBucketLoggingStatus and SOAPGetBucketLoggingStatus operations.

Amazon S3 checks the validity of the proposed `BucketLoggingStatus` when you try to Set or `PUT` to it. If the `TargetBucket` does not exist, is not owned by you, or does not have the appropriate grants, you will receive the `InvalidTargetBucketForLogging` error. If your proposed `BucketLoggingStatus` document is not well-formed XML or does not match our published schema, you will receive the `MalformedXMLError`.

BucketLoggingStatus Changes Take Effect Over Time

Changes to the logging status for a bucket are visible in the configuration API immediately, but they 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. Or, 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 will eventually take effect without any further action on your part.

Delivery of Server Access Logs

Important
This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the Amazon S3 Developer Forum.

Server access logs are written to the bucket of your choice, which can be the bucket from which the logs originate or a different bucket. If you choose a different bucket, it must have the same owner as the source bucket. Otherwise, no logs will be delivered.

Note
The source and the target buckets must be in the same location. For more information about bucket location constraints, see Buckets and Regions (p. 83).

When a log file is delivered to the target bucket, it is stored under a key in the following format.

```
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.

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 TargetPrefix component of the key is a string provided by the bucket owner using the logging configuration API. For more information, see Server Access Logging Configuration API (p. 421).

The UniqueString component of the key carries no meaning and should be ignored by log processing software.

The system does not delete old log files. If you do not want server logs to accumulate, you must delete them yourself. To do so, use the List operation with the `prefix` parameter to locate old logs to delete. For more information, see Listing Object Keys (p. 226).

Access Control Interaction

Log files will be written to the target bucket under the identity of a member of the `http://acs.amazonaws.com/groups/s3/LogDelivery` group. These writes are subject to the usual access control restrictions. Therefore, logs will not be delivered unless the access control policy of the target bucket grants the log delivery group `WRITE` access. To ensure log files are delivered correctly, the log delivery group must also have `READ_ACP` permission on the target bucket. For more information about access control lists and groups, see Access Control (p. 277). For more information about correctly configuring your target bucket's access control policy, see the Setting Up Server Access Logging (p. 428).

Log files created in the target bucket have an access control list entry that consists of a FULL_CONTROL grant to the bucket owner and grants to any users specified through the *`TargetGrants`* element.

Best Effort Server Log Delivery

The server access logging feature is designed for best effort. You can expect that most requests against a bucket that is properly configured for logging will result in a delivered log record, and that most log records will be delivered within a few hours of the time that they were recorded.

However, the server logging feature is offered on a best-effort basis. 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 the bucket owner an idea of the nature of traffic against his or her bucket. It is not meant to be a complete accounting of all requests.

Usage Report Consistency

It follows from the best-effort nature of the server logging feature that the usage reports available at the AWS portal might include usage that does not correspond to any request in a delivered server log.

Server Access Log Format

Important
This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the Amazon S3 Developer Forum.

The log files consist of a sequence of new-line delimited log records. Log records appear in no particular order. Each log record represents one request and consists of space delimited fields described in the following table.

Field Name	Example Entry	Notes
Bucket Owner	`314159b66967d86f031c7249d1d9a8024` `9109428335cd0ef1cdc487b4566cb1b`	The canonical user id of the owner of the source bucket.
Bucket	`mybucket`	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.
Time	`[04/Aug/2006:22:34:02 +0000]`	The time at which the request was received. The format, using `strftime()` terminology, is `[%d/%b/%Y:%H:%M:%S %z]`
Remote IP	`72.21.206.5`	The apparent Internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.

Field Name	Example Entry	Notes
Requester	`314159b66967d86f031c7249d1d9a80` `249109428335cd0ef1cdc487b4566cb1b`	The canonical user id of the requester, or the string "Anonymous" for unauthenticated requests. This identifier is the same one used for access control purposes.
Request ID	`3E57427F33A59F07`	The request ID is a string generated by Amazon S3 to uniquely identify each request.
Operation	`SOAP.CreateBucket` or `REST.PUT.OBJECT`	Either `SOAP.`*`operation`*, `REST.`*`HTTP_method`*`.`*`resource_type`* or *WEBSITE.HTTP_method.resource_type*
Key	`/photos/2006/08/puppy.jpg`	The "key" part of the request, URL encoded, or "-" if the operation does not take a key parameter.
Request-URI	`"GET /mybucket/photos/2006/08/ puppy.jpg?x-foo=bar"`	The Request-URI part of the HTTP request message.
HTTP status	`200`	The numeric HTTP status code of the response.
Error Code	`NoSuchBucket`	The Amazon S3 Error Code (p. 417), or "-" if no error occurred.
Bytes Sent	`2662992`	The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.
Object Size	`3462992`	The total size of the object in question.
Total Time	`70`	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.

Field Name	Example Entry	Notes
Turn-Around Time	`10`	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.
Referrer	`"http://www.amazon.com/webservices"`	The value of the HTTP Referrer header, if present. HTTP user-agents (e.g. browsers) typically set this header to the URL of the linking or embedding page when making a request.
User-Agent	`"curl/7.15.1"`	The value of the HTTP User-Agent header.
Version Id	3HL4kqtJvjVBH40Nrjfkd	The version ID in the request, or "-" if the operation does not take a *`versionId`* parameter.

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.

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 will ignore query-string parameters that begin with "x-", but will include 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/mybucket/photos/2006/08/puppy.jpg?x-user=johndoe" will work the same as the same request for "s3.amazonaws.com/mybucket/photos/2006/08/puppy.jpg", except that the "x-user=johndoe" string will be included in the `Request-URI` field for the associated log record. This functionality is available in the REST interface only.

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.

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 table describes the fields in the record that relate to the `GET` part of the copy operation.

Field Name	Example Entry	Notes
Bucket Owner	`314159b66967d86f031c7249d1d9a8024` `9109428335cd0ef1cdc487b4566cb1b`	The owner of the bucket that stores the object being copied.
Bucket	`mybucket`	The name of the bucket that stores the object being copied.
Time	`[04/Aug/2006:22:34:02 +0000]`	The time at which the request was received. The format, using `strftime()` terminology, is `[%d/%B/%Y:%H:%M:%S %z]`
Remote IP	`72.21.206.5`	The apparent Internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.
Requester	`314159b66967d86f031c7249d1d9a80` `249109428335cd0ef1cdc487b4566cb1b`	The canonical user id of the requester, or the string "Anonymous" for unauthenticated requests. This identifier is the same one used for access control purposes.
Request ID	`3E57427F33A59F07`	The request ID is a string generated by Amazon S3 to uniquely identify each request.
Operation	`REST.COPY.OBJECT_GET`	Either `SOAP.`*`operation`*, `REST.`*`HTTP_method`*`.`*`resource_type`* or *`WEBSITE.HTTP_method.resource_type`*
Key	`/photos/2006/08/puppy.jpg`	The "key" of the object being copied or "-" if the operation does not take a key parameter.
Request-URI	`"GET /mybucket/photos/2006/08/` `puppy.jpg?x-foo=bar"`	The Request-URI part of the HTTP request message.
HTTP status	`200`	The numeric HTTP status code of the `GET` portion of the copy operation.
Error Code	`NoSuchBucket`	The Amazon S3 Error Code (p. 417), of the `GET` portion of the copy operation or "-" if no error occurred.
Bytes Sent	`2662992`	The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.

Field Name	Example Entry	Notes
Object Size	`3462992`	The total size of the object in question.
Total Time	`70`	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.
Turn-Around Time	`10`	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.
Referrer	`"http://www.amazon.com/webservices"`	The value of the HTTP Referrer header, if present. HTTP user-agents (e.g. browsers) typically set this header to the URL of the linking or embedding page when making a request.
User-Agent	`"curl/7.15.1"`	The value of the HTTP User-Agent header.
Version Id	3HL4kqtJvjVBH40Nrjfkd	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.

Setting Up Server Access Logging

Important
This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the Amazon S3 Developer Forum.

The Amazon S3 server access logging feature lets you generate access log files for buckets that you own. These log files are delivered to you by writing them into a (possibly different) bucket that you own. Once delivered, the access logs are ordinary objects that you can read, list or delete at your convenience.

These instructions assume that you want to enable server access logging on one of your pre-existing buckets, and that you want to have those logs delivered into a new bucket you will create just for logging. We suppose that the bucket you want to log access to is called 'mybucket' and the new bucket you will create to hold your access logs is called 'mylogs'. This makes 'mybucket' the source bucket for logging and 'mylogs' the target bucket for logging. Whenever you see 'mybucket' or 'mylogs' in the example, replace them with the name of your bucket that you want to log, and the bucket you want to store your access logs, respectively.

This tutorial makes use of s3curl (go to s3curl.pl sample program) to work with the Amazon S3 REST API. Make sure you use the most recent version of s3curl, as it has been updated to support this tutorial. After invoking s3curl, always check for a `200 OK` HTTP response. If you get some other response code, refer to the XML error response which likely contains information about what went wrong.

Preparing the Target Bucket

To prepare the target bucket

1. First, decide if you want your logs delivered to an existing bucket, or if you want to create a new bucket just for access log files. Following is a command that creates a new target bucket for logging. Notice the canned ACL argument that grants the system permission to write log files to this bucket.

   Note
   The source and the target buckets must be in the same location. For more information about bucket location constraints, see Buckets and Regions (p. 83)

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
-acl log-delivery-write --put /dev/null -- -s -v https://s3.amazonaws.com/my
logs
```

