Object Storage Service

Developer Guide

为了无法计算的价值 | 阿里云

Developer Guide

Document Overview

Alibaba Cloud Object Storage Service (OSS) provides you with the network-based data access services. OSS enables you to store and invoke structured or unstructured datan objects including text files, images, audio, and videos via network connection. Alibaba Cloud OSS provides simple network service interfaces for users to store and retrieve any quantity of data anytime anywhere on the network.

The Guide introduces how to send a request to create a bucket, how to store and retrieve an object and how to manage permissions of a resource. It also describes how to perform access control and identity authentication. Access control is used to determine the users who can access objects and buckets in Alibaba Cloud OSS and the access types. Identity authorization is used to authenticate the identity of a user who tries to access computing services of Alibaba Cloud.

The following table lists the resources you may need when using OSS.

Resource	Description
Alibaba Cloud OSS Quick Start	Describes how to complete basic tasks on the OSS Console.
Alibaba Cloud OSS Developer Guide (this document)	Describes the core concepts, functions, and operation procedure of OSS, as well as examples of how to use APIs and SDKs
Alibaba Cloud OSS Console User Guide	Describes all operations supported by the OSS Console. Using the OSS Console, you can perform partial OSS functions
Alibaba Cloud OSS Best Practice	Describes the application scenarios and configuration practices of OSS
Alibaba Cloud OSS API Reference	Describes the REST API operations supported by OSS and provides related examples
Alibaba Cloud OSS SDK Reference	Describes the SDK development and related parameters based on major languages

Product introduction

What is OSS?

Alibaba Cloud Object Storage Service (OSS) provides you with network-based data access services. OSS enables you to store and invoke structured or unstructured datan objects including text files, images, audio, and videos via network connection.

You can execute basic and advanced OSS tasks directly on the OSS Console without coding, in addition to using the Alibaba Cloud software development kit (SDK) or calling the REST API in an application program.

Introduction to basic OSS concepts

This section will introduce the basic concepts of the OSS product for you to better understand the product.

Bucket

A bucket is a container of objects. All objects must belong to a bucket. You can set and modify the attributes of a bucket for region and object access control and object lifecycle management. These attributes apply to all objects in the bucket. Therefore, you can create different buckets to perform different management functions.

- The storage space in a bucket is non-hierarchical without features of an object system, such as directory. Therefore, all objects are directly affiliated with their corresponding buckets.
- Each user can have multiple buckets.
- Bucket names must be globally unique within the OSS and cannot be changed once a bucket is created.
- An unlimited number of objects can be stored in a bucket.

Bucket naming rules

- The bucket name can only contain lower-case letters, digits, and hyphens (-).
- It must start with a lower-case letter or number.
- The length must be 3-63 bytes

Objects/Files

Objects are the basic elements used in OSS to store data. They are also called OSS files. An object is composed of metadata (Object Meta), user data (Data), and an object name (Key).Objects are labeled by a key that is unique within the bucket. An Object Meta is a key-value pair that expresses the object’ s attributes, such as its last modification time and size. Users can also store user-defined information in the Object Meta.

Object size may vary depending on the upload method. Multipart Upload can support objects of up to 48.8 TB. Other upload methods support a maximum size of 5 GB.

An object’ s lifecycle starts from when it has been successfully uploaded till it has been deleted. The object information cannot be changed during the lifecycle. Objects with the same name which are uploaded repeatedly will overwrite the previous object. Thus, unlike file systems, OSS does not allow users to modify only part of an object/file.

OSS provides the Append Upload function, which allows users to continually append data to the end of an object.

> Object naming rules
>
> - It uses UTF-8 encoding
> - The length must be 1-1023 bytes
> - It cannot start with “/ ” or “\”

Notes

Object names are case sensitive. Unless otherwise stated, files are equivalent to objects herein.

Region

A region is an area where OSS data centers are physically located.Users can select data storage regions based on their fees, request sources, and other factors. Generally, the closer the user is to a region, the faster the access speed. For details, see OSS Regions and Endpoints.

The region is specified when a bucket is created and cannot be subsequently changed. All the objects in this bucket will be stored in the corresponding data center. Currently, setting regions on an object basis is not supported.

Endpoint (Access Domain Name)

An endpoint is the domain name used to access the OSS.OSS provides services to the external through HTTP RESTful APIs. Different regions use different domain names. For access to the same region through the intranet or Internet, different endpoints are used. For example, the Internet endpoint for the Hangzhou region is oss-cn-hangzhou.aliyuncs.com, while the intranet endpoint is oss-cn-hangzhou-internal.aliyuncs.com. For more details, refer to OSS Regions and Endpoints.

AccessKey

An AccessKey (AK) indicates an AccessKeyId and AccessKeySecret pair used in access identity verification.The OSS verifies the identity of a request sender by using the AccessKeyId/AccessKeySecret symmetric encryption method. The AccessKeyId identifies a user. With the AccessKeySecret, a user can encrypt the signature string and the OSS can verify the AccessKey of the signature string. The AccessKeySecret must be kept confidential. In the OSS, AccessKeys come from the following sources:

- The AccessKey applied for by the bucket owner.
- The AccessKey granted by the bucket owner to an authorized third-party requestor through RAM.
- The AccessKey granted by the bucket owner to an authorized third-party requestor through STS.

For more information about AccessKeys, see RAM.

High Consistency

In the OSS, object operations are atomic. These operations must either succeed or fail without intermediate status. After a user uploads an object, OSS ensures it is complete. OSS will not return an upload success response for partial objects.

Object operations in OSS are likewise highly consistent. Once a user receives an upload (PUT) success response, this object can be read immediately and the data have already been written in triplicate. There are no intermediate upload statuses, i.e. read-after-write, but data is unreadable. The same goes for delete operations. After a users deletes an object, this object no longer exists.

This high-consistency feature facilitates user architectural design. The logic of OSS usage is the same as that of a traditional storage device: modifications are immediately visible and users do not have to consider final consistency issues.

Comparison of OSS and File Systems

OSS is a distributed object storage service that uses a Key-Value pair format. Users retrieve object content based on unique object names (Keys). Although users can use names like test1/test.jpg, this does not indicate the object is saved in a directory named test1. In the OSS, test1/test.jpg is simply a string, no difference in nature than a.jpg. Therefore, similar amounts of resources are consumed when accessing objects with different names.

File systems use a typical tree-type index structure. To access an object named test1/test.jpg, the client must first access the directory test1 and then find the file named test.jpg in this directory. This makes it easy for file systems to support folder operations, such as renaming directories, deleting directories, and moving directories, as these operations are only directory node operations. This structure means that more resources are consumed when more directory levels must be accessed and operations involving directories with many files are slow.

In OSS, there are several operations that can be used to simulate similar functions, but this is very expensive. For example, if a users wants to rename the test1 directory test2, the actual OSS operation would be to replace all objects starting with test1/ with copies starting with test2/. Such an operation would consume a great deal of resources. Thus, when using OSS, users should try to avoid such operations as far as possible.

OSS does not support the modification of saved objects (in the append object operation, users must call a specific API, and the generated object is of a different type than normally uploaded objects). To modify even just a single byte, the user must upload the entire object again. File systems allow users to modify files, e.g. modify the content at a specified offset location or truncate the end of an object. These features give file systems wide applicability. However, on the other hand, OSS supports massive concurrent access volumes, whereas file systems are limited by the performance of a single device.

Therefore, mapping the OSS onto an object system is very inefficient and not recommended. If mounting an object system is required, the user should try to confine the operations to writing new files, deleting files, and reading files. When using the OSS, users should make full use of its advantages, i.e. its massive data volume processing capabilities to store massive volumes of unstructured data, such as images, videos, and documents.

Comparison of OSS and File System Concepts:

OSS	File System
Object	File
Bucket	Main directory
Region	N/A
Endpoint	N/A
AccessKey	N/A
N/A	Multilevel directory
GetService	Retrieving the list of main directories
GetBucket	Retrieving the list of files
PutObject	Writing an object
AppendObject	Append writing an object
GetObject	Reading an object
DeleteObject	Deleting an object
N/A	Modifying file content
CopyObject (same target and source)	Modifying file attributes
CopyObject	Copying an object
N/A	Renaming an object

OSS Glossary

Term	Definition
Bucket	Storage space
Object	An object or file
Endpoint	The domain name for OSS access
Region	An area or data center
AccessKey	An alias for the AccessKeyId and AccessKeySecret pair
Put Object	Simple upload
Post Object	Form upload
Multipart Upload	An upload in multiple parts
Append Object	An upload that appends data
Get Object	Simple download
Callback	Upload callback
Object Meta	File Metadata;it describes the file, e.g. length and type
Data	File data
Key	A file name
ACL(Access Control List)	Permissions for buckets or files

Advantages

OSS outperforms user-created server storage in the following ways:

Item	OSS	User-created Server Storage
Reliability	- Service availability is no lower than 99.9% - Automatic scaling, without affecting external services - Data persistence no less than 99.99999999%, automatic redundant data backup	- Limited by hardware reliability, traditional storage is prone to faults. If a bad track is found on a disk, data may be irretrievably lost - Manual data recovery is difficult and time and energy-consuming
Security	- Provides enterprise-level, multi-layer security protection and anti-DDoS defenses, including	- Expensive cleaning and black hole equipment must be purchased separately - Security mechanisms must

	automatic black hole cleaning - Multi-user resource isolation mechanisms; supports remote disaster recovery - Provides various authentication and authorization mechanisms, as well as white list, anti-leeching, and primary/subaccount features	be implemented independently, resulting in high development and maintenance costs
Costs	- Very cost-effective, with a price starting from RMB0.14/GB/Month; a free quota each month - Multi-line BGP backbone network; no bandwidth restrictions; free upstream traffic - No maintenance staff or hosting costs; no O&M cost	- Large, one-time investment; low resource utilization - Storage space is limited by hardware capacity, must be resized manually - Single and double-line access methods provide slow speeds; limited bandwidth; must be manually resized during peak traffic periods - Requires professional O&M staff, resulting in high costs
Data Processing Capabilities	Provides image processing, audio/video transcoding, accelerated content delivery, porn identification service, archiving services, etc. The range of value-added data services continues to expand	Must be purchased and deployed separately

Application Scenarios

OSS is an object storage service applicable to a wide range of scenarios.

Massive Storage Space for Image and Audio/Video Applications

Suitable for the storage of massive volumes of images, audio/video, logs, and other files; supports various devices; websites and mobile apps directly write or read data to/from OSS; supports stream writing and file writing methods.

Static/Dynamic Resource Separation for Webpages and

Apps

Developers can directly use OSS and the BGP bandwidth for direct data download at an ultra-low latency. The OSS can also be used with Alibaba Cloud CDN acceleration service to deliver images, audio/video files, and app updates, enhancing the user experience.

Cloud Data Processing

After uploading an object to OSS, the user can use the integrated Media Transcoding Service (MTS) and Image Processing Service (IMG) to process data on the cloud.

OSS Change History

Date	Event Description
2016/3/10	Support for automatic fragment cleaning function: Fragment lifecycles can be set through the object lifecycle management function.
2016/3/6	Official release of the OSS Media SDK. Camera equipment encapsulates data for direct upload to OSS
2016/1/14	OSS supports back-to-source settings and can send back-to-source requests to the origin site through the mirror or redirection method
2016/1/7	Release of the cross-region replication function, providing a live management mechanism for different places
2015/12/7	Release of OSS FUSE
2015/11/26	Release of the OSS Ruby SDK
2015/11/10	Image processing function enabled by default for OSS buckets
2015/10/23	Size limit for file uploads to the console raised to 500 MB
2015/8/20	The migration tool oss-import becomes available
2015/7/29	Unit price for CDN back-to-source OSS traffic reduced by 40% to 0.15 RMB/GB, OSS downstream traffic price reduced by 15% in the South China 1 region (originally Shenzhen)
2015/7/18	Append and write function available

2015/7/8	Support for Keep Alive connections, optimized connection quality
2015/7/8	Support for callback of uploaded files to the application server
2015/6/15	OSS supports VPC network connection configuration; supports object-level ACL settings
2015/4/26	Support for URL redirects; automatic refresh of CDN cache when content is overwritten
2015/4/26	OSS connected to RAM; support for primary/subaccount authorization and temporary authorization
2015/2/10	Free 5 GB provided in the North China 2 (originally Beijing) and South China 1 (originally Shenzhen) regions, unit price for CDN back-to-source OSS traffic reduced by 66% to 0.25 RMB/GB
2015/1/27	Reduction of unit prices for storage: 15% reduction in the North China 1 region (originally Qingdao), 10% reduction in the North China 2 region (originally Beijing), and 5% reduction in the East China 1 region (originally Hangzhou)
2014/10/20	File lifecycle management available, support for bulk file management
2014/10/14	Internet traffic priced according to time, with early morning traffic charged at half price
2014/5/28	Unit price for Internet traffic reduced by 10% in the North China 1 region (originally Qingdao)
2014/3/28	Unit price per GB of storage reduced by 40%
2014/3/15	OSS supports the CORS function (supports user webpages' browser links)
2014/2/12	OSS supports chunked encoded uploads and Post form uploads
2012/11/4	Server-side encryption function made available
2012/9/4	CNAME function made available
2012/8/9	Support for log functions, records of all user access requests are logged in the specified bucket
2012/6/20	Support for the static website hosting function
2012/3/30	Support for separate billing for Internet and intranet traffic, reduces users' ECS costs

2012/3/29	Support for https
2012/2/29	Support for multipart upload
2011/12/16	Support for Copy Object, anti-leeching function, and HTTP Headers function
2011/10/22	Official launch of the OSS paid service

Access and Control

OSS Access Domain Names

Composition Rules for OSS Domain Names

For all network requests for the OSS except those for the GetService API, the domain names are third-level domain names for specific buckets. The domain name is composed by the bucket name and **endpoint**:bucketname.endpoint. Here, **endpoint** varies with the region (data center) of the bucket and the intranet/Internet access method.

