

# Cloud Object Storage

## Practical Tutorial

### Product Documentation



## Copyright Notice

©2013-2025 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice

 Tencent Cloud

All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## Practical Tutorial

- Overview

- Access Control and Permission Management

  - ACL Practices

  - CAM Practices

  - Granting Sub-Accounts Access to COS

  - Authorization Cases

  - Working with COS API Authorization Policies

  - Security Guidelines for Using Temporary Credentials for Direct Upload from Frontend to COS

  - Generating and Using Temporary Keys

  - Authorizing Sub-Account to Get Buckets by Tag

  - Descriptions and Use Cases of Condition Keys

  - Granting Bucket Permissions to a Sub-Account that is Under Another Root Account

- Performance Optimization

  - Request Rate and Performance Optimization

  - Working with COSBench

- Data Migration

  - Migrating Local Data to COS

  - Migrating Data from Third-Party Cloud Storage Service to COS

  - Migrating Data from URL to COS

  - Migrating Data Within COS

  - Migrating Data Between HDFS and COS

- Accessing COS with AWS S3 SDK

- Data Disaster Recovery and Backup

  - Disaster Recovery and High Availability Architecture Based on Cross-Bucket Replication

  - Cloud Data Backup

  - Local Data Backup

- Domain Name Management Practice

  - Switch bucket to custom domain

  - Supporting HTTPS for Custom Endpoints

  - Setting CORS

  - Building a Frontend Single-Page Application with COS's Static Website Feature

  - Configuring a Custom CDN Domain Name to Support Gzip Compression

- Image Processing

  - Hybrid Watermarking

## Audio/Video Practices

COS Audio/Video Player Overview

Playing back COS Video File with TCPlayer

Playing back Video in COS with DPlayer

Playing back Video in COS with VideojsPlayer

## Workflow

Using Custom Function to Manage COS Files

## Direct Data Upload

Practice of Direct Transfer for Web End

Practice of Direct Upload Through WeChat Mini Program

Practice of Direct Upload for Mobile Apps

uni-app Direct Upload Practice

## Content Moderation

Blocking CDN Cache Based on Moderation Result

## Data Security

Introduction to COS Data Security Solution

Anti-Fraud Guide

Hotlink Protection Practice

## Data Verification

MD5 Verification

CRC64 Check

## Big Data Practice

Using COS as Deep Storage of Druid

Importing/Exporting COS Using DataX

Configuring COSN for CDH

COS Ranger Permission System Solution

Connecting Oceanus to COS

## Using COS in the Third-party Applications

Use the general configuration of COS in third-party applications compatible with S3

Storing Remote WordPress Attachments to COS

Storing Ghost Attachment to COS

Backing up Files from PC to COS

Using Nextcloud and COS to Build Personal Online File Storage Service

Mounting COS to Windows Server as Local Drive

Setting up Image Hosting Service with PicGo, Typora, and COS

Managing COS Resource with CloudBerry Explorer

# Practical Tutorial

## Overview

Last updated : 2024-01-06 17:50:58

COS offers a variety of best practices for common use cases, such as access control and permission management, performance optimization, data migration, direct data upload and backup, data security, domain name management, big data, and serverless architecture, facilitating your diverse business needs. Specific best practices are as detailed below:

Best Practice	Description
Access control and permission management	Access control and permission management is one of the most practical features of COS. Best practices are outlined in the following documents: <a href="#">ACL Practices</a> <a href="#">CAM Practices</a> <a href="#">Granting Sub-accounts Access to COS</a> <a href="#">Authorization Cases</a> <a href="#">Working with COS API Access Policies</a> <a href="#">Security Guidelines for Using Temporary Credentials for Direct Upload from Frontend to COS</a> <a href="#">Generating and Using Temporary Keys</a> <a href="#">Authorizing Sub-Account to Get Buckets by Tag</a> <a href="#">Descriptions and Use Cases of Condition Keys</a> <a href="#">Granting Sub-account Under One Root Account Permission to Manipulate Buckets Under Another Root Account</a>
Performance optimization	COS supports performance expansion to achieve a higher request rate. For detailed directions, see <a href="#">Request Rate and Performance Optimization</a> .
<a href="#">Accessing COS Using the AWS S3 SDK</a>	COS offers APIs compatible with AWS S3. You can access files in COS using the AWS S3 SDK with simple configurations.
Disaster recovery and backup	Best practices and applicable solutions for disaster recovery and backup are outlined in the following three scenarios. <a href="#">High-Availability Disaster Recovery Architecture Based on Cross-Region Replication</a> <a href="#">Cloud Data Backup</a> <a href="#">Local Data Backup</a>
Domain name management	You can configure an HTTPS custom domain name to access COS. For more information, see <a href="#">Supporting HTTPS for Custom Endpoints</a> . You can configure CORS rules in COS. For more information, see <a href="#">Setting CORS</a> .

	<p>You can host static websites in COS. For more information, see <a href="#">Hosting Static Website</a>.</p> <p>You can build frontend single-page application in COS. For more information, see <a href="#">Building a Frontend Single-Page Application with COS's Static Website Feature</a>.</p>
Direct data upload	<p>Below are the best practices of direct data upload:</p> <ul style="list-style-type: none"> <li><a href="#">Practice of Direct Transfer for Web End</a></li> <li><a href="#">Practice of Direct Upload Through WeChat Mini Program</a></li> <li><a href="#">Practice of Direct Upload for Mobile Apps</a></li> <li><a href="#">uni-app Direct Upload Practice</a></li> </ul>
Data security	<p>This practice describes the data security solution in terms of pre-event prevention, mid-event monitoring, and post-event backtracking. For more information, see <a href="#">Introduction to COS Data Security Solution</a>.</p> <p>You can configure hotlink protection in COS to control access sources. For more information, see <a href="#">Hotlink Protection Practice</a>.</p>
Data verification	<p>You can ensure the integrity of data uploaded to COS by using an MD5 checksum. For more information, see <a href="#">MD5 Verification</a>.</p> <p>You can verify data by using a CRC-64 checksum. For more information, see <a href="#">CRC64 Check</a>.</p>
Big data	<p>You can use COS as the deep storage of Druid. For more information, see <a href="#">Using COS as Deep Storage of Druid</a>.</p> <p>You can use Terraform to manage COS.</p> <p>You can use DataX to import or export data to or from COS. For more information, see <a href="#">Importing/Exporting COS Using DataX</a>.</p> <p>You can configure COSN by using CDH. For more information, see <a href="#">Configuring COSN for CDH</a>.</p> <p>You can use COS Ranger to control permissions. For more information, see <a href="#">COS Ranger Permission System Solution</a>.</p> <p>You can connect Oceanus to COS. For more information, see <a href="#">Connecting Oceanus to COS</a>.</p>
Using COS in third-party applications	<p>You can store data of S3-compatible third-party applications to COS by using COS general settings. For more information, see <a href="#">Use the general configuration of COS in third-party applications compatible with S3</a>.</p> <p>You can use the remote attachment feature to store forum attachments in COS.</p> <p>You can store multimedia content in COS by using the WordPress plugin. For more information, see <a href="#">Storing Remote WordPress Attachments to COS</a>.</p>
<a href="#">Using APIs to zip files</a>	<p>You can package multiple files through an API.</p>

# Access Control and Permission Management

## ACL Practices

Last updated : 2024-03-25 15:11:18

### ACL Overview

Access Control List (ACL) is a resource-based access policy option that manages access to buckets and objects. You can grant read and write permissions to other root accounts, sub-accounts and user groups using ACLs.

**Unlike an access policy, an ACL in Tencent Cloud has the following limits on the permissions:**

Only supports permission grants to Tencent Cloud accounts.

Only supports five operation sets: Read Objects, Write to Objects, Read ACLs, Write to ACLs, and All Permissions.