2. If you just created a new bucket for logging, skip to the next section. Otherwise, to have your access logs files delivered to an existing bucket, you must modify the access control policy of that bucket by hand. Fetch the ?acl sub-resource of the target bucket and save it to a local file:

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
- -s -v 'https://s3.amazonaws.com/mylogs?acl' > mylogs.acl
```

3. Now open the local copy of the logging resource in your favorite text editor and insert a new `<Grant>` element to the `<AccessControlList>` section that gives the log delivery group `WRITE` and `READ_ACP` permission to your bucket.

```
<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>
```

4. Finally, apply the modified access control policy by writing it back to Amazon S3.

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
-put mylogs.acl -- -s -v 'https://s3.amazonaws.com/mylogs?acl'
```

Enabling Server Access Logging on the Source Bucket

Now that the target bucket can accept log files, we'll update the `?logging` sub-resource of the source bucket to turn on server access logging. Remember that you must be the bucket owner to read or write this resource.

Fetch the `?logging` sub-resource for modification using the command shown in the following example.

Example

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY --
-s -v 'https://s3.amazonaws.com/mybucket?logging' > mybucket.logging
```

Open mybucket.logging in your favorite text editor and uncomment the `<LoggingSettings>` section. Replace the contents of the `<TargetBucket>` and `<TargetPrefix>` with 'mylogs' and 'mybucket-access_log-' respectively.

Additionally, to grant users access to log files within the bucket, you can specify one or more users in the `<TargetGrants>` section, You can specify users through their e-mail address (EmailAddress) or canonical user ID (CanonicalUser). Permissions include READ, WRITE, and FULL_CONTROL. The result should be similar to the following.

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">

    <LoggingEnabled>

        <TargetBucket>mylogs</TargetBucket>

        <TargetPrefix>mybucket-access_log-/</TargetPrefix>

            <TargetGrants>

         <Grant>

         <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">

             <EmailAddress>user@company.com</EmailAddress>

         </Grantee>

         <Permission>READ</Permission>

     </Grant>

        </TargetGrants>

    </LoggingEnabled>

</BucketLoggingStatus>
```

Note
For general information about authentication, see Access Control (p. 277).

Now apply your modifications by writing the document back to the `?logging` sub-resource in Amazon S3.