Endpoint Naming Rules for External Network

Here, external network refers to the Internet.

```
Region + .aliyuncs.com
```

Endpoint Naming Rules for Internal Network

Here, internal network refers to AliCloud's intranet.

```
Region + -internal + .aliyuncs.com
```

OSS Regions and Endpoints

Region Name	Region Expression	Internet Endpoint	Intranet Endpoint for ECS Access
East China 1	oss-cn-hangzhou	oss-cn-	oss-cn-hangzhou-

(Hangzhou)		hangzhou.aliyuncs.com	internal.aliyuncs.com
East China 2 (Shanghai)	oss-cn-shanghai	oss-cn-shanghai.aliyuncs.com	oss-cn-shanghai-internal.aliyuncs.com
North China 1 (Qingdao)	oss-cn-qingdao	oss-cn-qingdao.aliyuncs.com	oss-cn-qingdao-internal.aliyuncs.com
North China 2 (Beijing)	oss-cn-beijing	oss-cn-beijing.aliyuncs.com	oss-cn-beijing-internal.aliyuncs.com
South China 1 (Shenzhen)	oss-cn-shenzhen	oss-cn-shenzhen.aliyuncs.com	oss-cn-shenzhen-internal.aliyuncs.com
Hong Kong data center	oss-cn-hongkong	oss-cn-hongkong.aliyuncs.com	oss-cn-hongkong-internal.aliyuncs.com
Silicon Valley data center	oss-us-west-1	oss-us-west-1.aliyuncs.com	oss-us-west-1-internal.aliyuncs.com
Virginia data center	oss-us-east-1	oss-us-east-1.aliyuncs.com	oss-us-east-1-internal.aliyuncs.com
Asia Pacific (Singapore) data center	oss-ap-southeast-1	oss-ap-southeast-1.aliyuncs.com	oss-ap-southeast-1-internal.aliyuncs.com

For more information, refer to Access Domain Names and Data Centers

Domain Name Settings in OSS SDK

The OSS SDK has spliced an access domain name for each action of a user. However, users need to set different **endpoints** when operating on buckets of different regions.

Taking the Java SDK as an example, users need to set the **endpoint** during class instantiation before operating on buckets on the Hangzhou node:

```
String accessKeyId = "<key>";
String accessKeySecret = "<secret>";
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
OSSClient client = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

Using Intranet Endpoints to Access OSS in ECS

Access is allowed between an ECS instance and an OSS instance in the same region through intranet addresses.

For example, a user has purchased an ECS instance located in Beijing. Assuming that this user has an OSS bucket which is named 'beijingres' and located in Beijing, the user can access resources in 'beijingres' through 'beijingres.oss-cn-beijing-internal.aliyuncs.com'. Assuming that this user has another bucket which is named 'qingdaores' and located in Qingdao, the user is unable to access it through the intranet address 'qingdaores.oss-cn-qingdao-internal.aliyuncs.com ' from the ECS instance located in Beijing. In this case, the user must access the OSS in Qingdao through the Internet address 'qingdaores.oss-cn-qingdao.aliyuncs.com'.

In the above sample Java SDK, the Internet address of the bucket is used for OSS access. To access the OSS through the intranet, just modify **endpoint**:

```
String accessKeyId = "<key>";
String accessKeySecret = "<secret>";
String endpoint = "http://oss-cn-hangzhou-internal.aliyuncs.com";
OSSClient client = new OSSClient(endpoint, accessKeyId, accessKeySecret);
```

Accessing OSS

OSS Access URLs

OSS is an object storage service based on HTTP APIs. In all operations, users need to specify the OSS resource to access.This resource may be a bucket or an object. During the access, the OSS resource is expressed in URL format.Construction of OSS URLs

```
<Schema>://<Bucket>.<Endpoint>/<Object> Third-level domain name access method

Schema: value of HTTP or HTTPS
Bucket: the user's OSS storage space
Endpoint: the access domain name for a bucket's data center
Object: an object uploaded by a user to the OSS
```

Notes

1. Here, the endpoint must be consistent with the bucket’ s data center (region). For example: If a bucket was created in Hangzhou, the Hangzhou endpoint must be used. Endpoints for other regions cannot be used.
2. For access to the OSS, ECS instances can use the intranet endpoint for OSS resources in the same region.For a list of regions and their endpoints, refer to Access Domain Names.

If a user uses HTTPS to send a request to the Hangzhou OSS for an object named mytest/oss-test-object in a bucket named oss-sample,the accessed third-level domain name will be as follows:

```
https://oss-sample.oss-cn-hangzhou.aliyuncs.com/mytest/oss-test-object
```

Users can directly use object URLs in HTML, as shown below:

```
<img src="https://oss-example.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png" />
```

OSS Access Security

HTTP requests sent to OSS are divided into two types depending on whether they include identity authentication information: Requests with identity verification information and anonymous requests without identity verification information.The identity verification information in requests can be structured in two ways:

- Authorization is contained in the request header, in the format: OSS + AccessKeyId + signature string.
- OSS AccessKeyId and Signature fields are contained in the request URL.

OSS Access Verification Process

Anonymous Request Access Process

1. The user’ s request is sent to the OSS’ s HTTP server.
2. The OSS resolves the URL to get the bucket and object.
3. If no ACL is set for the object, the process goes directly to Step 4. Otherwise, if the object’ s ACL permits anonymous access, the process skips to step 5. If it does not, the request is rejected and the process ends.
4. If the bucket’ s ACL does not permit anonymous access, an error message is returned and the process ends.
5. The request passes permission verification and the object content is returned to the user.

Access Process for Requests with ID Verification Information

1. The user’ s request is sent to the OSS’ s HTTP server.
2. The OSS resolves the URL to get the bucket and object.
3. Based on the request’ s OSS AccessKeyId, the OSS retrieves the ID information of the requestor. If this cannot be obtained, an error message is returned and the process ends. If the information is obtained, but the requestor is not permitted to access this resource, an error message is returned and the process ends. If the information is obtained, but the signature calculated based on the request’ s HTTP parameters does not match the sent signature, an error message is returned and the process ends.
4. If no ACL is set for the object, the process goes directly to Step 5. Otherwise, if the object’ s ACL permits anonymous access, the process skips to step 6. If it does not, the request is rejected and the process ends.
5. If the bucket’ s ACL does not permit anonymous access, an error message is returned and

the process ends.
6. The request passes permission verification and the object content is returned to the user.

Three Methods for OSS Access with ID Verification Information

- Using the console to access the OSS:In the console, the identity verification process is concealed from users. When users access the OSS through the console, they do not have to concern themselves with the details of this process.
- Using SDKs to access the OSS:The OSS provides SDKs for multiple development languages. A signature algorithm is implemented in an SDK, where users only need to input the AK information as a parameter.
- Using APIs to access the OSS:If you want to write your own code to package a call to the RESTful API, you need to implement a signature algorithm to calculate the signature. For details about the signature algorithm, refer to Adding a Signature to the Header and Adding a Signature to the URL in the API Reference.

For an explanation of AccessKeys and more information on identity authentication operations, refer to RAM

User-defined OSS Domain Names

The OSS allows users to bind custom domain names to their buckets. This operation must be performed through the OSS Console. According to the requirements of Internet Management Regulations, all users activating this function shall provide valid materials including the MIIT filing number and the ID of the domain name holder. This function can be used only after these materials are approved by Alibaba Cloud. After the CNAME function is activated, the OSS will automatically process access requests on that domain name.

Example of the CNAME application scenario:

- User A possesses a website with the domain name abc.com and which contains a page with the link http://img.abc.com/logo.png.
- At this time, user A must send a request to migrate the image img.abc.com to the OSS. However, he does not want to modify any website code or change the external link. The CNAME function is perfectly suited to this scenario.
- Through the OSS Console, user A submits an application to bind the user-defined domain name img.abc.com to abc-img and provides the associated materials.
- After AliCloud approves the application, the OSS background will map img.abc.com onto abc-img (permission verification will be performed at this time).
- On his own domain name server, user A adds a CNAME rule, mapping img.abc.com onto abc-img.oss-cn-hangzhou.aliyuncs.com. Thus, all access traffic to the user’ s img.abc.com domain name will be forwarded to abc-img.oss-cn-hangzhou.aliyuncs.com on the OSS.

- After a request for http://img.abc.com/logo.png reaches the OSS, the OSS will find the img.abc.com and abc-img mapping and actually convert the request to an access request for the abc-img bucket. When a user attempts to access http://img.abc.com/logo.png, after passing through the OSS, the website actually accessed is http://abc-img.oss-cn-hangzhou.aliyuncs.com/logo.png.

	Without Bound CNAME	With Bound CNAME
Process Comparison	Access to http://img.abc.com/logo.png -> DNS resolution to the user’s server IP -> Accesses logo.png on the user’s server	Access to http://img.abc.com/logo.png -> DNS resolution to abc-img.oss-cn-hangzhou.aliyuncs.com -> Access to logo.png in the OSS bucket abc-img

CNAME Usage Notes

```
1. First, approval must be obtained from Alibaba Cloud.
2. The user creates a bucket on OSS and uploads the relevant files.
3. The user activates CNAME on the console and maps the user-defined domain name on the bucket.
4. The user resolves the user-defined domain name to Bucket.Endpoint on the DNS provider's console.

Perform these operations in the above order.
```

Operation process:

- Console: Domain Name Management

OSS Access Domain Names

OSS Regions and Endpoints

The intranet/internet **endpoints** in each region for a typical network are as follows:

Region Name	Region Expression	Internet Endpoint	Intranet Endpoint for ECS Access
East China 1 (Hangzhou)	oss-cn-hangzhou	oss-cn-hangzhou.aliyuncs.com	oss-cn-hangzhou-internal.aliyuncs.com
East China 2 (Shanghai)	oss-cn-shanghai	oss-cn-shanghai.aliyuncs.com	oss-cn-shanghai-internal.aliyuncs.com
North China 1 (Qingdao)	oss-cn-qingdao	oss-cn-qingdao.aliyuncs.com	oss-cn-qingdao-internal.aliyuncs.com

North China 2 (Beijing)	oss-cn-beijing	oss-cn-beijing.aliyuncs.com	oss-cn-beijing-internal.aliyuncs.com
South China 1 (Shenzhen)	oss-cn-shenzhen	oss-cn-shenzhen.aliyuncs.com	oss-cn-shenzhen-internal.aliyuncs.com
Hong Kong data center	oss-cn-hongkong	oss-cn-hongkong.aliyuncs.com	oss-cn-hongkong-internal.aliyuncs.com
Silicon Valley data center	oss-us-west-1	oss-us-west-1.aliyuncs.com	oss-us-west-1-internal.aliyuncs.com
Virginia data center	oss-us-east-1	oss-us-east-1.aliyuncs.com	oss-us-east-1-internal.aliyuncs.com
Asia Pacific (Singapore) data center	oss-ap-southeast-1	oss-ap-southeast-1.aliyuncs.com	oss-ap-southeast-1-internal.aliyuncs.com

NOTE

-It is recommended you use third-level domain names, i.e., 'Bucket' + 'Endpoint' format when sharing links or binding CNAME domain names. For example, the third-level domain name for the Shanghai bucket oss-sample would be oss-sample.oss-cn-shanghai.aliyuncs.com.

When using new SDK versions (except 'C SDK'), please use 'http://andhttps://+Endpoint' as the initialization parameter. Take the Shanghai endpoint as an example. This would be http://oss-cn-shanghai.aliyuncs.com or https://oss-cn-shanghai.aliyuncs.com。**. Do not use a third-level domain name as the initialization parameter** (i.e., do not use http://bucket.oss-cn-shanghai.aliyuncs.com).

However, because earlier SDK versions (for example, the 'C, PHP, and Python SDKs') directly use endpoints (for example, 'oss-cn-shanghai.aliyuncs.com'). please refer to the documentation or code instructions for the SDK version you are using.

The original address oss.aliyuncs.com is directed to the Internet address of the Hangzhou node by default.

The original intranet address oss-internal.aliyuncs.com is directed to the intranet address of the Hangzhou node by default.

VPC Network Regions and Endpoints

To access OSS, ECS of a VPC network can **only** use the following **endpoints**:

Region Name	Region Expression	Endpoint of the VPC

		Network
East China 1 (Hangzhou)	oss-cn-hangzhou	vpc100-oss-cn-hangzhou.aliyuncs.com
East China 2 (Shanghai)	oss-cn-shanghai	vpc100-oss-cn-shanghai.aliyuncs.com
North China 1 (Qingdao)	oss-cn-qingdao	vpc100-oss-cn-qingdao.aliyuncs.com
North China 2 (Beijing)	oss-cn-beijing	vpc100-oss-cn-beijing.aliyuncs.com
South China 1 (Shenzhen)	oss-cn-shenzhen	vpc100-oss-cn-shenzhen.aliyuncs.com
Silicon Valley data center	oss-us-west-1	vpc100-oss-us-west-1.aliyuncs.com
Virginia data center	oss-us-east-1	oss-us-east-1-internal.aliyuncs.com
Asia-Pacific (Singapore) data center	oss-ap-southeast-1	vpc100-oss-ap-southeast-1.aliyuncs.com

Accessing OSS

OSS-based App Development

Development Architecture

There are four components in typical OSS-based app development

- OSS: Provides functions such as upload, download, and upload callback.
- Developer’ s mobile client (app or webpage application), called the client for short: Indirectly accesses the OSS though the service provided by the developer.
- Application server: The server that interacts with the client. This is also the server for the developer’ s service.
- AliCloud STS: Issues temporary credentials.

Service Development Process

Temporary Credential Upload Authorization

1. The client sends a request to the application server asking to upload an object to OSS.
2. The application server must send a request to the STS server to obtain temporary credentials.
3. The application server replies to the client, returning the temporary credentials.
4. The client obtains authorization to upload to OSS (the STS AccessKey and token) and calls the mobile client SDK provided by OSS to upload data.
5. The client successfully uploads data to the OSS. If callback is not set, the process is complete. If the callback function is set, the OSS will call the relevant interface.

Here are several key points:

- The client does not have to request authorization from the application server for each upload. After the first authorization, the client will cache the temporary credentials returned by the STS until they expire.
- STS provides powerful access control functions that can restrict client access permission at the object level. This completely isolates the objects uploaded to the OSS by different clients, greatly enhancing the security of applications.

For more information, refer to Authorized Third-Party Uploads

Signed URL Authorization for Uploads and Form Uploads

1. The client sends a request to the application server asking to upload an object to OSS.
2. The application server replies to the client, returning credentials (signed URL or form).
3. The client obtains authorization to upload to OSS (the signed URL or form) and calls the mobile client SDK provided by OSS to upload data or directly uploads a form.
4. The client successfully uploads data to the OSS. If callback is not set, the process is complete. If the callback function is set, the OSS will call the relevant interface.

For more information, refer to Authorized Third-Party Uploads

Temporary Credential Download Authorization

The process is similar to temporary credential upload authorization:

1. The client sends a request to the application server for downloading an object from OSS.
2. The application server must send a request to the STS server to obtain temporary credentials.
3. The application server replies to the client, returning the temporary credentials.
4. The client obtains authorization to download from OSS (the STS AccessKey and token) and calls the mobile client SDK provided by OSS to download data.
5. The client successfully downloads an object from OSS.

Here are several key points:

- Just as for uploads, the client will cache the temporary credentials to increase access speed.
- The STS likewise provides precise object download permission control, which, together with upload permission control, serves to completely isolate the OSS storage space of each mobile client.

Signed URL Authorization for Downloads

This is similar to signed URL authorization for uploads:

1. The client sends a request to the application server for downloading an object from OSS.
2. The application server replies to the client, returning the signed URL.
3. The client obtains authorization to download from OSS (the signed URL) and calls the mobile client SDK provided by OSS to download data.
4. The client successfully downloads an object from OSS.

Special Note

The client cannot store the developer’ s AccessKey, but may only obtain a signed URL or the temporary credentials issued by the STS (the STS AccessKey and token) from the application server.

Reference for Using the Function:

- SDK: Android SDK File Operations
- SDK: iOS SDK File Operations

OSS Quick Start

Quick Start with the Console:

1. Log onto the OSS Console and activate OSS.
2. Create a bucket.
3. Upload and download files.

For details, see the Console Quick Start documentation.

Quick Introduction to OSS Uploads and Downloads

Before getting started with SDKs, please refer to the sections about the upload and download functions in the Developer Guide.

OSS uses RESTful APIs to perform operations and all requests are standard HTTP requests.

- OSS provides different file upload methods, such as using a single PUT request to complete a Simple Upload, using webpage forms for direct uploads, called Form Uploads, and uploading large files with Multipart Uploads. For video monitoring and other applications, OSS also provides Append Object.
- Likewise, OSS provides multiple download methods: Simple Downloads and, for larger files, Resumable Data Transfer.

Quick Start with SDKs:

1. After activating OSS, retrieve the AccessKeyId and AccessKeySecret from the console.
2. Download the SDKs for various programming languages.
3. Based on the descriptions in the SDK documentation, perform uploads, downloads, and other operations.

For details, see the SDK Development Documentation.

Bucket Management

Create a Bucket

Users can create a bucket in an existing region. Note that the following conditions must be met:

- Each user can create up to 10 buckets.
- The name of each bucket must be globally unique; otherwise, the bucket cannot be created.
- The bucket name must comply with the naming rules.
- Once the bucket is created, its name and region cannot be modified.

The OSS provides an Access Control List (ACL) for permission control. You can configure an ACL when creating a bucket and modify the ACL after creating the bucket. If no ACL is configured, the default value is Private. For more details, refer to Set Bucket ACLs.

Function Usage Reference

Creating a Bucket

- Console: Create a Bucket
- SDK: Java SDK-Create a bucket in Bucket
- API: Put Bucket

Setting Bucket Read and Write Permissions (ACL)

You can not only set the bucket ACL when creating a bucket, but also modify the bucket ACL according to your service requirements. Only the creator of the bucket has permission to perform this operation. Currently, three access permissions are available for a bucket:

Permission	Access Restriction
public-read-write	Anyone (including anonymous users) can perform read and write operations on the files in the bucket. The fees incurred by these operations will be borne by the creator of the bucket. Please use this permission with caution.
public-read	Only the creator of the bucket can perform write operations on the files in the bucket, while others (including anonymous users) can perform read operations on the files.
private	Only authorized users are allowed to read, write, and delete files in the bucket. Others cannot access the files in the bucket without authorization.

For details, refer to Access Permissions

Function Usage Reference

Setting ACLs for a bucket

- Console: Access Permissions Configuration
- SDK：Java SDK-**Set Bucket ACL** in Bucket
- API: Put BucketACL

Obtaining ACLs for a bucket

- Console: After logging in to the console, users can view the ACL in the bucket attributes.
- SDK：Java SDK-**Obtain Bucket ACL** in Bucket
- API: Get BucketACL

Viewing the List of Buckets

The following section describes how to view the list of all buckets you have created.

Function usage reference:

- Console: After you log in to the console, the list of all buckets you have created is displayed by default.
- API: GetService
- SDK：Java SDK-List buckets in Bucket

Reference Links:

- Create aBucket

Obtaining Bucket Information

Users can obtain the region of a bucket. The region means the physical location of the data center. The returned Location field indicates the region where the bucket is located. For example, if the location is East China 1 (Hangzhou in former use), the returned Location field is oss-cn-hangzhou.OSS’s Access Domain Names

Function Usage Reference

- Console: The region information is directly displayed in the bucket attributes on the console.
- API: Get Bucket Location
- SDK: Java SDK-**Obtain the Bucket Address** in Bucket

Delete a Bucket

Users can delete the buckets they have created.

> NOTEA non-empty bucket (which contains files or file fragments) cannot be deleted. The bucket can be deleted only after the files or file fragments are deleted. If you want to delete all files in a bucket, it is recommended to use Lifecycle Management.

Function usage reference:

- API: Delete Bucket
- SDK: **Delete Bucket** in Java SDK-Bucket
- Console: Delete Bucket

Uploading Files

Simple Upload

Applicable Scenarios

Simple upload refers to the situation where a user uploads a single object by using the Put Object method in the OSS API. This is applicable to any scenario where a single HTTP request interaction completes an upload, e.g. upload of a small file upload.

Setting Object Meta When Uploading Files

A simple upload can carry an Object Meta that describes the object, for example, Content-Type and other standard HTTP headers, or user-defined information. For details, refer to Object Meta.

Upload Restrictions

- Size limit: The maximum object size is 5 GB in this mode.
- Naming restrictions
  - It uses UTF-8 encoding.
  - The length must be 1-1,023 bytes
  - It cannot start with “/ “ or “\”.

Upload of Large Files

Using a single HTTP request to upload a large object may lead to an excessive upload time. If a network error occurs during this time (for example, timeout or disconnection), the upload will probably fail. In this case, users may consider resumable upload (multipart upload). For objects larger than 5 GB, only resumable upload (multipart upload) can be used. For details, refer to Resumable Upload.

Upload Security and Authorization

To prevent unauthorized third parties from uploading objects to the developer’ s bucket, OSS provides bucket- and object-level access permission control. For details, refer to Access Control.In addition to bucket and object-level access permissions, OSS also provides account-level authorization to authorize third-party uploads. For details, see Authorized Third-party Upload for Upload Security.

Post-upload Operations

After an object has been uploaded to OSS, the developer can use Upload Callback to initiate a callback request to the specified application server in order to perform subsequent operations.To process uploaded images, users can use Cloud Processing for Uploaded Images.For audio/video file format conversion, users can use Media Transcoding.

Reference for Using the Function:

- API: PutObject
- SDK: Java SDK-**PutObject** in Object
- Console: Uploading Files

Best Practices

- RAM and STS User Guide
- Web Client Direct Data Transfer and Upload Callback

Reference Links:

- Upload Callback
- Introduction to Mobile Development Upload Scenarios
- Downloading Uploaded Files
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Access Control for Upload Security
- Authorized Third-party Upload for Upload Security
- Copying, Deleting, and Managing Uploaded Files

Form Upload

Applicable Scenarios

Form upload, suitable for uploading small files, refers to the situation where a user uploads an object by using the Post Object request in the OSS API. This method is quite helpful in uploading objects embedded in HTML webpages. A typically scenario is websites. Here, we use a job search website as an example:

	Without Using Form Upload	Using Form Upload
Procedures	A website user uploads a resume -> The website server responds to the upload page -> The resume is uploaded to the server -> The server uploads the resume to OSS	A website user uploads a resume -> The website server responds to the upload page -> The resume is uploaded to OSS

Upload Restrictions

- Size limit: The maximum object size is 5 GB in this mode.
- Naming restrictions:
    - It uses UTF-8 encoding.
    - The length must be 1-1,023 bytes.
    - It cannot start with “/ ” or “\”.

Advantages of Form Upload

- In procedure, the step of file forwarding is bypassed.
- In architecture, the way of uploading files to the website server may require resizing the website server, which becomes the bottleneck. With form upload, files are uploaded directly from the client to the OSS. The OSS will guarantee the quality of service, especially in the case of large upload volumes.

Upload Security and Authorization

To prevent unauthorized third parties from uploading objects to the developer’s bucket, OSS provides bucket- and object-level access permission control. For details, refer to Access Control.To prevent unauthorized third parties from uploading objects to the developer’s bucket, OSS provides bucket- and object-level access permission control. For details, refer to Access Control.

Basic Process

1. Construct a Post Policy. This policy allows the Website developer to restrict the uploads of Website users. For example, it can specify upload size restrictions, object name restrictions, and the URL the client jumps to and the status code received after a successful upload. For

details, refer to Post Policy.
In this simple policy example, the expiration time for site user uploads is 2115-01-27T10:56:19Z (here we have set a long expiration time so that the test is successful; in actual use we do not recommend this setting) and the maximum upload file size is 104,857,600 bytes.

```
This example uses Python code and the policy is a string in JSON format.

policy="{\"expiration\":\"2115-01-27T10:56:19Z\",\"conditions\":[[\"content-length-range\", 0, 104857600]]}"
```

1. Encode the policy string using base64 encoding.
2. Use the OSS AccessKeySecret to sign the base64 encoded policy.
3. Construct an HTML page for uploads.
4. Open the HTML page and select the file to upload.

Complete Python code example

```
#coding=utf8
import md5
import hashlib
import base64
import hmac
from optparse import OptionParser

def convert_base64(input):
return base64.b64encode(input)

def get_sign_policy(key, policy):
return base64.b64encode(hmac.new(key, policy, hashlib.sha1).digest())

def get_form(bucket, endpoint, access_key_id, access_key_secret, out):
#1 Construct a Post Policy
policy="{\"expiration\":\"2115-01-27T10:56:19Z\",\"conditions\":[[\"content-length-range\", 0, 1048576]]}"
print("policy: %s" % policy)

#2 Encode the policy string using base64 encoding
base64policy = convert_base64(policy)
print("base64_encode_policy: %s" % base64policy)

#3 Use the OSS AccessKeySecret to sign the base64 encoded policy
signature = get_sign_policy(access_key_secret, base64policy)

#4 Construct an HTML page for uploads
form = '''
<html>
<meta http-equiv=content-type content="text/html; charset=UTF-8">
<head><title>OSS form upload (PostObject)</title></head>
<body>
<form action="http://%s.%s" method="post" enctype="multipart/form-data">
<input type="text" name="OSSAccessKeyId" value="%s">
<input type="text" name="policy" value="%s">
```

```
<input type="text" name="Signature" value="%s">
<input type="text" name="key" value="upload/${filename}">
<input type="text" name="success_action_redirect" value="http://oss.aliyun.com">
<input type="text" name="success_action_status" value="201">
<input name="file" type="file" id="file">
<input name="submit" value="Upload" type="submit">
</form>
</body>
</html>
''' % (bucket, endpoint, access_key_id, base64policy, signature)
f = open(out, "wb")
f.write(form)
f.close()
print("form is saved into %s" % out)

if __name__ == '__main__':
parser = OptionParser()
parser.add_option("", "--bucket", dest="bucket", help="specify ")
parser.add_option("", "--endpoint", dest="endpoint", help="specify")
parser.add_option("", "--id", dest="id", help="access_key_id")
parser.add_option("", "--key", dest="key", help="access_key_secret")
parser.add_option("", "--out", dest="out", help="out put form")
(opts, args) = parser.parse_args()
if opts.bucket and opts.endpoint and opts.id and opts.key and opts.out:
get_form(opts.bucket, opts.endpoint, opts.id, opts.key, opts.out)
else:
print "python %s --bucket=your-bucket --endpoint=oss-cn-hangzhou.aliyuncs.com --id=your-access-key-id --key=your-access-key-secret --out=out-put-form-name" % __file__
```

Save this code segment as post_object.py and use Python to run it.