Example

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
-put mybucket.logging -- -s -v 'https://s3.amazonaws.com/mybucket?logging'
```

You can confirm your changes by fetching the `?logging` sub-resource and comparing it to what you just wrote.

Server access logging should now be enabled. Make a few requests against the source bucket now, and your access logs should begin to be delivered to the target bucket within the next few hours.

Disabling Server Logging for a Bucket

Fetch, modify, and apply the `?logging` sub resource in the same way as described in the preceding procedure, except use your text editor to remove the `<LoggingEnabled>` element.

Note
Log changes do not take effect immediately; logs will be delivered for a while after disabling logging.

Using the AWS SDKs and Explorers

Topics

You can use AWS SDKs when developing applications with Amazon S3. The AWS SDKs for Java, PHP, .NET, and Ruby wrap the underlying REST API, simplifying your programming tasks. Mobile SDKs are also available for building connected mobile applications using Amazon Web Services. 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 samples provided in this guide.

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.

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 plug-in for Eclipse for Java IDE that makes it easier for developers to develop, debug, and deploy Java applications using Amazon Web Services. The easy to use GUI interface enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as manage your buckets and objects, set IAM policies, while developing applications, all from within the context of Eclipse for Java IDE. For set up instructions, go to Setting Up the AWS Toolkit for Eclipse. For examples of using Amazon S3 using the explorer, go to Viewing and Editing Amazon S3 Buckets.

AWS Toolkit for Visual Studio

The 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 interface enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as manage your buckets and objects, set IAM policies, while developing applications, all from within the context of Visual Studio. For set up instructions, go to Setting Up the AWS Toolkit for Visual Studio. For examples of using Amazon S3 using the explorer, go to Using Amazon S3 from AWS Explorer.

AWS SDKs

You can download only the SDKs. For information about downloading the SDK libraries, go to Sample Code Libraries.

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 API to upload large objects in parts (see Uploading Objects Using Multipart Upload API (p. 167)). The API 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 such as letting you pause and resume multipart uploads, vary part sizes during the upload, or to begin uploads when you do not know the size of the data in advance. If you do not 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 data size reaches 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 using a stream for the source of data, the `TransferManager` class will 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 implementation APIs for Amazon S3 bucket and object operations.
  For example, it provides methods to create buckets, upload objects, get objects, delete objects, and to list keys.
- **com.amazonaws.services.s3.transfer—**Provides the high-level API data upload.
  This high-level API is designed to further simplify uploading objects to Amazon S3. It includes the `TransferManager` class. It is particularly useful when uploading large objects in parts. It also include 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 `ListObjectRequest` class to describe your list keys requests, and the

`InitiateMultipartUploadRequest` and `InitiateMultipartUploadResult` classes when initiating a multipart upload.

For more information about the AWS SDK for Java API, go to AWS SDK for Java API Reference.

Testing the Java Code Examples

The easiest way to get started with the Java code examples is to install the latest AWS Toolkit for Eclipse. For information on installing or updating to the latest version, go to http://aws.amazon.com/eclipse. The following tasks guide you through the creation and testing of the Java code samples provided in this section.

General Process of Creating Java Code Examples

1	Create a new AWS Java project in Eclipse. The project is pre-configured with the AWS SDK for Java and it also includes the AwsCredentials.properties file for your AWS security credentials.
2	Copy the code from the section you are reading to your project.
3	Update the code by providing any required data. For example, if uploading a file, provide the file path and the bucket name.
4	Run the code. Verify that the object is created by using the AWS Management Console. For more information about the AWS Management Console, go to http://aws.amazon.com/console/.

Using the AWS SDK for .NET

Topics

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. 167)). The API 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. 167)) it provides greater control, such as letting you pause and resume multipart uploads, vary part sizes during the upload, or to begin uploads when you do not 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 data size reaches 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 using a stream for the source of data, the `TransferUtility` class will 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 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. 199)).
- **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.

For more information about the AWS SDK for .NET API, go to AWS SDK for .NET Reference.

Testing the .NET Code Examples

The easiest way to get started with the .NET code examples is to install the AWS SDK for .NET. For more information, go to http://aws.amazon.com/sdkfornet. The following tasks guide you through creating and testing the C# code samples provided in this section.

General Process of Creating .NET Code Examples

1	Create a new Visual Studio project using the *AWS Empty Project* template.
2	In the AWS Access Credentials dialog box, provide your AWS credentials.

3	Note that the *AWS Empty Project* template is preconfigured with an `App.config` file for your AWS credentials and the following required references.
	`AWSSDK`
	`System.Configuration`
	Verify `App.config` file has the following keys with your credentials.
	`<configuration>`
	`  <appSettings>`
	`     <add key="AWSAccessKey" value="*** Your access key ID ***"/>`
	`    <add key="AWSSecretKey" value="*** Your secret key ***"/>`
	`  </appSettings>`
	`</configuration>`

4	Replace the code in the project file, `Program.cs`, with the code in the section you are reading.
5	Run the code. Verify that the object is created using the AWS Management Console. For more information about AWS Management Console, go to http://aws.amazon.com/console/.

Using the AWS SDK for PHP

The AWS SDK for PHP provides the API for Amazon S3 bucket and object operations. The API 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 APIs provides greater control over these operations. For example, you can batch your requests and execute them in parallel, or when using the multipart upload API (see Uploading Objects Using Multipart Upload API (p. 167)), you can manage the object parts. Note that these low-level API calls return a response that includes all the Amazon S3 response details.

High-Level API

The high-level APIs are intended to simplify common use cases. For example, for uploading large objects using the low-level API, you must first initialize the multipart upload by calling the `initiate_multipart_upload()`, uploads parts by calling the `upload_part()` method and complete the upload by calling the `complete_multipart_upload()` methods. Instead, you could use the high-level API and upload object by calling the `create_mpu_object()` method. Internally, this API calls the low-level API to initialize the upload, upload object parts and complete the multipart upload. Another example is when enumerating objects in a bucket you can use the high-level API which returns all the keys, regardless of how many objects you have stored in the bucket. If you use the low-level API the response returns only up to 1,000 keys and if you have more than a 1,000 objects in the bucket, the result will be truncated and you will have to manage the response and check for any truncation.

Using the AWS SDK for Ruby

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 Uploading Objects Using Multipart Upload). However, the API for a single operation upload can accept large objects as well 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 AWS Ruby gem. For more information, see the AWS SDK for Ruby Getting Started Guide. Once installed, you can access the API, including the following key classes:

- **AWS::S3—**Represents the interface to Amazon S3 for the Ruby SDK.

  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 `#objects` instance method for accessing the objects in a bucket as well as methods to delete a bucket and return information about a bucket such as the bucket policy.
- **AWS::S3::S3Object—**Represents an Amazon S3 object identified by its key.
  The `S3Object` 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 `S3Object` 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 API Reference.

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 http://aws.amazon.com/sdkforruby/. 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	Create a new AWS Ruby script and add the following lines to the top of the script.
	`#!/usr/bin/env ruby`
	
	`require 'rubygems'`
	`require 'aws-sdk'`
	
	The first line is the interpreter directive and the two `require` statements import two required gems into your script.

2	Use the `AWS::Core::Configuration` class to specify your credentials.
	`AWS.config(`
	`  :access_key_id => '*** Provide your access key ***',`
	`  :secret_access_key => '*** Provide your secret key ***'`
	`)`

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 http://aws.amazon.com/console/.

Appendices

This Amazon S3 developer guide appendix include the following sections.

Topics

Appendix A: The Access Policy Language

Topics

This appendix is for Amazon S3 users who want to write their own access control policies. You don't need to write your own policies if you want to allow access based only on AWS account ID and basic permissions. If you want to explicitly deny access or allow it based on finer conditions (like the time the request comes in or the IP address of the requester), you need to write your own policies and upload them to AWS.

> **Note**
> To write your own policies, you must be familiar with JSON. For more information, go to http://json.org.

The main portion of this appendix includes basic concepts you need to understand, how to write a policy, and the logic AWS uses to evaluate policies and decide whether to give the requester access to the resource. Although most of the information in this appendix is service-agnostic, there are some Amazon S3-specific details you need to know. For more information, see Special Information for Amazon S3 Policies (p. 459).

Overview

Topics

This section describes basic concepts you need to understand to use access policy language to write policies. It also describes the general process for how access control works with the access policy language, and how policies are evaluated.

Key Concepts

The following sections describe the concepts you need to understand to use the access policy language. They're presented in a logical order, with the first terms you need to know at the top of the list.

Permission

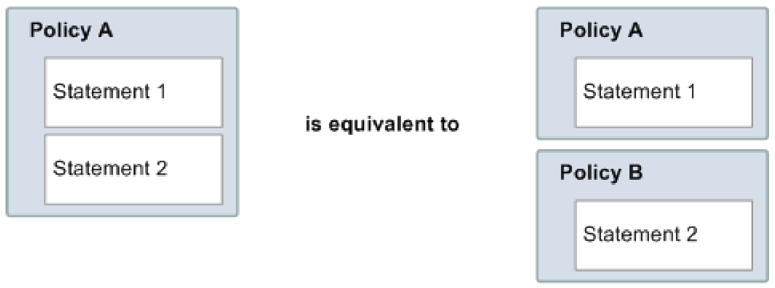
A *permission* is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, *Jane* (A) has permission to *put an object* (B) in *AcmeProductsBucket* (C) as long as an IP address is within a certain range (D). Whenever Jane sends a request to Amazon S3 to put an object in that bucket, the service checks to see if she has permission and if the request satisfies the conditions the bucket owner set forth in the policy.

Statement

A *statement* is the formal description of a single permission, written in the access policy language. You always write a statement as part of a broader container document known as a *policy* (see the next concept).

Policy

A *policy* is a document (written in JSON) that acts as a container for one or more statements. For example, a policy could have two statements in it: one that states that Jane can put objects in a specified bucket, and another that states that Bob cannot put objects into the same bucket. As shown in the following figure, an equivalent scenario would be to have two policies.



Amazon S3 uses the information in the statements (whether they're contained in a single policy or multiple) to determine if someone requesting access to a resource should be granted access.

Issuer

The *issuer* is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS service users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

Principal

The *principal* is the person or persons who receive the permission in the policy. The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you can set the principal to "anyone" (i.e., you can specify a wildcard to represent all people). You might do this, for example, if you want to grant everyone (whether they are authenticated or anonymous) access to a subset of your Amazon S3 resources.

Action

The *action* is the activity the principal has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." Typically, the action is just the operation in the request to AWS. You can specify one or multiple actions in a policy.

For the names of the Amazon S3 actions you can specify in a policy, see Amazon S3 Operations (p. 315).

Resource

The *resource* is the bucket or object the principal is requesting access to. The resource is C in the statement "A has permission to do B to C where D applies." You can specify one or more resources in a policy. For information about to specify a resource in a policy, see Specifying Amazon S3 Resources in Bucket Policies (p. 314).

Conditions and Keys

The *conditions* are any restrictions or details about the permission. The condition is D in the statement "A has permission to do B to C where D applies." The part of the policy that specifies the conditions can be the most detailed and complex of all the parts. Typical conditions are related to:

- Date and time (e.g., the request must arrive before a specific day)
- IP address (e.g., the requester's IP address must be part of a particular CIDR range)

A *key* is the specific characteristic that is the basis for access restriction. For example, the date and time of a request.

You use both *conditions* and *keys* together to express the restriction. The easiest way to understand how you actually implement a restriction is with an example: If you want to restrict access to before May 30, 2010, you use the condition called `DateLessThan`. You use the key called and set it to the value `2010-05-30T00:00:00Z`. AWS defines the conditions and keys you can use. The might also define service-specific keys. For more information about conditions, see Condition (p. 452). For more information about the available keys, see Bucket Keys in Amazon S3 Policies (p. 317) and Object Keys in Amazon S3 Policies (p. 320).

Requester

The *requester* is the person who sends a request to an AWS service and asks for access to a particular resource. The requester sends a request to AWS that essentially says: "Will you allow me to do B to C where D applies?"

Evaluation

Evaluation is the process the AWS service uses to determine if an incoming request should be denied or allowed based on the applicable policies. For information about the evaluation logic, see Evaluation Logic (p. 446).

Effect

The *effect* is the result that you want a policy statement to return at evaluation time. You specify this value when you write the statements in a policy, and the possible values are *deny* and *allow*.

For example, you could write a policy that has a statement that *denies* all requests that come from Antarctica (effect=deny given that the request uses an IP address allocated to Antarctica). Alternately, you could write a policy that has a statement that *allows* all requests that *don't* come from Antarctica (effect=allow, given that the request doesn't come from Antarctica). Although the two statements sound like they do the same thing, in the access policy language logic, they are different. For more information, see Evaluation Logic (p. 446).

Although there are only two possible values you can specify for the effect (allow or deny), there can be three different results at policy evaluation time: *default deny*, *allow*, or *explicit deny*. For more information, see the following concepts and Evaluation Logic (p. 446).

Default Deny

A *default deny* is the default result from a policy in the absence of an allow or explicit deny.

Allow

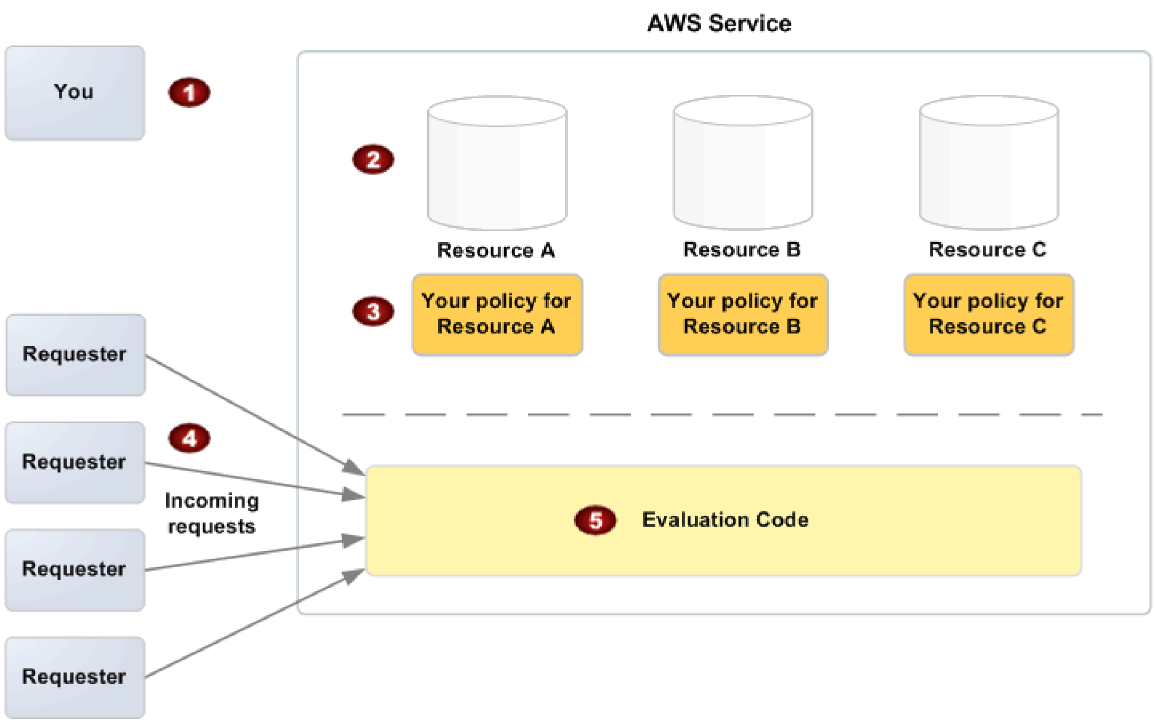
An *allow* results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests if they are received before 1:00 p.m. on April 30, 2010. An allow overrides all default denies, but never an explicit deny.

Explicit Deny