```
Usage:
python post_object.py --bucket=Your bucket --endpoint=The bucket's OSS domain name --id=Your AccessKeyId --key=Your AccessKeySecret --out=Output file name

Example:
python post_object.py --bucket=oss-sample --endpoint=oss-cn-hangzhou.aliyuncs.com --id=tphpxp --key=ZQNJzf4QJRkrH4 --out=post.html
```

NOTE

In the constructed form, "success_action_redirect" value="http://oss.aliyun.com"indicates the page to jump to after a successful upload. This can be replaced as needed.
In the constructed form, "success_action_status" value="201" indicates that Status Code 201 is returned after a successful upload. This can be replaced as needed.
If the generated HTML file is post.html, open post.html and select the file to upload. In this example, the client jumps to the OSS homepage after a successful upload.

Reference for Using the Function:

- API: PostObject

Best Practices

- Web Client Direct Data Transfer
- OSS Cross-origin Resource Sharing (CORS) User Guide

Reference Links:

- Upload Callback
- Introduction to Mobile Development Upload Scenarios
- Downloading Uploaded Files
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Access Control for Upload Security
- Authorized Third-party Upload for Upload Security
- Copying, Deleting, and Managing Uploaded Files

Resumable Upload

Applicable Scenarios

When using simple upload (PutObject) to upload large files to OSS, the upload may fail by a network error. A re-attempt uploads the file from the very beginning. To address this problem, the OSS provides the multipart upload so that users can resume interrupted uploads.As its name suggests, multipart upload splits the uploaded file into multiple data blocks (or parts in OSS) and then uploads each part separately. When the upload is complete, an OSS API is called to combine the parts into an object.

Compared to other upload methods, multipart upload is useful in the following scenarios:

- **Poor network environments**. If an upload fails when using a cell phone, the user can re-upload only the failed part, and does not have to re-upload the other parts.
- **Resumable upload required**. After pausing an upload in progress, it can be restarted from the paused part.
- **Accelerating upload**. When the file to be uploaded to OSS is very large, multiple parts can be uploaded in parallel to accelerate the upload.
- **Stream upload**. Users can start to upload an object of an unknown size. A typical application is the video monitoring industry.

Basic Process

The process is as follows:

- Split the file to be uploaded according to a specified part size.
- Initialize a multipart upload task (InitiateMultipartUpload).
- Upload the parts in sequence or in parallel (UploadPart).

  Complete the upload (CompleteMultipartUpload).

  NOTE:

  Each part except the last cannot be smaller than 100 KB; otherwise, the call to the CompleteMultipartUpload will fail.

  After splitting the file into parts, the parts are ordered by the partNumbers specified during the upload. In actual execution, the parts can be uploaded in parallel. Upload in sequential is not required. The upload speed does not necessarily increase with the number of parts uploaded in parallel, as both the user’ s network conditions and the device load must be considered.

  By default, when the upload is complete, but CompleteMultipartUpload has not been called, the parts will not be automatically recycled. Therefore, to terminate the upload and delete the data-occupied storage space, call AbortMultipartUpload. To automatically recycle uploaded parts, refer to Lifecycle Management.

Resumable Upload

Because the lifecycle of uploaded parts is permanent, it is easy to implement the resumable upload function.

If the system crashes during a multipart upload, the user can resume the upload by using the ListMultipartUploads and ListParts APIs to retrieve all multipart upload tasks for an object and list the completed uploads in each task. This allows uploads to be resumed from the last uploaded part. The same principles apply to pausing and resuming uploads.

This function is especially useful for mobile device and large file uploads.

Upload Restrictions

- Size limit: the object size is determined by part size. The function supports a maximum of 10,000 parts, with a minimum part size of 100 KB (the last part may be smaller) and a

maximum part size of 5 GB.
- Naming restrictions
    - It uses UTF-8 encoding
    - The length must be 1-1,023 bytes.
    - It cannot start with “/ “ or “\”.

Upload Security and Authorization

To prevent unauthorized third parties from uploading objects to the developer’ s bucket, OSS provides bucket- and object-level access permission control. For details, refer to Access Control.In addition to bucket and object-level access permissions, OSS also provides account-level authorization to authorize third-party uploads. For details, refer to Authorized Third-party Upload for Upload Security.

Post-upload Operations

After an object has been uploaded to OSS, the developer can use Upload Callback to initiate a callback request to the specified application server in order to perform subsequent operations.To process uploaded images, users can use Cloud Processing for Uploaded Images.For audio/video file format conversion, users can use Media Transcoding.

Reference for Using the Function:

- APIs: MultipartUpload, InitiateMultipartUpload, UploadPart, UploadPartCopy, CompleteMultipartUpload, AbortMultipartUpload, ListMultipartUploads, ListParts
- SDK: Java SDK-Multipart upload in MultipartUpload

Best Practices

- RAM and STS User Guide
- Web Client Direct Data Transfer

Reference Links:

- Upload Callback
- Introduction to Mobile Development Upload Scenarios
- Downloading Uploaded Files
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Access Control for Upload Security
- Authorized Third-party Upload for Upload Security

- Copying, Deleting, and Managing Uploaded Files

Append Upload

Applicable Scenarios

The Simple Upload, Form Upload, and Resumable Upload methods create normal-type objects which have fixed content after the upload is finished. They can only be read, but cannot be modified. If the object content changes, the user must upload an object of the same name to overwrite the content. This is a major difference between OSS and file systems.

This feature makes many application scenarios inconvenient, such as video monitoring and live video broadcast, since video data is constantly produced in real time. Using other upload methods, users must slice the video stream into small pieces and then upload them as new objects. In actual use, these methods have obvious defects:

- The software architecture is quite complex and users must consider intricate issues such as file fragments.
- Storage space is required for metadata, e.g. the list of generated objects. Thus, each request must read the metadata to judge if any new object has been generated. This puts a high level of access pressure on the server. In addition, each client request must be transmitted twice, causing a certain amount of delay.
- If the object parts are small, the delay is quite short. However this will complicate the management of most objects. If the object parts are large, the data will suffer a substantial delay.

To simply development and reduce costs in such a scenario, OSS provides the append object method, which allows users to directly append content to the end of an object. This method is used to operate on Appendable objects. The objects uploaded by other methods are Normal objects. The data appended is instantly readable.

With append object, the previous scenario becomes very simple. When video data are produced, they can be immediately added to the same object through the append object method. The client simply needs to regularly retrieve the object length and compare it with the previous value. If new readable data are found, this triggers a read operation to retrieve the newly uploaded data segments. This method greatly simplifies the architecture and enhances the scalability of applications.

In addition to video scenarios, the append object method can also be used to append log data.

Upload Restrictions

- Size limit: The maximum object size is 5 GB in this mode.
- Naming restrictions

  - It uses UTF-8 encoding.
  - The length must be 1-1,023 bytes.
  - It cannot start with “/ “ or “\”.

- File type: Only files created through append object can be appended with new data. Therefore, new data cannot be appended to files created through simple upload, form upload, or multipart upload.

Upload Security and Authorization

To prevent unauthorized third parties from uploading objects to the developer’ s bucket, OSS provides bucket- and object-level access permission control. For details, refer to Access Control.In addition to bucket- and object-level access permissions, OSS also provides account-level authorization to authorize third-party uploads. For details, refer to Authorized Third-party Upload for Upload Security.

Post-upload Operations

To process uploaded images, users can use Cloud Processing for Uploaded Images.For audio/video file format conversion, users can use Media Transcoding.

Reference for Using the Function:

- API: Append Object
- SDK: Java SDK-Append Object Example

NOTE

Append object does not support upload callback.

Best Practices

- RAM and STS User Guide

Reference Links:

- Downloading Uploaded Files
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Access Control for Upload Security
- Authorized Third-party Upload for Upload Security

Authorized Third-party Upload

Applicable Scenarios

In a typical client/server system architecture, the server is responsible for receiving and processing requests from the client. If we consider a scenario where OSS is used as a backend storage service, the client sends files to upload to the application server, which then forwards them to the OSS. In this process, the data need to be transmitted twice, once from the client to the server, and once from the server to the OSS. In the case of high access volumes, the server needs ample bandwidth resources to satisfy multiple clients’ simultaneous upload needs. This presents a challenge to the architecture’s scalability.

This challenge is solved since the OSS provides the authorized third-party upload function. Using this function, each client can directly upload files to the OSS, rather than going through the server first. This reduces the cost for application servers and takes full advantage of the OSS’s ability to process massive data volumes. Because users do not have to worry about bandwidth and concurrency restrictions, servers focus on service processing.

Currently, there are two ways to grant upload permissions:

URL Signature

URL signature is a way to authorize access. This adds the OSS AccessKeyID and Signature fields in the request URL, allowing users to directly use this URL for an upload. Each URL signature has an expiration time to ensure security. For details, refer to Adding a Signature to the URL.

Temporary Access Credentials

Temporary access credentials are granted through the Alibaba Cloud SecurityTokenService(STS) and provide users with access authorization.For information on the implementation of temporary access credentials, refer to STS Java SDK.

- The client initiates a request to the server to obtain authorization. The server first verifies the client’s legality. If the client is legal, the server uses its own AccessKey to initiate an authorization request to STS. For details, refer to Access Control.
- After the server obtains the temporary credentials, it returns them to the client.
- The client uses these temporary credentials to initiate an upload request to OSS. For the request structure details, refer to Issuing a Temporary Access Credential. The client can cache these credentials and use them for subsequent uploads until they expire. Then, new credentials must be requested from the server.

Best Practices

- RAM and STS User Guide
- Web Client Direct Data Transfer and Upload Callback

Reference Links:

- Upload Callback
- Introduction to Mobile Development Upload Scenarios
- Downloading Uploaded Files
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Access Control for Upload Security
- Form Upload

Upload Callback

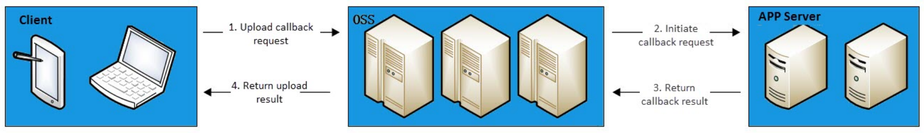
Applicable Scenarios

When an upload is complete, the OSS can perform a callback to the application server. To perform callback, users simply need to attach the relevant Callback parameter to the request sent to OSS. APIs that currently support CallBack include PutObject, PostObject, and CompleteMultipartUpload.

A typical upload callback scenario is when authorized third-party users upload files to the OSS, the clients specify the servers for callback. Then after the upload is complete, the OSS will automatically initiate a callback request to the application server over the HTTP. This promptly notifies the application server that the upload is complete, so it can complete operations such as database modification. Upon receiving a response from the server, the OSS will return the status to the client.

When the OSS sends a POST callback request to the application server, the POST request’ s body will contain parameters that provide certain information. Such parameters are divided into two types: system-defined parameters (such as bucket name and object name) and user-defined parameters. Users can specify user-defined parameters based on the application logic when sending a request including callback to the OSS. User-defined parameters can be used to carry information relevant to the application logic, such as the user ID of the request initiator. For information on user-defined parameters, refer to Callback.

The appropriate use of the upload callback mechanism can decrease the complexity of the client’ s logic and reduce the consumption of network resources. The process is as follows:


Client
1. Upload callback request
4. Return upload result
OSS
2. Initiate callback request
3. Return callback result
APP Server

NOTE

Currently, upload callback is supported only in Chinese Mainland.
At the moment, this function is only available for simple uploads (PutObject) and form uploads (PostObject). Support for multipart uploads will be added on a later date.

Reference for Using the Function:

- API: Callback
- SDK: iOS Callback Notification after Upload

Best Practices:

Web Client Direct Data Transfer and Upload Callback

If a Callback Application Server Is Built (Including Sample Code Download)

Reference Links:

- Direct Data Transfer for a Mobile App
- Permission Management for a Mobile App
- Introduction to Mobile Development Upload Scenarios
- Downloading Uploaded Files
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Access Control for Upload Security
- Authorized Third-party Upload for Upload Security
- Copying, Deleting, and Managing Uploaded Files

File Download

Simple Download

A simple download occurs when a user downloads an uploaded file (object). The object download is accomplished through an HTTP GET request. For the rules of generating object URLs, refer to Accessing OSS.For the access to an object by a user-defined domain name, refer to Accessing OSS with User-defined Domain Names.

When a user accesses a certain object, there are two possibilities:

- This object does not have anonymous read permission, but the user has a corresponding AccessKey, which can be used to sign the GET request and access the object.
- This object has anonymous read permission, so all users can directly access the object through GET requests.

For details about object and bucket access permission control, refer to Access Control.

To authorize a third-party user to download an object from a private bucket, refer to Authorized Third-party Download.

To use resumable download, refer to Resumable Download.

Function usage reference:

- API: Get Object
- SDK: Java SDK-Object
- Console: Getting File Access Addresses

Best Practices:

- RAM and STS User Guide

Reference Links:

- File Upload Methods
- Upload Callback
- Mobile Client Development and Download Scenario Introduction
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Secure Download Access Control
- Authorized Third-party Download for Download Security
- Copying, Deleting, and Managing Uploaded Files

Resumable Download

OSS provides a “start object download from specified point” function. This allows users to spilt large objects into multiple downloads. If the download is interrupted, it will continue from where it left off when restarted.

Just as for simple upload, the user must have read permission for this object. Resumable downloads are supported when the Range parameter is set. This function is recommended for larger objects.For the definition of Range, refer to the HTTP RFC.If the Range parameter is specified in the request header, the returned message contains the length of the entire file and the range returned this time.For example, Content-Range: bytes 0-9/44 indicates that the length of the entire file is 44, and the range returned this time is 0–9. If the range requirement is not met, the system transfers the entire file and does not include Content-Range in the result.The return code is 206.

Function usage reference:

- API: Get Object
- SDK: Java SDK-Multipart File Access

Reference Links:

- File Upload Methods
- Upload Callback
- Mobile Client Development and Download Scenario Introduction
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Secure Download Access Control
- Authorized Third-party Download for Download Security
- Copying, Deleting, and Managing Uploaded Files

Authorized Third-Party Download

When users want to grant a third party authorization to download objects in a private bucket, they should not directly give the AccessKey to the downloader, but use one of the following two methods.

URL Signature

OSS provides a signature download method. The developer can add a signature into the URL and forward this URL to a third party to authorize access.The third-party user can access this URL using an HTTP GET request to download the object.

Implementation Method

Example URL that includes a signature:

```
http://<bucket>.<region>.aliyuncs.com/<object>?OSSAccessKeyId=<user access_key_id>&Expires=<unix
```

```
time>&Signature=<signature_string>
```

The signature in the URL must include at least the following three parameters: Signature, Expires, and OSSAccessKeyID.

- OSSAccessKeyId: the developer’ s AccessKeyId.
- Expires: the developer’ s desired URL expiration time.
- Signature: the developer’ s signature string. For details, refer to API Documentation - signature section.

NOTE: This link must undergo URL encoding.

Specific Implementation

Function usage reference:

- API: Get Object
- SDK: Java SDK-Using URL Signature to Authorize Access
- Console: Getting File Access Addresses

NOTE

On the console, when there is a bucket set to private read/write permission, the retrieved access address will be a URL with a signature. Otherwise, the URL will not have a signature.

Temporary Access Credentials

OSS uses STS (Security Token Service) to provide temporary credentials to third-party users. By adding a signature in the request header, these users can access the object.This authorization method is applicable to mobile scenario downloads.For information on the implementation of temporary access credentials, refer to STS Java SDK.

Implementation Method

Third-party users send a request to the application server to obtain an AccessKeyID, AccessKeySecret, and STS Token issued by STS.They then use the STS AccessKeyID, AccessKeySecret, and STS Token as a signature to request the developer’ s object resource.

Reference for Using the Function:

- API: Temporary Access Credentials
- SDK: Java SDK-**Using STS Service Temporary Authorization** in Object
- Console: Getting File Access Addresses

Best Practices:

- RAM and STS User Guide

Reference Links:

- File Upload Methods
- Upload Callback
- Mobile Client Development and Download Scenario Introduction
- Cloud Processing for Uploaded Images
- Cloud Processing for Uploaded Audio/Video Files
- Secure Download Access Control
- Authorized Third-party Download for Download Security
- Copying, Deleting, and Managing Uploaded Files

File Management

Object Meta

Object Meta describes the attributes of files uploaded to OSS. These attributes come in two types: HTTP standard attributes (HTTP Headers) and User Meta (custom metadata).File metadata can be configured when files are uploaded or copied in various ways.

HTTP Standard Attributes

Name	Description
Cache-Control	Cache action of the web page when the object is downloaded
Content-Disposition	Name of the object when downloaded
Content-Encoding	Content encoding format when the object is downloaded
Content-Language	Specifies the content language encoding when the object is downloaded
Expires	Expiry time
Content-Length	Size of the object
Content-Type	File type of the object
Last-Modified	Time of last modification

User Meta

This attribute is designed for users to enrich the description of objects. In OSS, all parameters prefixed with "x-oss-meta-" are considered as User Meta, such as x-oss-meta-location. A single object can have multiple similar parameters, but the total size of all User Meta cannot exceed 8 KB. User Meta information will be returned in the HTTP header during GetObject or HeadObject operations.

Setting Object Meta When Uploading Objects

Users can set Object Meta when uploading objects.

Reference for Using the Function:

- API: Put Object
- SDK: **Setting Object HTTP Headers** and **Custom Metadata** in the Java SDK-Uploads documentation

Users can set Object Meta when using multipart uploads (resumable data transfer).

Reference for Using the Function:

- API: InitiateMultipartUpload
- SDK: 'Initializing Multipart Upload' in Java SDK-Initializing Multipart Upload

Modifying Object Meta After Uploading Objects

To modify the Object Meta without changing actual data, users should use the copy object interface. To do so, users only need to put the new metadata (note that this metadata must be complete) in the HTTP header and set the copy source and destination addresses to the current address of the object.

Reference for Using the Function:

- API: Copying Objects
- SDK: Java SDK-Using CopyObjectRequest to Copy Objects

Retrieving Object Meta

This feature applies where the user needs to retrieve Object Meta, but not the object data.

Reference for Using the Function:

- API: Head Object
- SDK: Java SDK-Only Retrieve File Metadata

View the Object List

This feature lists the files (objects) uploaded by the user to the bucket. By calling this OSS interface, users can obtain a list of up to 1,000 objects in a certain bucket at a time. The following four parameters provide users with extended capabilities:

Name	Function
Delimiter	Used to group object name characters. All objects whose names are found between the specified prefix and the first occurrence of the Delimiter act as a group of elements: CommonPrefixes.
Marker	Sets up the returned results to begin from the first entry after the Marker in alphabetical order.
MaxKeys	Limits the maximum number of objects returned for one request. If not specified, the default value is 100. The MaxKeys value cannot exceed 1,000.
Prefix	Indicates that only the objects whose Keys contain the specified prefix are returned. Note that the keys returned from queries using a prefix will still contain the prefix.

Folder Simulation

The OSS service does not use folders. All elements are stored as objects. Creating a simulated folder is simply creating an object with a size of 0. This object can also be uploaded and downloaded. The console will display any object ending with "/ " as a folder. Therefore, users can create simulated folders this way.

Users can use a combination of Delimiters and Prefixes to simulate folder functions. Combinations of Delimiter and Prefix serve the following purposes:

- Setting the Prefix as the name of a folder enumerates the files starting with this prefix, recursively returning all files and subfolders (directories) in this folder. The file names are shown in Contents.
- When the Delimiter is set as "/ ", the returned values will enumerate the files in the folder and the subfolders (directories) will be returned in the CommonPrefixes section. Recursive files and folders in subfolders will not be displayed.

```
For example:
In this example, the OSS bucket oss-sample, contains the following objects:

File D
```

```
Directory A/File C
Directory A/File D
Directory A/Directory B/File B
Directory A/Directory B/Directory C/File A
Directory A/Directory C/File A
Directory A/Directory D/File B
Directory B/File A

1. List first-level directories and files
Based on the API request conventions, you must set the Prefix to "", and the Delimiter to "/":
The returned results are as follows:
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult>
<Name>oss-sample</Name>
<Prefix></Prefix>
<Marker></Marker>
<MaxKeys>1000</MaxKeys>
<Delimiter>/</Delimiter>
<IsTruncated>false</IsTruncated>
<Contents>
<Key>File D</Key>
<LastModified>2015-11-06T10:07:11.000Z</LastModified>
<ETag>"8110930DA5E04B1ED5D84D6CC4DC9080"</ETag>
<Type>Normal</Type>
<Size>3340</Size>
<StorageClass>Standard</StorageClass>
<Owner>
<ID>oss</ID>
<DisplayName>oss</DisplayName>
</Owner>
</Contents>
<CommonPrefixes>
<Prefix>Directory A/</Prefix>
</CommonPrefixes>
<CommonPrefixes>
<Prefix>Directory B/</Prefix>
</CommonPrefixes>
</ListBucketResult>

We can see that:
Contents returns the first-level file: "File D".
CommonPrefixes returns the first-level directories: "Directory A/" and "Directory B/", but the files in these
directories are not shown.

2. List second-level directories and files under Directory A
Based on the API request conventions, you must set the Prefix to "Directory A", and the Delimiter to "/":
The returned results are as follows:

<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult>
<Name>oss-sample</Name>
<Prefix>Directory A/</Prefix>
<Marker></Marker>
<MaxKeys>1000</MaxKeys>
<Delimiter>/</Delimiter>
<IsTruncated>false</IsTruncated>
```

```
<Contents>
<Key>Directory A/File C</Key>
<LastModified>2015-11-06T09:36:00.000Z</LastModified>
<ETag>"B026324C6904B2A9CB4B88D6D61C81D1"</ETag>
<Type>Normal</Type>
<Size>2</Size>
<StorageClass>Standard</StorageClass>
<Owner>
<ID>oss</ID>
<DisplayName>oss</DisplayName>
</Owner>
</Contents>
<Contents>
<Key>Directory A/File D</Key>
<LastModified>2015-11-06T09:36:00.000Z</LastModified>
<ETag>"B026324C6904B2A9CB4B88D6D61C81D1"</ETag>
<Type>Normal</Type>
<Size>2</Size>
<StorageClass>Standard</StorageClass>
<Owner>
<ID>oss</ID>
<DisplayName>oss</DisplayName>
</Owner>
</Contents>
<CommonPrefixes>
<Prefix>Directory A/Directory B/</Prefix>
</CommonPrefixes>
<CommonPrefixes>
<Prefix>Directory A/Directory C/</Prefix>
</CommonPrefixes>
<CommonPrefixes>
<Prefix>Directory A/Directory D/</Prefix>
</CommonPrefixes>
</ListBucketResult>

We can see that:
Contents returns the second-level files: "Directory A/File C" and "Directory A/File D".
CommonPrefixes returns the second-level directories: "Directory A/Directory B/", "Directory A/Directory C/", and
"Directory A/Directory D/". The file names under these directories are not shown.
```

Reference for Using the Function:

- API: Get Bucket
- SDK：Java SDK-Listing Files in a Bucket

Copy an Object

This allows users to copy files from a bucket. In certain situations, users simply need to copy an object to another bucket, without modifying its content. In this case, normally we download and then upload the object to the new bucket. However, because the data are the same, this is a waste of network bandwidth. Therefore, the OSS provides the CopyObject function for users to copy objects within the OSS, removing the need to transmit large volumes of data between the user and the OSS.

In addition, because the OSS does not support renaming, it is best to call the OSS CopyObject interface when renaming an object. First copy the original data to an object with a new name and then delete the original file. To modify an object’ s Object Meta only, users can also call the CopyObject interface and set the source address and destination address to the same value. In this way, the OSS will only update the Object Meta.For more information about Object Meta, refer to Object Meta.

Users must note the following when carrying out the operation:

- Users must have permissions to operate the source object. Otherwise the operation may fail.
- This operation cannot copy data across regions. For example, an object in a Hangzhou bucket may not be copied to Qingdao.
- This operation supports objects up to 1 GB.

Reference for Using the Function:

- API: Copy Object
- SDK: Java SDK-Object

Copying Large Objects

Users must take different steps to copy a large object. The OSS supports the function of copying large files similar to Resumable Data Transfer.

The basic operations are the same as those described in Resumable Data Transfer. The one difference is that UploadPart is replaced byUploadPartCopy. The syntax of UploadPartCopy is basically the same as that of UploadPart. However, instead of being directly uploaded from the HTTP request, the data are retrieved from the source object.

Reference for Using the Function:

- API: UploadPartCopy
- SDK：Java SDK-Copying Large Files

Delete an Object

This allows users to delete files (objects) that have been uploaded to OSS buckets. The OSS allows users to delete objects in the following ways:

- Delete Single Object:deletes a specified object.
- Batch Delete:Deletes up to 1,000 objects at a time.

- Auto Delete: This feature applies where large numbers of objects must be deleted according to certain rules, for example, to regularly delete objects that are created a certain number of days ago or to regularly empty the entire bucket. To do so, we recommend Lifecycle Management. Once the rules are specified, the OSS will use these rules to recycle expired objects, helping greatly reduce the number of user requests for deletion and increasing the speed of deletion.

Reference for Using the Function:

- API: Delete Object and Delete Multiple Objects
- SDK：Java SDK-Deleting Files
- Console: Deleting Files

Object Lifecycle Management

The OSS enables users to manage objects through object (file) lifecycle management. The user can configure the lifecycle of a bucket to define various rules for the bucket’s objects. Currently, users can use rules to delete matching objects. Each rule is composed of the following parts:

- The object name prefix; this rule will only apply to objects with the matched prefix.
- Operation; the operation the user wishes to perform on the matched objects.
- Date or number of days; the user will execute the operation on the objects on the specified date or a specified number of days after the object’s last modification time.

A rule applies to an object if the object name prefix matches the rule prefix. For example, a bucket has the following several objects:

```
logs/program.log.1
logs/program.log.2
logs/program.log.3
doc/readme.txt
```

If the prefix of a rule is logs/, the rule applies to the first three objects prefixed with logs/. If the prefix of a rule is doc/readme.txt, the rule only applies to doc/readme.txt.

Currently, rules allow “overdue deletion”. For example, a user can set a rule as follows: If the last update date of objects prefixed with logs/ is 30 days ago, delete the objects. A date can also be specified to delete doc/readme.txt.

When an object matches an overdue rule, the OSS will include the x-oss-expiration header in the response to the GET Object or HEAD Object requests. The header contains two key-value pairs: expiry-date indicates the expiration date of the object; rule-id indicates the matched rule ID.

Users can set the lifecycle configurations of a bucket through the open interface of the OSS. Lifecycle configurations are given in XML format. Below is a specific example.