An *explicit deny* results from a statement that has effect=deny, assuming any stated conditions are met. Example: Deny all requests if they are from Antarctica. Any request that comes from Antarctica will always be denied no matter what any other policies might allow.

Architectural Overview

The following figure and table describe the main components that interact to provide access control for your resources.

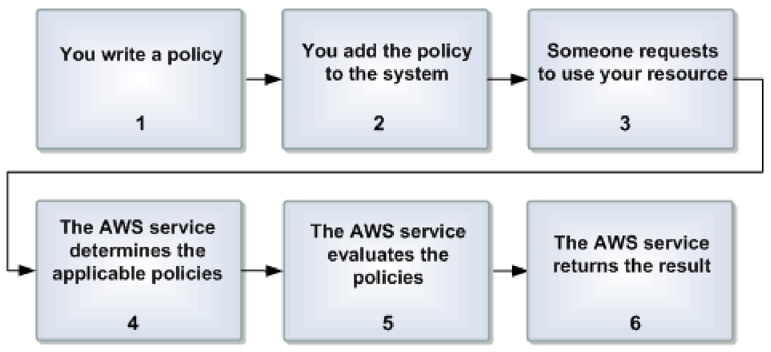


1	You, the resource owner.
2	Your resources (contained within the AWS service; e.g., Amazon S3 buckets or objects).
3	Your policies. In Amazon S3 there is only one policy per bucket. Amazon S3 provides an API that enables to you to upload and manage your bucket policy. For information about the content of the policies, see How to Write a Policy (p. 449).
4	Requesters and their incoming requests to the AWS service.
5	The access policy language evaluation code. This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource. For information about how the service makes the decision, see Evaluation Logic (p. 446).

For the typical process of how the components work together, see Using the Access Policy Language (p. 445).

Using the Access Policy Language

The following figure and table describe the general process of how access control works with the access policy language.


You write a policy
1
You add the policy to the system
2
Someone requests to use your resource
3
The AWS service determines the applicable policies
4
The AWS service evaluates the policies
5
The AWS service returns the result
6

Process for Using Access Control with access policy language

1	You write a policy for your resource. For example, you write a policy to specify permissions for your Amazon S3 objects. For more information, see How to Write a Policy (p. 449).
2	You upload your policy to AWS. The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon S3 `PUT Bucket policy` action to set a policy on a bucket.
3	Someone sends a request to use your resource. For example, a user sends a request to Amazon S3 to upload an object to a bucket.
4	The AWS service determines which policies are applicable to the request. For example, Amazon S3 looks at all the available Amazon S3 policies and determines which ones are applicable (based on what the resource is, who the requester is, etc.).
5	The AWS service evaluates the policies. For example, Amazon S3 evaluates the policies and determines if the requester is allowed to upload the object to the bucket. For information about the decision logic, see Evaluation Logic (p. 446).
6	The AWS service either denies the request or continues to process it. For example, based on the policy evaluation result, the service either returns an "Access denied" error to the requester or continues to process the request.

Related Topics

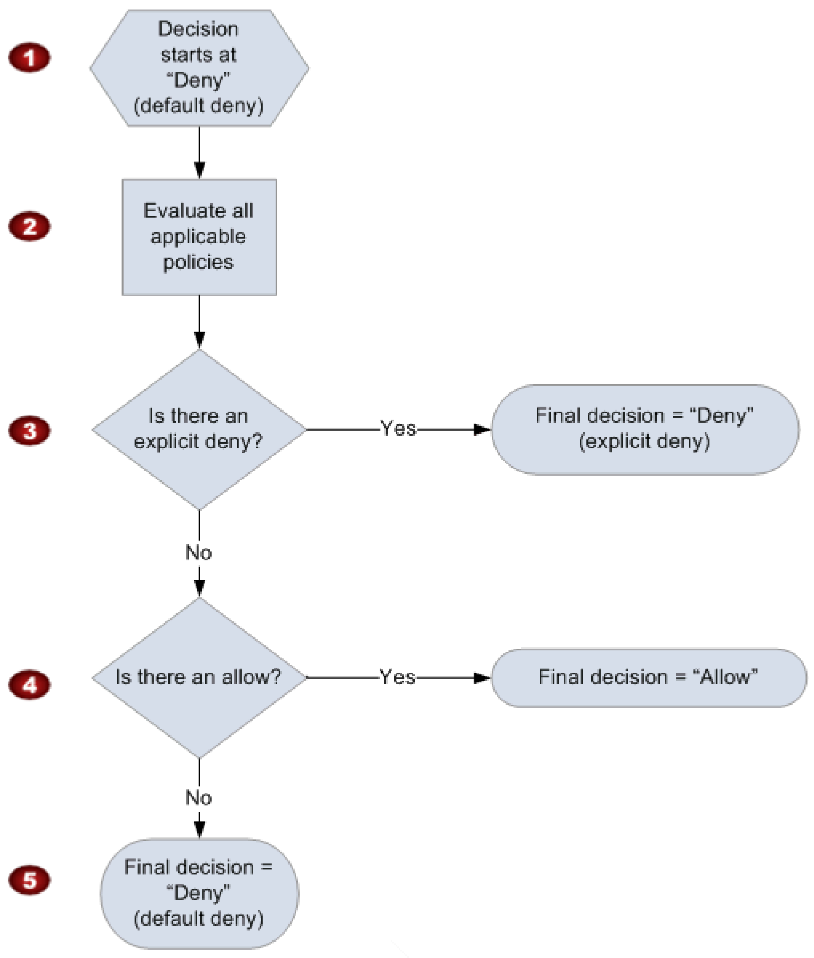
- Architectural Overview (p. 444)

Evaluation Logic

The goal at evaluation time is to decide whether a given request should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied
- An allow overrides any default denies
- An explicit deny overrides any allows
- The order in which the policies are evaluated is not important

The following flow chart and discussion describe in more detail how the decision is made.


1
Decision starts at "Deny" (default deny)
2
Evaluate all applicable policies
3
Is there an explicit deny?
Yes
Final decision = "Deny" (explicit deny)
No
4
Is there an allow?
Yes
Final decision = "Allow"
No
5
Final decision = "Deny" (default deny)

1	The decision starts with a default deny.
2	The enforcement code then evaluates that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important.

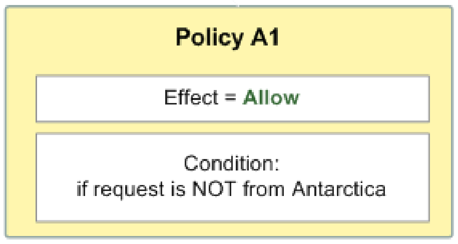
3	In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request. If it finds even one, the enforcement code returns a decision of "deny" and the process is finished (this is an explicit deny; for more information, see Explicit Deny (p. 443)).
4	If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request. If it finds even one, the enforcement code returns a decision of "allow" and the process is done (the service continues to process the request).
5	If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a default deny (for more information, see Default Deny (p. 443)).

The Interplay of Explicit and Default Denials

A policy results in a default deny if it doesn't directly apply to the request. For example, if an account requests to use Amazon S3, but the only policy that applies to the account states that the account can only list the contents of an Amazon S3 bucket, then that policy results in a default deny.

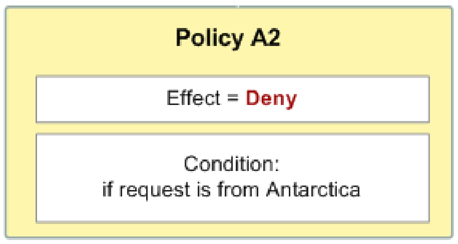
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the *Effect* element in the policy.

For example, let's say you want to prevent requests coming in from Antarctica. You write a policy (called Policy A1) that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the policy.



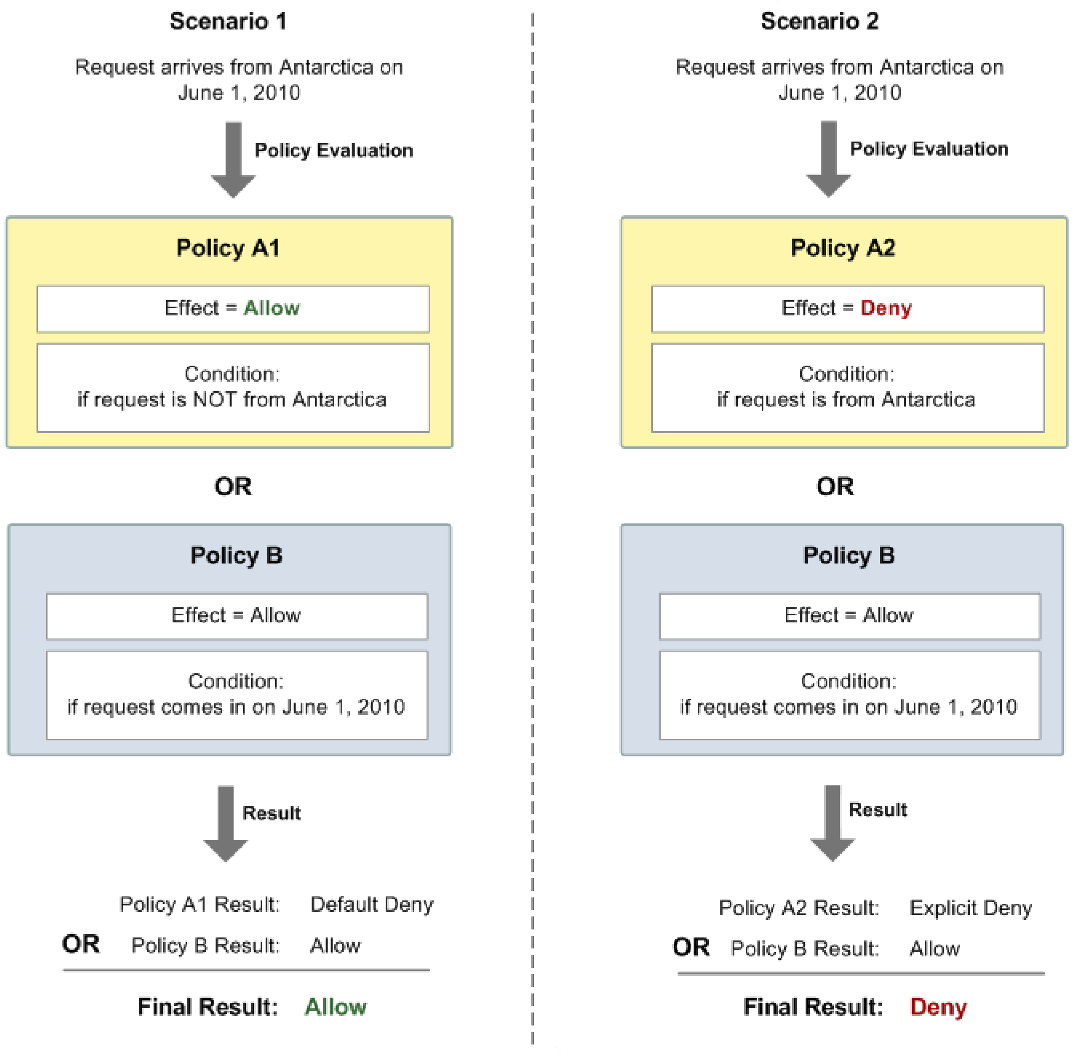
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

You could turn the result into an explicit deny by rewriting the policy (named Policy A2) as in the following diagram. Here, the policy explicitly denies a request if it comes from Antarctica.



If someone sends a request from Antarctica, the condition is met, and the policy's result is therefore an explicit deny.

The distinction between a default deny and an explicit deny is important because a default deny can be overridden by an allow, but an explicit deny can't. For example, let's say there's another policy that allows requests if they arrive on June 1, 2010. How does this policy affect the overall outcome when coupled with the policy restricting access from Antarctica? We'll compare the overall outcome when coupling the date-based policy (we'll call Policy B) with the preceding policies A1 and A2. Scenario 1 couples Policy A1 with Policy B, and Scenario 2 couples Policy A2 with Policy B. The following figure and discussion show the results when a request comes in from Antarctica on June 1, 2010.


Scenario 1
Request arrives from Antarctica on June 1, 2010
Policy Evaluation
Policy A1
Effect = Allow
Condition: if request is NOT from Antarctica
OR
Policy B
Effect = Allow
Condition: if request comes in on June 1, 2010
Result
Policy A1 Result: Default Deny
OR Policy B Result: Allow
Final Result: Allow
Scenario 2
Request arrives from Antarctica on June 1, 2010
Policy Evaluation
Policy A2
Effect = Deny
Condition: if request is from Antarctica
OR
Policy B
Effect = Allow
Condition: if request comes in on June 1, 2010
Result
Policy A2 Result: Explicit Deny
OR Policy B Result: Allow
Final Result: Deny

In Scenario 1, Policy A1 returns a default deny, as described earlier in this section. Policy B returns an allow because the policy (by definition) allows requests that come in on June 1, 2010. The allow from Policy B overrides the default deny from Policy A1, and the request is therefore allowed.

In Scenario 2, Policy B2 returns an explicit deny, as described earlier in this section. Again, Policy B returns an allow. The explicit deny from Policy A2 overrides the allow from Policy B, and the request is therefore denied.

How to Write a Policy

Topics

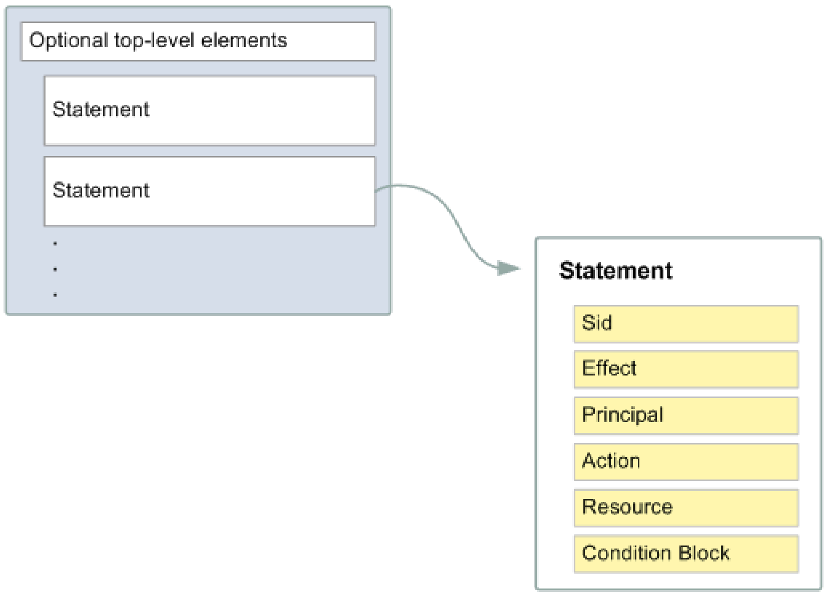
This section describes how to write policies and gives reference information about each policy element.

Basic Policy Structure

Each policy is a JSON document. As illustrated in the following figure, a policy includes:

- Optional policy-wide information (at the top of the document)
- One or more individual *statements*

Each statement includes the core information about a single permission. If a policy includes multiple statements, we apply a logical `OR` across the statements at evaluation time. If multiple policies are applicable to a request, we apply a logical `OR` across the policies at evaluation time.



The information in a statement is contained within a series of *elements*. For information about these elements, see Element Descriptions (p. 449).

Element Descriptions

Topics

This section describes the elements you can use in a policy and its statements. The elements are listed here in the general order you use them in a policy. The `Id`, `Version`, and `Statement` are top-level policy elements; the rest are statement-level elements. JSON examples are provided.

All elements are optional for the purposes of parsing the policy document itself. The order of the elements doesn't matter (e.g., the `Resource` element can come before the `Action` element). You're not required to specify any Conditions in the policy.

Version

The `Version` is the access policy language version. This is an optional element, and currently the only allowed value is `2008-10-17`.