```
<LifecycleConfiguration>
<Rule>
<ID>delete logs after 10 days</ID>
<Prefix>logs/</Prefix>
<Status>Enabled</Status>
<Expiration>
<Days>10</Days>
</Expiration>
</Rule>

<Rule>
<ID>delete doc</ID>
<Prefix>doc/</Prefix>
<Status>Disabled</Status>
<Expiration>
<CreatedBeforeDate>2014-12-31T00:00:00.000Z</CreatedBeforeDate>
</Expiration>
</Rule>
</LifecycleConfiguration>
```

In the above example, all elements are described as follows:

- **ID**: a unique identifier of each rule.
- **Status**: Enabled or Disabled. The OSS only uses the Enabled rules.
- **Prefix**: the prefix.
- **Expiration**: the operation expiration date. The sub-elements **CreatedBeforeDate** and **Days** specify the absolute and relative expiry time, respectively.
    - **CreatedBeforeDate** indicates that files with a last modification time before 2014-12-31T00:00:00.000Z will be deleted. Objects modified after this time will not be deleted.
    - **Days** indicates that files that were last modified more than 10 days ago will be deleted.

In the first rule, the OSS will delete objects that are prefixed with logs/ and were last updated 10 days ago. The second rule indicates that objects prefixed with doc/ that were last modified before December 31, 2014 will be deleted, but the rule will not take effect because it is in disabled status.

Detail analysis:

1. The naming rules of the prefix are the same as those of the object.
2. When the prefix is empty, the rule applies to all objects in the bucket.
3. Each prefix of a rule must be unique. For example, if a bucket has two rules whose prefixes are respectively logs/ and logs/program, the OSS will return an error.
4. If a rule is set to delete objects on a specific date, the date must be zero o' clock UTC and comply with the ISO8601 format, for example, 2014-01-01T00:00:00.000Z. In the above example, the OSS deleted matched objects after zero o' clock on January 1, 2014.
5. If the number of days is specified in a rule to delete objects, the OSS will sum up the last update time (Last-Modified) and the specified number of days, and then round the sum to

the next zero o’ clock UTC. For example, if the last update time of an object is 01:00 a.m. on April 12, 2014 and the number of days specified in the matched rule is 3, the expiry time is zero o’ clock on April 16, 2014.

6. The OSS deletes the objects matched with the rule at the specified time. Note that objects are usually deleted shortly after the specified time.Usually the last update time of an object is nearly the same as the creation time. If an object is put multiple times, the last update time is the time of the last Put operation. If an object was copied to itself, the last update time is the time when the object was last copied.

Reference for Using the Function:

- API: Put Bucket Lifecycle

Cross-Region Replication

Bucket Cross-Region Replication automatically and asynchronously copies objects in buckets across different OSS data centers. It will synchronize changes to objects in the source bucket (creation, overwriting, deletion, etc.) to the target bucket. This function provides ideal cross-region disaster recovery for buckets or enables users to copy data. The target bucket objects are precise copies of the source bucket objects, with the same object names, metadata, and content (for example, creation time, owner, user-defined metadata, Object ACL, and object content).

Application Scenarios

You may configure bucket Cross-Region Replication for a variety of reasons, including:

- Compliance requirements: Although the OSS creates multiple copies of each stored object on a physical disk, copies must be stored at a certain distance from each other for compliance with established requirements. Through cross-region synchronization, data can be copied between OSS data centers located far apart to satisfy these compliance requirements.
- Minimized latency: Customers may be located at two geographical locations. In order to minimize object access latency, a copy of the object can be maintained at an OSS data center closer to users.
- Data backup and disaster recovery: You require very high data security and availability and wish to explicitly maintain copies of all written data at a second data center for protection from natural disasters. In the event of an earthquake, tsunami, or other event that damages OSS data centers, you can use the backup data in another OSS data center.
- Data copying: For business reasons, you may need to migrate data from one OSS data center to another.
- Operational reasons: You may have computing clusters in different data centers and wish to use them to analyze the same group of objects. Therefore, you may choose to maintain

object copies in these different regions.

Instructions for Use

Currently, cross-region synchronization supports buckets with different names. For two buckets in different regions, the user can sync the data in the source bucket to the target bucket in real time by enabling the synchronization function. The following features are currently supported:

1. Real-time data synchronization: This monitors data addition, deletion, and modification in real time and syncs the changes to the target region bucket. For files of 2M or larger, synchronization may take several minutes. This ensures the ultimate consistency of the data on both sides.
2. Historical data migration: This lets you synchronize historical data in the source bucket as well, forming two completely identical data copies.
3. Real-time synchronization progress retrieval: This shows the latest synchronization time node for real-time data synchronization. For historical data migration, this shows the percentage of data migrated.
4. Easy configuration: The OSS Console provides easy-to-use management interfaces.

Constraints

1. Users can simultaneously operate on two buckets in the synchronization status. However, objects copied from the source bucket may overwrite objects of the same name in the target bucket. Please note this.
2. Because Bucket Replication uses an asynchronous copying method, it may take some time for data to be copied to the target bucket. This may take anywhere from a few minutes to several hours depending on the data size.
3. Cross-region synchronization applies only when the two buckets to be synced do not enable data to be synced to or from a third bucket. For example, if synchronization is activated from Bucket A to Bucket B, the user cannot activate synchronization from Bucket A to Bucket C before deleting the synchronization configuration between Bucket A and Bucket B. Likewise, if synchronization is activated from Bucket A to Bucket B, the user cannot activate synchronization from Bucket C to Bucket B.
4. The two buckets involved in data synchronization must belong to different regions. Data synchronization cannot be performed between buckets in the same region.
5. Currently, cross-region synchronization is available between the Beijing and Shanghai regions. This function will be gradually extended to other regions.

Reference for Using the Function:

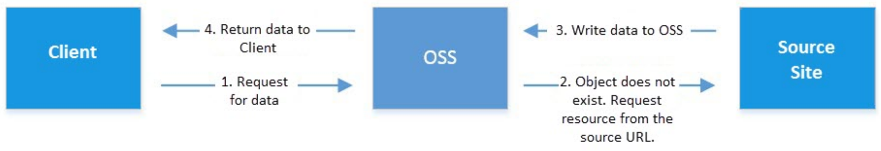
- Console: Cross-Region Replication

Back-to-Source Settings

Back-to-source settings allow for multiple back-to-source reading methods in response to requests for data, meeting users' needs for hot data migration and specific request redirection.

The rules method enables users to match the URL of each OSS Get request and then use a specified method for back-to-source. A maximum of five rules can be configured. Requests are compared to the rules in a set sequence until matched to a valid rule. The specified method can be either image or redirect method.

Image Method



If image write-back is enabled, for a request to get an object that does not exist, the file is requested from the source URL, returned to the user, and simultaneously written to the OSS.

Application Scenarios

Image write-back is primarily used to seamlessly migrate data to OSS. In this situation, a service that is already running on a user-established origin site or on another cloud product needs to be migrated to OSS without interrupting the service. The image write-back function is designed to serve this purpose. An analysis of specific scenarios is given below:

- The origin site has an amount of cold data and is constantly generating new hot dataFrist, the user can use the migration tool to migrate the cold data to the OSS (this migration tool is ossimport2 , and at the same time, the user can configure image write-back and set the origin site' s URL to OSS. Even if some newly generated data are not migrated when the domain name is switched to OSS (or Alibaba Cloud CDN, with OSS back-to-source), the user can still access it normally on OSS and the files will be saved to OSS after they have been accessed for the first time. After switching the domain name for an origin site that no longer produces new data, the site will be scanned once and all non-migrated data will be imported to the OSS at a time. In this situation, the user may disable image write-back.

If the configured origin site is an IP address, after the domain name is migrated to the OSS, data can still be imaged to the origin site. However, if it is a domain name, no image can be produced because the domain name is resolved to the OSS or CDN. In this situation, the user can apply for another domain name to image the origin site. This domain name and the in-service domain name would both be resolved to the same IP address. This allows origin site imaging to continue when the service domain name is migrated.

- Only some origin site traffic is switched to the OSS or CDN, while the origin site continually produces dataThe migration method is similar to that described in scenario 1. After switching only a portion of traffic to the OSS, the user does not have to delete the image write-back configuration. This ensures that the traffic switched to OSS or CDN can also obtain data from the origin site.

Usage Rules

1. The OSS only executes image write-back to request an object from the origin site when GetObject() returns a 404 code.
2. The URL requested from the origin site is ‘MirrorURL+object’ and the name of the file written back to the OSS is “object”. For example, assume that a bucket is named example-bucket, image write-back is configured, the MirrorURL is ‘http://www.example-domain.com/ ’, and the file ‘image/example_object.jpg’ does not exist in this bucket. Then, when downloading this file, the OSS will initiate a Get request to ‘ http://www.example-domain.com/image/example_object.jpg ’ and return the result to the user while at the same time writing it to the OSS. Once downloaded, the file will be present on OSS as ‘image/example_object.jpg’. This is the same as migrating an object with the same name to the OSS. If the MirrorURL carries path information, such as ‘ http://www.example-domain.com/dir1/ ’, the process is the same as above, but the OSS back-to-source URL will be ‘http://www.example-domain.com/dir1/image/example_object.jpg ’ although the object written to the OSS will remain image/example_object.jpg. This is the same as migrating an object from an origin site directory to the OSS.
3. The header and querystring information transmitted to the OSS will not be sent to the origin site.
4. If the origin site returns data in chunked code, the OSS will likewise return data to the user in chunked code.
5. The OSS will return and save the following header information from the origin site to the OSS:

```
Content-Type
Content-Encoding
Content-Disposition
Cache-Control
Expires
Content-Language
Access-Control-Allow-Origin
```

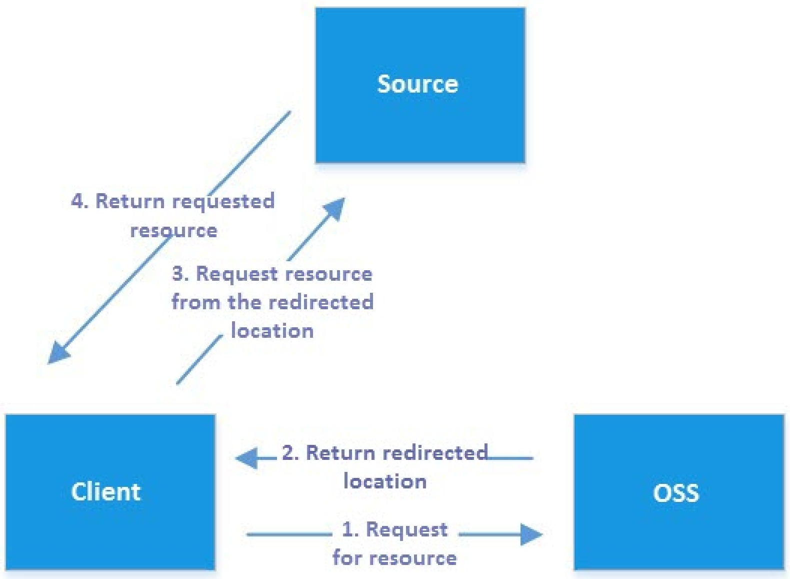
6. An x-oss-tag response header will be added to image write-back files, with the value “MIRROR” + space + url_decode(back-to-source URL). In the example given above, this would be ‘x-oss-tag:MIRROR http%3a%2f%2fwww.example-

domain.com%2fdir1%2fimage%2fexample_object.jpg’ . After the file is written back to the OSS, so long as it is not overwritten again, this header will be added each time it is downloaded to indicate that it is taken from a image.

7. Assuming that the file has already been written to the OSS through image write-back, if the corresponding file on the origin site is changed, the OSS will not update the file that exists on the OSS because this file which is already present on the OSS does not meet the image write-back conditions.
8. If the file also does not exist in the image source (i.e. the image source returns the HTTP status 404 to the OSS), the OSS will return 404 to the user. If the image source returns another non-200 status code (including file retrieval failure due to network-related causes), the OSS will return 424 to the user, the error code for ‘MirrorFailed’ .

Redirection

The URL redirection function returns a 3xx hop to the user based on the user-defined conditions and corresponding hop configuration. Users can use this hop function to redirect files and provide various services based on this action.


Source
4. Return requested resource
3. Request resource from the redirected location
Client
2. Return redirected location
1. Request for resource
OSS

1. Seamlessly migrating other data sources to OSS: Users can asynchronously migrate data from their data sources to the OSS. In this process, requests for un-migrated data use the URL rewrite method to return a 302 redirect request to the user. The user’ s client will then read back the data from the user’ s data source based on the location in the 302 redirect request.
2. Used to configure page jump functions: For example, if a user wishes to hide objects with a certain header prefix, a special page can be returned to visitors.
3. To configure page jumps when a 404 or 500 error occurs: When this type of error occurs,

the user can be taken to a preset page. Therefore, OSS errors will not be completely exposed to users when the system goes wrong.

Reference for Using the Function:

- Console: Back-To-Source Rule Management

Security Management

Identity Authentication for visitors

Sending an OSS Access Request

OSS developers can access the OSS directly by calling a RESTful API provided by the OSS or using an API-encapsulated SDK. Each request for access to the OSS requires identity verification or direct anonymous access based on the current bucket permission and operation.

Access to OSS resources is divided into access by the owner and third-party access. Here, the owner refers to the bucket owner, also known as “developer”. Third-party users are users who access resources in a bucket.There are two access methods: anonymous access and signature-based access. In the OSS, a request that does not contain any identification information is considered anonymous access.Signature-based access refers to requests that, according to the rules in the OSS API documentation, contain signature information in the request header or URL.

Types of AccessKeys

Currently, there are three types of AccessKeys (AccessKey) for OSS access. They are described below:

Alibaba Cloud Account AccessKeys

These are the AccessKeys of bucket owners. The AccessKey provided by each Alibaba Cloud account has full access to its own resources. Each Alibaba Cloud account can simultaneously have 0 to 5 active or inactive AccessKey pairs (AccessKeyID and AccessKeySecret). You can log in to Console and add or delete AccessKey pairs on AccessKey Console. Each AccessKey pair may be in two states: active and inactive.

- Active indicates that the user’s AccessKey is in the active state and can be used for identity

authentication.
- Inactive indicates that the user’ s AccessKey is in the inactive state and cannot be used for identity authentication.

The AccessKey of the Alibaba Cloud account should not be directly used unless necessary.

RAM Account AccessKeys

Resource Access Management (RAM) is a resource access control service provided by Alibaba Cloud. RAM account AKs are the access keys granted by RAM. These AKs only allow access to resources in a bucket according to the rules defined by RAM. RAM helps you to collectively manage your users (such as employees, systems or applications) and controls which resources your users can access. For example, you can allow your users to have only the read permission on a bucket.Subaccounts are subordinate to normal accounts and cannot own any actual resources. All resources belong to primary accounts.

STS Account AccessKeys

The AliCloud STS (Security Token Service) is a service that provides temporary access credentials. STS account AKs are the AKs issued by the STS. These AKs only allow access to buckets in accordance with the rules defined by the STS.

Implementation of Identity Authentication

Currently, there are three methods of authentication:

- AK authentication
- RAM authentication
- STS authentication

Before sending a request to the OSS as an individual identity, a user needs to generate a signature string for the request according to the format specified by the OSS and then encrypt the signature string using the AccessKeySecret to generate a verification code.After receiving the request, the OSS finds the corresponding AccessKeySecret based on the AccessKeyID, and obtains the signature string and verification code in the same way. If the obtained verification code is the same as the provided verification code, the request is assumed valid. If not, the OSS rejects the request and returns an HTTP 403 error.Users can directly use the SDKs provided by the OSS with different types of AccessKeys for different types of identity authentication.

Permission Control

OSS provides various permission control mechanisms for access to its stored objects:

- Bucket-level permissions
- Object-level permissions
- Account-level permissions (RAM)
- Temporary account permissions (STS)

Bucket-level Permissions

Bucket Permission Types

The OSS provides an Access Control List (ACL) for permission control. The OSS ACL provides bucket-level access control. Currently, three access permissions are provided for a bucket: public-read-write, public-read, and private. They are described as follows:

Permission	Access Restriction
public-read-write	Anyone (including anonymous users) can read, write, and delete the objects in the bucket. The fees incurred by such operations shall be borne by the creator of the bucket. **Please use this permission with caution**.
public-read	Only the creator of a bucket can write or delete the objects in the bucket. Anyone (including anonymous users) can read the objects in the bucket.
private	Only the creator of a bucket can read, write, and delete the objects in the bucket. Others cannot access the objects in the bucket without authorization.

Bucket Permission Settings and Read Methods

Function usage reference:

- API: Put BucketACL
- SDK: Java SDK-Set Bucket ACL
- Console: Create Bucket Permission Setting
- API: Get BucketACL
- SDK: Java SDK-Obtain Bucket ACL

Object-level Permissions

Object Permission Types

The OSS ACL also provides object-level permission access control.Currently, four access permissions are available for an object, including private, public-read, public-read-write and default. You can use the “x-oss-object-acl” header in the Put Object ACL request to set the access permission. Only the bucket owner has the permission to perform this operation.

Permission	Access Restriction
public-read-write	Indicates that the object can be read and written by the public. That is, all users have the permission to read and write the object.
public-read	Indicates that the object can be read by the public. Only the owner of the object has the permission to read and write the object. Other users only have the permission to read the object.
private	Indicates that the object is a private resource. Only the owner of the object has the permission to read and write the object. Other users have no permission to operate the object.
default	Indicates that the object inherits the permission of the bucket.

Considerations

- If no ACL is configured for an object, the object uses the default ACL, indicating that the object has the same ACL as the bucket where the object is stored.
- If an ACL is configured for an object, the object ACL has higher-level permission than the bucket ACL. For example, an object with the public-read permission can be accessed by authenticated users and anonymous users, regardless of the bucket permission.

Object Permission Settings and Read Methods

Function usage reference:

- API: Put Object ACL
- SDK: Java SDK-Set the object ACL in ObjectACL
- API: Get Object ACL
- SDK: Java SDK-Read the object ACL from ObjectACL

Account-level Permissions (RAM)

Application Scenarios

If you have purchased cloud resources and multiple users in your organization need to use them, these users have to share the AccessKey of your Alibaba Cloud account. There are two problems:

- If your key is shared by many people, it has a high risk of leakage.

- You cannot determine which resources (e.g. buckets) can be accessed by the users.

Solution: Under your Alibaba Cloud account, you can use RAM to create subusers with their own AccessKeys. In this case, your Alibaba Cloud account will be the primary account and the created accounts will be subaccounts. Subaccounts can only use their AccessKeys for the operations and resources authorized by the primary account.

Specific Implementation

For details about the RAM, refer to RAM User Manual. The RAM User Manual describes how to grant permissions, create RAM accounts, and manage group permissions in detail.For details about how to configure the policies required in authorization, refer to the final section of this chapter.

Temporary Account Permissions (STS)

Application Scenarios

Users managed by your local identity system, such as your app users, your local corporate account, or third-party apps, may also directly access OSS resources. They are called federated users. In addition, users can also be the applications you create that have access to your AliCloud resources.

With respect to these federated users, short-term access permission management is provided for the AliCloud account (or RAM users) through the Security Token Service (STS) of AliCloud. You do not need to reveal the long-term key (such as the login password and AccessKey) of your AliCloud account (or RAM users), but only need to create a short-term access credential for a federated user. The access permission and validity of this credential are both up to you.You do not need to care about permission revocation. The access credential automatically becomes invalid when it expires.

STS-based access credentials include the security token (SecurityToken) and the temporary access key (AccessKeyId and AccessKeySecret). The AccessKey method is the same as the method of using the AccessKey of the Alibaba Cloud account or RAM user. In addition, each OSS access request must carry a security token.

Specific Implementation

For details about the STS, refer to Role Management in the RAM User Guide. The key is to call AssumeRole of the STS interface to obtain valid access credential. You can also directly use STS SDK to call the access credential.

For details about role management and usage, refer to Role Management in the RAM User Guide.

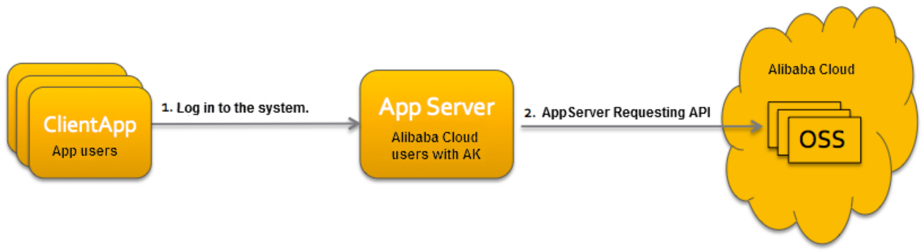
For details about how to configure the policies required in authorization, refer to the final section of this chapter.

RAM and STS Application Scenario Practices

In different application scenarios, how the access identity is verified may vary. The following describes two methods for access identity verification in typical application scenarios.

A mobile app is used as an example. Assume that you are a mobile app developer. You attempt to use the Alibaba Cloud OSS to store end user data of the app. You also have to ensure data is isolated between app users to prevent an app user from obtaining data of other app users.

Mode 1: Using AppServer for Data Transit and Data Isolation

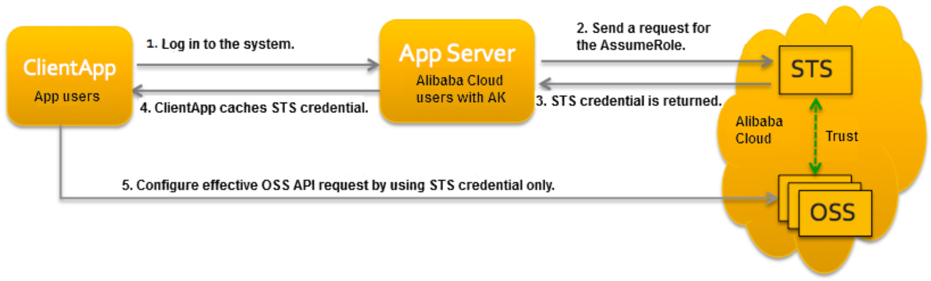


As shown in the figure above, you need to develop an AppServer. Only the AppServer can access the ECS. The ClientApp can read or write data only through the AppServer. The AppServer ensures isolated access to different user data.

In this method, you can use the key provided by your AliCloud account or RAM account for signature verification. In case of any security problem, you are recommended not to directly use the key of your AliCloud account (root account) to access the OSS.

Mode 2: Using STS For Direct Access to OSS

The STS solution is shown below:


ClientApp
App users
1. Log in to the system.
4. ClientApp caches STS credential.
App Server
Alibaba Cloud users with AK
2. Send a request for the AssumeRole.
3. STS credential is returned.
STS
Alibaba Cloud
Trust
5. Configure effective OSS API request by using STS credential only.
OSS

The solution is described in detail as follows:

1. Log in as the app user. The app user is irrelative to the Alibaba Cloud account but is an end user of the app. The AppServer allows the app user to log in. For each valid app user, the AppServer needs to define the minimum access permission for them.
2. The AppServer requests a security token (SecurityToken) from the STS. Before calling STS, the AppServer needs to determine the minimum access permission (described in policy syntax) of app users and the expiration time of the authorization.Then, the AppServer uses AssumeRole to obtain a security token indicating a role. For details about role management and usage, refer to Role Management in the RAM User Guide.

3. The STS returns a valid access credential to the AppServer, where the access credential includes a security token, a temporary access key (AccessKeyID and AccessKeySecret), and the expiry time.
4. The AppServer returns the access credential to the ClientApp. The ClientApp caches this credential.When the credential becomes invalid, the ClientApp needs to request a new valid access credential from the AppServer. For example, if the access credential is valid for one hour, the ClientApp can request the AppServer to update the access credential every 30 minutes.
5. The ClientApp uses the access credential cached locally to request Alibaba Cloud Service APIs. The ECS perceives the STS access credential, relies on STS to verify the credential, and correctly responds to the user request.

RAM and STS Authorization Policy Configuration

The detailed rules of the use of policies during RAM or STS authorization are as follows.

Example

First, let’ s look at the following policy example:

```
{
"Version": "1",
"Statement": [
{
"Action": [
"oss:GetBucketAcl",
"oss:ListObjects"
],
"Resource": [
"acs:oss:*:1775305056529849:mybucket"
],
"Effect": "Allow",
"Condition": {
"StringEquals": {
"acs:UserAgent": "java-sdk",
"oss:Delimiter": "/",
"oss:Prefix": "foo"
},
"IpAddress": {
"acs:SourceIp": "192.168.0.0"
}
}
},
{
"Action": [
"oss:PutObject",
"oss:GetObject",
"oss:DeleteObject"
],
"Resource": [
"acs:oss:*:1775305056529849:mybucket/file*"
],
"Effect": "Allow",
"Condition": {
"IpAddress": {
"acs:SourceIp": "192.168.0.0"
}
}
}
]
}
```

This is an authorization policy. You can use this policy to grant permissions for users through RAM or STS.The policy has a Statement (one policy can have multiple Statements). In the Statement, Action, Resource, Effect, and Condition are specified.

This policy authorizes your ‘mybucket’ and ‘mybucket/file*’ resources to corresponding users and supports GetBucketAcl, GetBucket, PutObject, GetObject, and DeleteObject actions. The Condition indicates that authentication is successful and authorized users can access related resources only when UserAgent is java-sdk and the source IP address is 192.168.0.1.The Prefix and Delimiter conditions apply during the GetBucket (ListObjects) action. For details about the two fields, see OSS API Documentation.

Configuration Rules

Version

Policy version is defined. For configuration method in this document, it is set to “1”.

Statement

The Statement describes the authorization meaning. It can contain multiple meanings based on the business scenario. Each meaning includes a description of the Action, Effect, Resource, and Condition. The request system will check each statement for a match one by one. All successfully matched statements will be divided into Allow and Deny based on the difference of Effect settings, and Deny is given priority. If the matches are all Allow, the request passes authentication. If one of the matches is Deny or there are no matches, this request is denied to access.

Action

Actions fall into two categories: bucket-level actions and object-level actions. Bucket-level actions include oss:PutBucketAcl and oss:GetBucketLocation. The action objects are buckets and the action names correspond to the involved interfaces in a one-to-one manner. Object-level actions include oss:GetObject, oss:PutObject, oss:DeleteObject, oss:DeleteObject, and oss:AbortMultipartUpload. If you want to authorize actions for a type of object, you can select one or more of the above actions. In addition, all action names must be prefixed with "oss:", as shown in the example above. Action is a list. There can be multiple Actions. The mapping between Actions and APIs is as follows:

Server-level

API	Action
GetService (ListBuckets)	oss:ListBuckets

Bucket-level

API	Action
PutBucket	oss:PutBucket
GetBucket (ListObjects)	oss:ListObjects
PutBucketAcl	oss:PutBucketAcl
DeleteBucket	oss:DeleteBucket
GetBucketLocation	oss:GetBucketLocation
GetBucketAcl	oss:GetBucketAcl
GetBucketLogging	oss:GetBucketLogging
PutBucketLogging	oss:PutBucketLogging
DeleteBucketLogging	oss:DeleteBucketLogging
GetBucketWebsite	oss:GetBucketWebsite
PutBucketWebsite	oss:PutBucketWebsite
DeleteBucketWebsite	oss:DeleteBucketWebsite
GetBucketReferer	oss:GetBucketReferer
PutBucketReferer	oss:PutBucketReferer
GetBucketLifecycle	oss:GetBucketLifecycle
PutBucketLifecycle	oss:PutBucketLifecycle
DeleteBucketLifecycle	oss:DeleteBucketLifecycle
ListMultipartUploads	oss:ListMultipartUploads
PutBucketCors	oss:PutBucketCors
GetBucketCors	oss:GetBucketCors
DeleteBucketCors	oss:DeleteBucketCors
PutBucketReplication	oss:PutBucketReplication
GetBucketReplication	oss:GetBucketReplication

DeleteBucketReplication	oss:DeleteBucketReplication
GetBucketReplicationLocation	oss:GetBucketReplicationLocation |
GetBucketReplicationProgress	oss:GetBucketReplicationProgress |

Object level

API	Action
GetObject	oss:GetObject
HeadObject	oss:GetObject
PutObject	oss:PutObject
PostObject	oss:PutObject
InitiateMultipartUpload	oss:PutObject
UploadPart	oss:PutObject
CompleteMultipart	oss:PutObject
DeleteObject	oss:DeleteObject
DeleteMultipartObjects	oss:DeleteObject
AbortMultipartUpload	oss:AbortMultipartUpload
ListParts	oss:ListParts
CopyObject	oss:GetObject,oss:PutObject
UploadPartCopy	oss:GetObject,oss:PutObject
AppendObject	oss:PutObject
GetObjectAcl	oss:GetObjectAcl
PutObjectAcl	oss:PutObjectAcl

Resource

Resource stands for a specific resource or resources on the OSS (the wildcard is supported). Resources are named in the format of “acs:oss:region:bucket_owner:bucket_name/object_name”. For all bucket-level actions, the final part “/object_name” is not required. You can just render it as “acs:oss:region:bucket_owner:bucket_name”. Resource is also a list and there can be multiple Resources. Here, the region field is currently not supported and set as “*”.

Effect

Effect indicates the authorization result of the Statement. Two value options are available: Allow and Deny. When there are multiple Statement matches, the Deny is given higher priority.

Condition

Condition indicates the conditions for the authorization policy. In the above example, you can set check conditions for acs:UserAgent and acs:SourceIp. The oss:Delimiter and oss:Prefix fields are used to restrict resources during the GetBucket action.

The OSS supports the following conditions:

Condition	Function	Valid Value
acs:SourceIp	Specifying the IP address segment	Common IP address, wildcard (*) supported
acs:UserAgent	Specifying the http useragent header	String
acs:CurrentTime	Specifying valid access time	ISO8601 format
acs:SecureTransport	Whether HTTPS is used	“true” or “false”
oss:Prefix	Used as the prefix for ListObjects	Valid object name
oss:Delimiter	Used as the delimiter for ListObject	Valid delimiter value

More Examples

For more examples of authorization policies in specific scenarios, click here；For convenient online graphical policy configuration tools, click here.

Best Practices:

RAM and STS User Guide

Access Logging Configuration

The OSS provides users with automatic saving of server access logs. A bucket owner can log in to OSS Consoleto enable the server access logging feature for all the owner’ s buckets. When access logging is activated for a bucket (Source Bucket), the OSS will generate an object containing all access request logs of that bucket (by hour) and write the object into the user-designated bucket (Target Bucket) according to fixed naming rules.

Object Naming Rules for Access Logging