```
"Version":"2008-10-17"
```

Id

The `Id` is an optional identifier for the policy. We recommend you use a UUID for the value, or incorporate a UUID as part of the ID to ensure uniqueness.

Important
Amazon S3, which implements the access policy language might require the *`Id`* element and have uniqueness requirements for it. For service-specific information about writing policies, see Special Information for Amazon S3 Policies (p. 459).

Permission

A permission is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, Jane (A) has permission to receive messages (B) from John's Amazon SQS queue (C), as long as she asks to receive them before midnight on May 30, 2009 (D). Whenever Jane sends a request to Amazon SQS to use John's queue, the service checks to see if she has permission and if the request satisfies the conditions John set forth in the permission.

Statement

The `Statement` is the main element for a statement. It can include multiple elements (see the subsequent sections in this guide).

The `Statement` element contains an array of individual statements. Each individual statement is a distinct JSON block enclosed in curly brackets { }.

```
"Statement":[{...},{...},{...}]
```

Policy

A policy is a document (written in the access policy language) that acts as a container for one or more statements. For example, a policy could have two statements in it: one that states that Jane can read John's object, and another that states that Bob cannot read John's object. An equivalent scenario would be to have two policies, one containing the statement that Jane can read John's object, and another containing the statement that Bob cannot read John's object.

Issuer

The issuer is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS product users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

Sid

The `Sid` (statement ID) is an optional identifier you provide for the policy statement. Essentially it is just a sub-ID of the policy document's ID.

Important
The Amazon S3 implementation of the access policy language might require this element and have uniqueness requirements for it. For service-specific information about writing policies, see Special Information for Amazon S3 Policies (p. 459).

```
"Sid" : "1"
```

Effect

The `Effect` is a required element that indicates whether you want the statement to result in an allow or an explicit deny (for more information, see Explicit Deny (p. 443)).

Valid values for `Effect` are `Allow` and `Deny`.

```
"Effect":"Allow"
```

Principal

The `Principal` is the person or persons who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). You can specify multiple principals, or a wildcard (*) to indicate all possible users. You can view your account ID by logging in to your AWS account at http://aws.amazon.com and clicking **Account Activity**.

In JSON, you use `"AWS":` as a prefix for the principal's AWS account ID.

Action

The `Action` is the specific type or types of access allowed or denied (for example, read or write). You can specify multiple values for this element. The values are free-form but must match values the AWS service expects (for more information, see Special Information for Amazon S3 Policies (p. 459)). You can use a wildcard (*) to give the principal access to all the actions the specific AWS service lets you share with other developers.

NotAction

The `NotAction` element is useful if you want to make an exception to a list of actions. You could use this, for example, if you want your users to be able to use only the `GET Object`.

The following example refers to all actions *other* than `GET Object`. You would use this in a policy with `"Effect":"Deny"` to keep users from accessing any other actions.

```
"NotAction":"s3:GetObject"
```

Resource

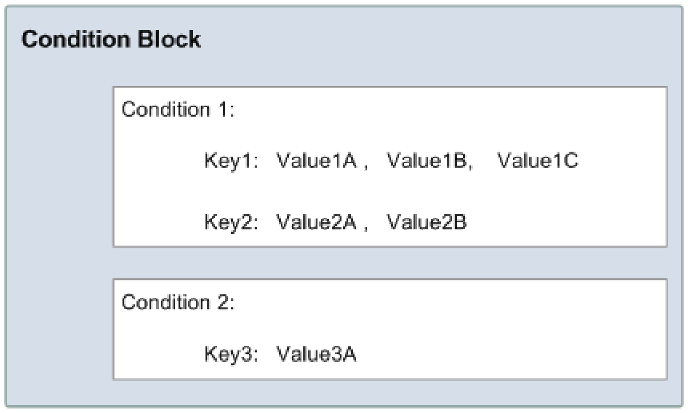
The `Resource` is the object or objects the policy covers. The value can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. The values are free-form, but must follow the format the AWS service expects.

Condition

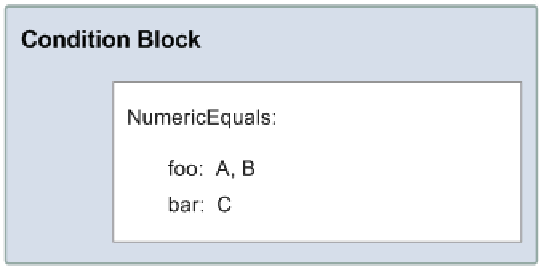
This section describes the `Condition` element and the information you can use inside the element.

The Condition Block

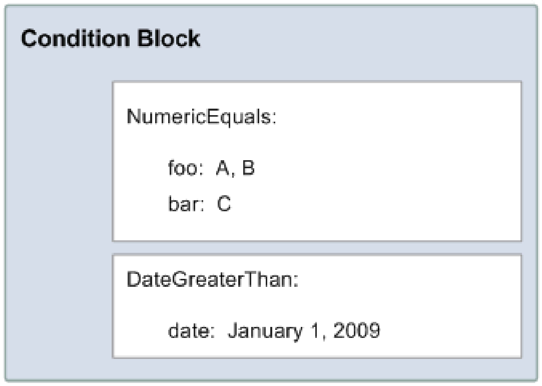
The `Condition` element is the most complex part of the policy statement. We refer to it as the *condition block*, because although it has a single `Condition` element, it can contain multiple conditions, and each condition can contain multiple key-value pairs. The following figure illustrates this. Unless otherwise specified for a particular key, all keys can have multiple values.



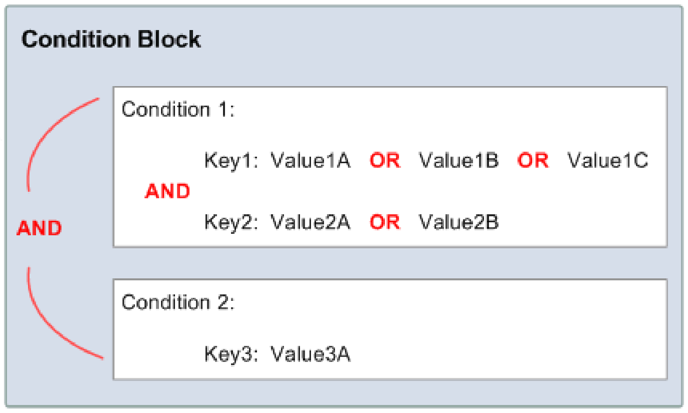
When creating a condition block, you specify the name of each condition, and at least one key-value pair for each condition. AWS defines the conditions and keys you can use (they're listed in the subsequent sections). An example of a condition is `NumericEquals`. Let's say you have a fictional resource, and you want to let John use it only if some particular numeric value *foo* equals either A or B, and another numeric value *bar* equals C. Then you would create a condition block that looks like the following figure.



Let's say you also want to restrict John's access to after January 1, 2009. Then you would add another condition, `DateGreaterThan`, with a date equal to January 1, 2009. The condition block would then look like the following figure.



As illustrated in the following figure, we always apply a logical `AND` to the conditions within a condition block, and to the keys within a condition. We always apply a logical `OR` to the values for a single key. All conditions must be met to return an allow or an explicit deny decision. If a condition isn't met, the result is a default deny.



As mentioned, AWS defines the conditions and keys you can use (for example, one of the keys is `aws:CurrentTime`, which lets you restrict access based on the date and time). The AWS service itself can also define its own service-specific keys. For a list of available keys, see Available Keys (p. 454).

For a concrete example that uses real keys, let's say you want to let John upload an object under the following three conditions:

- The time is after 12:00 noon on 8/16/2010
- The time is before 3:00 p.m. on 8/16/2010
- The request comes from an IP address within the 192.168.176.0/24 range or the 192.168.143.0/24 range

Your condition block has three separate conditions, and all three of them must be met for John to have access to your bucket .

The following shows what the condition block looks like in your policy.