```
<TargetPrefix><SourceBucket>-YYYY-mm-DD-HH-MM-SS-UniqueString
```

In the naming rules, the TargetPrefix is specified by the user; YYYY, mm, DD, HH, MM and SS give the year, month, day, hour, minutes and seconds of the creation time in Arabic numerals (note the digits); and UniqueString is the string generated by the OSS system. An example for the name of an object actually used to store OSS access logs is given below:

```
MyLog-oss-example-2012-09-10-04-00-00-0000
```

In the above example, “MyLog-” is the Object prefix specified by the user; “oss-example” is the name of the origin bucket; “2012-09-10-04-00-00” is the Object creation time (Beijing time); and “0000” is the string generated by the OSS system.

Log File Format

(Separated by spaces from left to right):

Name	Example	Description
Remote IP	119.140.142.11	IP address from which the request is initiated (the proxy or user firewall may block this field)
Reserved	-	Reserved field
Reserved	-	Reserved field
Time	[02/May/2012:00:00:04 +0800]	Time when the OSS receives the request
Request-URI	“GET /aliyun-logo.png HTTP/1.1”	User-Requested URI (including query-string)
HTTP Status	200	HTTP status code returned by the OSS
SentBytes	5576	Traffic that the user downloads from the OSS
RequestTime (ms)	71	Time spent in completing this request (in ms)
Referer	http://www.aliyun.com/product/oss	Requested TTP Referer
User-Agent	curl/7.15.5	HTTP User-Agent header
HostName	oss-example.oss-cn-hangzhou.aliyuncs.com	Domain name for access request
Request ID	505B01695037C2AF032593A4	UUID used to uniquely identify this request
LoggingFlag	true	Whether the access logging function is enabled
Reserved	-	Reserved field

Requester AliCloud ID	1657136103983691	AliCloud ID of the requester, “- “ for anonymous access
Operation	GetObject	Request type
Bucket	oss-example	Name of the bucket requested for access
Key	/aliyun-logo.png	User-Requested Key
ObjectSize	5576	Object size
Server Cost Time (ms)	17	Time taken by the OSS server to process this request (in ms)
Error Code	NoSuchBucket	Error code returned by the OSS
Request Length	302	Length of user request (byte)
UserID	1657136103983691	ID of the bucket owner
Delta DataSize	280	Bucket size variation, “- “ for no change
Sync Request	-	Whether this is a back-to-source request from CND, “- “ for no
Reserved	-	Reserved field

Detail Analysis

1. The source bucket and target bucket must belong to the same user.
2. TargetPrefix indicates the name prefix of the object used for storing access logs. The field can be left blank.
3. The source bucket and target bucket can be the same or different buckets. You can save logs from multiple source buckets to the same target bucket (in this case, it is recommended that you assign different values to TargetPrefix).
4. The OSS generates a bucket access log file every hour. However, all requests in the hour may not be recorded in the log file, but may be recorded in the previous or next log file.
5. In the naming rules for log files generated by the OSS, “UniqueString” is just a UUID that the OSS generates for an object to uniquely identify the file.
6. Each time the OSS generates a bucket access log file, this is considered a PUT operation and the occupied space is recorded, but the generated traffic is not recorded. After log files are generated, you can operate these log files as common objects.
7. The OSS ignores all query-string parameters prefixed by “x- “ but such query-string parameters are recorded in access logs. If you want to mark a special request from massive access logs, you can add a query-string parameter prefixed by “x- “ to the URL. For example:http://oss-example.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.pnghttp://oss-example.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png?x-user=adminWhen the OSS

processes the above two requests, the results are the same. However, you can search access logs with “x-user=admin” to quickly locate the marked request.

8. You may see “- “ in any field of OSS logs. It indicates that data is unknown or the field is invalid for the current request.Certain fields will be added to the end of OSS log files in the future based on the requirements. It is recommended that developers take compatibility issues into consideration when developing log processing tools.

Function usage reference:

- Console: Server Access Logging

Anti-leech Settings

The OSS collects service fees based on use. To prevent users’ data on OSS from being leeched, OSS supports anti-leech based on the field referer in the HTTP header. Users can log in to OSS Console or use APIs to configure a referer white list for a bucket or whether to allow access by requests where referer is blank. For example, for a bucket named oss-example, set its referer white list to ‘http://www.aliyun.com ’. Then, only requests with a referer of ‘http://www.aliyun.com ’ can access the objects in the bucket.

Detail Analysis

1. Anti-leech verification will be performed only when users access objects through URL signatures or anonymously. When the request header contains the “Authorization” field, anti-leech verification is not performed.
2. A bucket supports multiple referer fields, which are separated by the comma “,” .
3. The referer field supports the wildcard “* “ and “?”。
4. Users can set whether to allow access requests with empty referer fields.
5. When the white list is empty, the system will not check if the referer field is null (otherwise, all requests will be rejected).
6. When the white list is not empty and the rules do not allow null referer fields, only requests with referers in the white list will be allowed. Other requests (including null referer requests) will be rejected.
7. If the white list is not empty and the rules allow empty referer fields, requests with empty referer and with the referers in the white list will be allowed. Other requests will be rejected.
8. The three bucket permissions (private, public-read, and public-read-write) will all check the referer field.

Wildcard details:

- Asterisk “” :The asterisk can be used to represent 0 or multiple characters. If you are looking for an object name prefixed with AEW but have forgotten the remaining part, you can enter AEW* to search for all types of files starting with AEW, such as AEWT.txt, AEWU.EXE and AEWI.dll. If you want to narrow down the search scope, you can enter AEW*.txt to search for all .txt files starting with AEW, such as AEWIP.txt and AEWDF.txt.

- Question mark “?”: The question mark can be used to represent one character. If you enter love?, all types of files starting with love and ending with one character will be displayed, such as lovey and lovei. If you want to narrow the search scope, you can enter love?.doc to search for all .doc files starting with love and ending with one character, such as lovey.doc and loveh.doc.

Function usage reference:

- API: Put Bucket Referer
- Console: Anti-leech Settings

Cross-origin Resource Sharing Configuration

Cross-origin access, or the cross-origin of JavaScript, is a browser restriction set for the sake of security, namely, the same-origin policy. When Website A tries to use the JavaScript code in its webpage to access Website B, the attempt will be rejected by the browser because A and B are two websites of different origins.

Cross-origin access needs arise frequently in actual usage, such as when OSS is used at the back end for the user’s website www.a.com.The upload function implemented with JavaScript is provided in the webpage. However, requests could only be sent to www.a.com in the webpage, and all the requests sent to other websites are rejected by the browser. Thus the data uploaded by users has to be relayed to other sites via www.a.com.If cross-origin access is set, users could upload their data directly to OSS instead of relaying it via www.a.com.

Cross-origin Resource Sharing (CORS) is the standard across-origin solution provided by HTML5. Currently, the CORS standard is supported by OSS for cross-origin access. For details about the specific CORS rules, refer to W3C CORS Norms. In simple terms, CORS indicates the origin of where the request is originated by using a header containing the origin of the HTTP request. As in the previous example, the origin header contains www.a.com.After receiving the request, the server will judge based on certain rules whether the request should be accepted or not. If yes, the server will attach the Access-Control-Allow-Origin header in the response. The header contains www.a.com, indicating that cross-origin access is allowed. In case that the server accepts all the cross-origin requests, just set the Access-Control-Allow-Origin header to *. The browser will determine whether the cross-origin request is successful or not based on whether the corresponding header has been returned or not. In case that no corresponding header is attached, the browser will block the request. In case that the request is not a simple one, the browser will firstly send an OPTIONS request to obtain the CORS configuration of the server. In case that the server does not support the following operations, the browser will also block the following requests.

OSS provides the configuration of the CORS rule, accepting or rejecting corresponding cross-origin requests as needed. The rule is configured at the bucket level. The details are available in PutBucketCORS.

Here are several key points:

- Attaching relevant CORS headers and other actions are automatically executed by the browser, and no additional action is required by the user. Only in the browser environment could the CORS operations be meaningful.
- Whether a CORS request is accepted is completely independent of OSS authentication and other such measures, i.e. the OSS CORS rule is only used to determine whether to attach the relevant CORS headers. Whether the request should be blocked should be exclusively determined by the browser.
- When using cross-origin requests, make sure the browser’ s cache function is enabled. For example, the same cross-origin resource have been requested by two webpages running on the same browser (originated from www.a.com and www.b.com) at the same time respectively. If the request of www.a.com is received by the server in the first place, the server will return to the user the resource with the Access-Control-Allow-Origin header “www.a.com”. When www.b.com initiates its request, the browser will return its previous cached request to the user. As the header content does not match the CORS request, the subsequent request fails.

Function usage reference:

- API: Cross-origin Resource Sharing
- SDK: Java SDK-Cross-origin Resource Sharing
- Console: Cross-origin Resource Sharing

Server-Side Encryption

OSS supports server-side encryption of data uploaded by users: When a user uploads data, the OSS encrypts the received user data and permanently stores the encrypted data. When a user downloads data, the OSS automatically decrypts the encrypted data, returns the original data to the user, and declares in the header of the returned HTTP request that the data has been encrypted on the server side. In other words, there is major big difference between downloading an object encrypted on the server side and downloading a common object, because the OSS manages the entire codec process for users.Currently, the OSS’ s server-side encryption is an attribute of objects. When creating an object, a user only needs to add the HTTP Header “x-oss-server-side-encryption” to the Put Object request and specify its value as “AES256”. Then, the object can be encrypted on the server side before it is stored. Currently, server-side encryption is supported by the following operations:

- Put Object
- Copy Object

- Initiate Multipart Upload

Detail Analysis

1. Except the Put Object, Copy Object, and Initiate Multipart Upload requests, if any other request received by the OSS contains the ‘x-oss-server-side-encryption’ header, the OSS will directly return HTTP Status Code 400, with the error code in the message body being InvalidArgument.
2. Currently, the OSS only supports the AES256 encryption algorithm. If the user specifies another value for the ‘x-oss-server-side-encryption’ header, the OSS will directly return HTTP Status Code 400, with the error code in the message body being ‘InvalidEncryptionAlgorithmError’.
3. For objects stored after server-side encryption, the OSS returns the x-oss-server-side-encryption header in the API requests below, with its value being the entropy encryption algorithm:
    - Put Object
    - Copy Object
    - Initiate Multipart Upload
    - Upload Part
    - Complete Multipart Upload
    - Get Object
    - Head Object

Specific Implementation

- API: Append Object
- API: Put Object
- API: Copy Object
- API: Post Object

Static Website Hosting

The OSS supports static website hosting. On the OSS Console users can set up their storage space to work in static website hosting mode. After the configuration takes effect,The domain name of a static website hosted in bucket located in the Hangzhou region is as follows:

```
http://<Bucket>.oss-cn-hangzhou.aliyuncs.com/
```

For users to manage static websites hosted on the OSS more easily, the OSS provides two functions:

- Index Document SupportThe index document refers to the default index document (equivalent to index.html of the website) returned by the OSS when a user directly accesses the root domain name of the static website. If you have set static website hosting mode for a bucket, you have to specify an index document.

- Error Document SupportThe error document refers to the error page the OSS returns to a user if the HTTP 4XX error (the most typical error is 404 “NOT FOUNT”) occurs when the user accesses the static website. By specifying the error page, you can provide your users with appropriate error prompts.

For example: The user sets the index document support as index.html, the error document support as error.html, the bucket as oss-sample, and the endpoint as oss-cn-hangzhou.aliyuncs.com. Thus:

- When the user accesses http://oss-sample.oss-cn-hangzhou.aliyuncs.com/ and http://oss-sample.oss-cn-hangzhou.aliyuncs.com/directory/,This is the same as accessing http://oss-sample.oss-cn-hangzhou.aliyuncs.com/index.html
- When the user accesses http://oss-sample.oss-cn-hangzhou.aliyuncs.com/object, OSS will return http://oss-sample.oss-cn-hangzhou.aliyuncs.com/error.html if the object does not exist.

Detail Analysis

1. Static websites are websites where all web pages are composed of static content, including scripts such as JavaScript executed on the client. The OSS does not support content that needs to be processed by the server, such as PHP, JSP, and APS.NET.
2. For access to a bucket-based static website through a user-defined domain name, you can use Domain Name CNAME.
3. Because the OSS restricts access by bucket domain names, users’ files cannot be directly viewed in a browser. Users are recommended to use CNAMEs.
4. When a user sets static website hosting mode for a bucket, the index page must be specified and the error page is optional.
5. When a user sets a bucket to static website hosting mode, the specified index page and error page must be objects in this bucket.
6. After a bucket is set to static website hosting mode, the OSS returns the index page for anonymous access to the root domain name of the static website, and returns Get Bucket results for signed access to the root domain name of the static website.
7. After static website hosting mode is set for a bucket and the user accesses the root domain name of a static website or a nonexistent object, the OSS will return a specified object to the user and bills the return traffic and requests.

Reference for Using the Function:

- API: Put Bucket Website
- Console: Static Website Hosting