```
"Condition" :  {
      "DateGreaterThan" : {
         "aws:CurrentTime" : "2009-04-16T12:00:00Z"
       }
      "DateLessThan": {
         "aws:CurrentTime" : "2009-04-16T15:00:00Z"
       }
       "IpAddress" : {
          "aws:SourceIp" : ["192.168.176.0/24","192.168.143.0/24"]
      }
}
```

Available Keys

AWS provides a set of common keys supported by all AWS products that adopt the access policy language for access control. These keys are:

- `aws:CurrentTime`–For date/time conditions (see Date Conditions (p. 456))
- `aws:EpochTime`–Number of seconds since epoch.
- `aws:MultiFactorAuthAge`–Key that provides a numeric value indicating how long ago (in seconds) the MFA-validated security credentials making the request were issued using Multi-Factor Authentication (MFA). Unlike other keys, if MFA is not used successfully, this key is not present (see Existence of Condition Keys (p. 457) and Numeric Conditions (p. 455)). For more information, go to Using Multi-Factor Authentication (MFA) Devices with AWS.
- `aws:SecureTransport`–Boolean representing whether the request was sent using SSL (see Boolean Conditions (p. 456))
- `aws:SourceIp`–The requester's IP address, for use with IP address conditions (see IP Address (p. 457))
- `aws:UserAgent`–Information about the requester's client application, for use with string conditions (see String Conditions (p. 455))
- `aws:Referer`–Same as the HTTP *`referer`* field.

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

> **Note**
> If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

Each AWS service that uses the access policy language might also provide service-specific keys. For a list of any service-specific keys you can use, see Special Information for Amazon S3 Policies (p. 459).

Condition Types

These are the general types of conditions you can specify:

- String
- Numeric
- Date and time
- Boolean
- IP address
- Amazon Resource Name (ARN)
- Existence of condition keys

String Conditions

String conditions let you constrain using string matching rules. The actual data type you use is a string.

Condition	Description
`StringEquals`	Strict matching Short version: `streq`
`StringNotEquals`	Strict negated matching Short version: `strneq`
`StringEqualsIgnoreCase`	Strict matching, ignoring case Short version: `streqi`
`StringNotEqualsIgnoreCase`	Strict negated matching, ignoring case Short version: `strneqi`
`StringLike`	Loose case-insensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: `strl`
`StringNotLike`	Negated loose case-insensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: `strnl`

Numeric Conditions

Numeric conditions let you constrain using numeric matching rules. You can use both whole integers or decimal numbers. Fractional or irrational syntax is not supported.

Condition	Description
`NumericEquals`	Strict matching Short version: `numeq`
`NumericNotEquals`	Strict negated matching Short version: `numneq`

Condition	Description
`NumericLessThan`	"Less than" matching Short version: `numlt`
`NumericLessThanEquals`	"Less than or equals" matching Short version: `numlteq`
`NumericGreaterThan`	"Greater than" matching Short version: `numgt`
`NumericGreaterThanEquals`	"Greater than or equals" matching Short version: `numgteq`

Date Conditions

Date conditions let you constrain using date and time matching rules. You must specify all date/time values with one of the W3C implementations of the ISO 8601 date formats (for more information, go to http://www.w3.org/TR/NOTE-datetime). You use these conditions with the `aws:CurrentTime` key to restrict access based on request time.

> **Note**
> Wildcards are not permitted for date conditions.

Condition	Description
`DateEquals`	Strict matching Short version: `dateeq`
`DateNotEquals`	Strict negated matching Short version: `dateneq`
`DateLessThan`	A point in time at which a key stops taking effect Short version: `datelt`
`DateLessThanEquals`	A point in time at which a key stops taking effect Short version: `datelteq`
`DateGreaterThan`	A point in time at which a key starts taking effect Short version: `dategt`
`DateGreaterThanEquals`	A point in time at which a key starts taking effect Short version: `dategteq`

Boolean Conditions

Condition	Description
`Bool`	Strict Boolean matching

IP Address

IP address conditions let you constrain based on IP address matching rules. You use these with the `aws:SourceIp` key. The value must be in the standard CIDR format (for example, 10.52.176.0/24). For more information, go to RFC 4632.

Condition	Description
`IpAddress`	Whitelisting based on the IP address or range
`NotIpAddress`	Blacklisting based on the IP address or range

Existence of Condition Keys

Use a Null condition to check if a condition key is present at the time of authorization. In the policy statement, use either true (the key doesn't exist) or false (the key exists and its value is not null). You can use this condition to determine if a user has authenticated with MFA. For example, the following condition states that MFA authentication must exist (be not null) for the user to use the Amazon EC2 API.

```
{
"Statement":[{
       "Action":["ec2:*"],
       "Effect":"Allow",
       "Resource":["*"],
        "Condition":{"Null":{"aws:MultiFactorAuthAge":"false"}
       }
     }
  ]
}
```

Amazon Resource Name (ARN)

Amazon Resource Name (ARN) conditions let you constrain based on ARN matching rules. The actual data type you use is a string.

Condition	Description
`ArnEquals`	Strict matching for ARN
`ArnNotEquals`	Strict negated matching for ARN
`ArnLike`	Loose case-insensitive matching of the ARN. Each of the six colon-delimited components of the ARN is checked separately and each can include a multi-character match wildcard (*) or a single-character match wildcard (?).
`ArnNotLike`	Negated loose case-insensitive matching of the ARN. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string.

```
{
   "Statement":[{
      "Effect": "Allow",
      "Principal": {
         "AWS": "210987654321"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:01234567891:your_queue_xyz",
      "Condition" : {
         "ArnEquals" : {
          "aws:SourceArn":"arn:aws:sns:us-east-1:123456789012:your_special_top
ic_1"}
            }
      }
   ]
}
```

Supported Data Types

This section lists the set of data types the access policy language supports. The language doesn't support all types for each policy element (for the supported data types for each element, see Element Descriptions (p. 449)).

The access policy language supports the following data types:

- Strings
- Numbers (Ints and Floats)
- Boolean
- Null
- Lists
- Maps
- Structs (which are just nested Maps)

The following table maps each data type to the serialization. Note that all policies must be in UTF-8. For information about the JSON data types, go to RFC 4627.

Type	JSON
String	String
Integer	Number
Float	Number
Boolean	true false
Null	null
Date	String adhering to the W3C Profile of ISO 8601
IpAddress	String adhering to RFC 4632
List	Array
Object	Object

Special Information for Amazon S3 Policies

The following list describes the restrictions on Amazon S3 policies:

- The maximum size of a policy is 20 KB
- The value for `Resource` must be prefixed with the bucket name or the bucket name and a path under it (bucket/*). If only the bucket name is specified, without the trailing /*, the policy applies to the bucket.
- Each policy must have a unique policy ID (*`Id`*)
- Each statement in a policy must have a unique statement ID (*`sid`*)
- Each policy must cover only a single bucket and resources within that bucket (when writing a policy, don't include statements that refer to other buckets or resources in other buckets)

Appendix B: Using the SOAP API

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

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:

- *`AWSAccessKeyId:`* The AWS Access Key ID of the requester
- *`Timestamp:`* The current time on your system
- *`Signature:`* The signature for the request

For information about endpoints, see Request Endpoints (p. 13).

Authenticating SOAP Requests

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:

- *`AWSAccessKeyId:`* 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. 11).

- `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>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <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. For more information, see the sample .NET SOAP libraries for an example of how to do this.

Setting Access Policy with SOAP

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 Access Control (p. 277). 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>&quot828ef3fdfa96f00ad9f27c383fc9ac7f&quot</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.

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 S3 Getting Started Guide	The Getting Started Guide provides a quick tutorial of the service based on a simple use case.
Amazon S3 API Reference	The API Reference describes Amazon S3 operations in detail.
Amazon S3Technical FAQ	The FAQ covers the top 20 questions developers have asked about this product.
Amazon S3 Release Notes	The Release Notes give a high-level overview of the current release. They specifically note any new features, corrections, and known issues.
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 S3without programming.
Discussion Forums	A community-based forum for developers to discuss technical questions related to Amazon Web Services.
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 Calculator	Use the AWS calculator to estimate your monthly charges for using AWS services.
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.

Resource	Description
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.

Document History

This document history is associated with the 2006-03-01 release of Amazon S3. This guide was last updated on December 31, 2012.

The following table describes the important changes since the last release of the *Amazon S3 Developer Guide*.

Change	Description	Date
Console support for Requester Pays	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. 95).	In this release.
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. 377).	27 December 2012
Console revision	Amazon S3 console has been updated. The documentation topics that refer to the console have been revised accordingly.	14 December 2012

Change	Description	Date
Support for Archiving Data to Amazon Glacier	Amazon S3 now support a storage option that enables you to utilize Amazon 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 Amazon 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. 106).	13 November 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 Configuring a Web Page Redirect (p. 388). For information about hosting websites, see Hosting a Static Website on Amazon S3 (p. 377).	04 October 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 Enabling Cross-Origin Resource Sharing (p. 124).	31 August 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 Cost Allocation Tagging (p. 99).	21 August 2012
Support for MFA-protected API access in bucket policies	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. To enforce MFA authentication, Amazon S3 now supports the `aws:MultiFactorAuthAge` key in a bucket policy. For an example bucket policy, see Adding Bucket Policy to Require MFA Authentication (p. 311).	10 July 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. For more information, see Object Expiration (p. 114).	27 December 2011
New Region supported	Amazon S3 now supports the South America (Sao Paulo) Region. For more information, see Buckets and Regions (p. 83).	14 December 2011.

Change	Description	Date
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. 236).	07 December 2011
New Region supported	Amazon S3 now supports the US West (Oregon) Region. For more information, see Buckets and Regions (p. 83).	08 November 2011
Documentation Update	Documentation bug fixes.	08 November 2011
Documentation Update	In addition to documentation bug fixes, this release includes the following enhancements: • New server-side encryption sections using the AWS SDK for PHP (see Specifying Server-Side Encryption Using the AWS SDK for PHP (p. 344)) and the AWS SDK for Ruby (see Specifying Server-Side Encryption Using the AWS SDK for Ruby (p. 346)). • New section on creating and testing Ruby samples (see Using the AWS SDK for Ruby (p. 437)).	17 October 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.	04 October 2011
Documentation Update	In addition to documentation bug fixes, this release includes the following enhancements: • Added Ruby and PHP samples to the Making Requests (p. 11) section. • Added sections describing how to generate and use pre-signed URLs. For more information, see Share an Object with Others (p. 151) and Uploading Objects Using Pre-Signed URLs (p. 201). • Updated an existing section to introduce AWS Explorers for Eclipse and Visual Studio. For more information, see Using the AWS SDKs and Explorers (p. 433).	22 September 2011

Change	Description	Date
Support for sending requests using temporary security credentials	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 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. For more information, see Making Requests (p. 11). For more information about IAM support for temporary security credentials, go to Granting Temporary Access to Your AWS Resources.	3 August 2011
Multipart Upload API extended to enable copying objects up to 5 TB	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, `Upload Part (Copy)`. You can use this multipart upload operation to copy objects up to 5 TB in size. For more information, see Copying Objects (p. 208). For conceptual information about multipart upload API, see Uploading Objects Using Multipart Upload API (p. 167).	21 June 2011
SOAP API calls over HTTP disabled	To increase security, SOAP API calls over HTTP are disabled. Authenticated and anonymous SOAP requests must be sent to Amazon S3 using SSL.	6 June 2011
IAM enables cross-account delegation	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. For more information, go to Enabling Cross-Account Access in *Using Identity and Access Management*. For more information on specifying principals in a bucket policy, see Specifying Principals in Bucket Policies (p. 314).	6 June 2011
New link	This service's endpoint information is now located in the Amazon Web Services General Reference. For more information, go to Regions and Endpoints in Amazon Web Services General Reference.	1 March 2011

Change	Description	Date
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. 377).	17 February 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. 142).	14 January 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. 167).	9 December 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. 167).	10 November 2010
Canonical ID support in bucket policies	You can now specify canonical IDs in bucket policies. For more information, see Specifying Principals in Bucket Policies (p. 314) and for a sample, see Granting Permission, Using Canonical ID, to a CloudFront Origin Identify (p. 311)	17 September 2010
Amazon S3 works with IAM	This service now integrates with AWS Identity and Access Management (IAM). For more information, go to Integrating with Other AWS Products in Using AWS Identity and Access Management.	2 September 2010
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 (SNS) topic when Amazon S3 detects a key event on a bucket. For more information, see Setting Up Notification of Bucket Events (p. 400).	14 July 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 Bucket Policies (p. 7).	6 July 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. 59).	9 June 2010

Change	Description	Date
New endpoint for EU (Ireland)	Amazon S3 now provides an endpoint for EU (Ireland): `http://s3-eu-west-1.amazonaws.com`.	9 June 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 S3 Console User Guide*.	9 June 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. 7).	12 May 2010
New Region supported	Amazon S3 now supports the Asia Pacific (Singapore) Region. For more information, see Buckets and Regions (p. 83).	28 April 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. 8) and Using Versioning (p. 355).	8 February 2010
New Region supported	Amazon S3 now supports the US-West (Northern California) Region. The new endpoint for requests to this Region is `s3-us-west-1.amazonaws.com`. For more information, see Buckets and Regions (p. 83).	2 December 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 APIs 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 and Explorers (p. 433).	11 November 2009

Glossary

100-continue	A method that enables a client to see if a server can accept a request before actually sending it. For large `PUT`s, this can save both time and bandwidth charges.
account	AWS account associated with a particular developer.
authentication	The process of proving your identity to the system.
bucket	A container for objects stored in Amazon S3. Every object is contained within 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`
canned access policy	A standard access control policy that you can apply to a bucket or object. Options include: private, public-read, public-read-write, authenticated-read.
canonicalization	The process of converting data into a standard format that will be recognized by a service such as Amazon S3.
consistency model	The method through which Amazon S3 achieves high availability, which involves replicating data across multiple servers within Amazon's data centers. After a "success" is returned, your data is safely stored. However, information about the changes might not immediately replicate across Amazon S3.
delete marker	A Delete Marker is an object with a key and version ID but it has no content. Amazon S3 inserts Delete Markers automatically into buckets when a `DELETE` request is executed on an object in a bucket where Versioning is enabled.
default deny	A permission that defaults to DENY.
key	The unique identifier for an object within a bucket. Every object in a bucket has exactly one key. Since a bucket and key together uniquely identify each object, Amazon S3 can be thought of as a basic data map between "bucket + key" and the object itself. Every object in Amazon S3 can be uniquely addressed through the combination of the web service endpoint, bucket name, and key, as in http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl, where "doc" is the name of the bucket, and "2006-03-01/AmazonS3.wsdl" is the key.
metadata	The metadata is a set of name-value pairs that describe the object. These include default metadata such as the date last modified and standard HTTP metadata such as Content-Type. The developer can also specify custom metadata at the time the Object is stored.

null object	A null object is one whose version ID is `null`. Amazon S3 adds a null object to a bucket when version for that bucket is suspended. It is possible to have only one null object for each key in a bucket.
object	The fundamental entities stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3.
part	A contiguous portion of object's data. The Amazon S3 multipart upload feature allows larger objects to be uploaded in parts. These object parts can be uploaded independently and in any order. If transmission of any part fails, that part can be retransmitted without affecting other parts. After all the object parts are uploaded, Amazon S3 assembles these parts and creates the object.
service endpoint	The host and port with which you are trying to communicate within the destination URL. For virtual hosted-style requests, this is `mybucket.s3.amazonaws.com`. For path-style requests, this is `s3.amazonaws.com`
Versioning	Every object in Amazon S3 has a key and a version ID. Objects with the same key but different version IDs can be stored in the same bucket. Versioning is enabled at the bucket layer using `PUT Bucket versioning`.

Index