Does not support additional conditions for the ACL rules to take effect.

Does not support explicit deny.

#### Control granularities supported by ACLs

Bucket

Object Key Prefix

Object

### Control Elements of ACLs

When a bucket or object is created, the root account to which the bucket or object's resources belong has full access to these resources, and the access cannot be modified or deleted. You can use ACLs to grant other Tencent Cloud accounts the access permissions.

The following is an ACL example for a bucket. 10000000001 is the root account, 10000000011 is its sub-account, and 10000000002 is another root account. The ACL contains an Owner element that identifies the bucket owner, who has full access to the bucket. The Grant element grants anonymous read permission, which is expressed as READ permission of `http://cam.qcloud.com/groups/global/AllUsers` .

```
<AccessControlPolicy>
  <Owner>
    <ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
    <DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Root
```

```

    <ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
    <DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
  </Grantee>
  <Permission>FULL_CONTROL</Permission>
</Grant>
<Grant>
  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
    <URI>http://cam.qcloud.com/groups/global/AllUsers</URI>
  </Grantee>
  <Permission>READ</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>

```

## Grantees

### Root account

You can grant access permissions to other root accounts using the definition of the principal in CAM, as described below:

```
qcs::cam::uin/100000000002:uin/100000000002
```

### Sub-account

You can grant access permissions to sub-accounts (such as 100000000011) under your root account or those under other root accounts using the definition of the principal in CAM, as described below:

```
qcs::cam::uin/100000000001:uin/100000000011
```

### Anonymous user

You can grant access permissions to anonymous users using the definition of the principal in CAM, as described below:

```
http://cam.qcloud.com/groups/global/AllUsers
```

## Operation Set

The operation sets supported by ACLs are listed below.

Operation Set	Access to Bucket	Access to Prefix	Access to Object
READ	Lists and reads objects in the bucket	Lists and reads objects in the directory	Reads objects
WRITE	Creates, overwrites and deletes any object in the bucket	Creates, overwrites and deletes any object in the directory	Not supported



READ_ACP	Reads the ACL of bucket	Reads the ACL in the directory	Reads the ACL of object
WRITE_ACP	Modifies the ACL of bucket	Modifies the ACL in the directory	Modifies the ACL of object
FULL_CONTROL	Any operations on the bucket and objects	Any operations on the objects in the directory	Any operations on the objects

## Standard ACL

COS supports a range of predefined authorizations, which are called standard ACLs. The meanings of the standard ACLs are listed below.

### Note:

A root account always has the FULL\_CONTROL permission, which is not described here.

Standard ACL	Description
(Null)	This is the default policy. Others do not have permissions. The permissions for resources are inherited from upper level.
private	Other users do not have permissions
public-read	Anonymous user group has the READ permission
public-read-write	Anonymous user group has the READ and WRITE permissions. This is not recommended for buckets.

## Directions

### Using ACLs in the console

#### Set an ACL for a bucket

The following example grants another root account the read access to a **bucket**:

**Bucket ACL (Access Control List)**

Public Permissions  Private (read-write)  Public read & Private write  Public (read-write)

User ACL

User Type	Account ID ⓘ	Permissions	Actions
Root account	10000	Full control	--
<input type="text" value="Root account"/>	<input type="text" value="100000000"/>	<input checked="" type="checkbox"/> Read <input type="checkbox"/> Write <input type="checkbox"/> Read ACL ⓘ <input type="checkbox"/> Write ACL ⓘ <input type="checkbox"/> Full control	<a href="#">Save</a> <a href="#">Del</a>
<a href="#">Add User</a>			

### Set an ACL for an object

The following example grants another root account the read access to an **object**:

**Object ACL (Access Control List)**

Public Permissions  Inherit  Private (read-write)  Public read & Private write

User ACL

User Type	Account ID	Permissions	Actions
Root account	10000	Full control	--
<input type="text" value="Root account"/>	<input type="text" value="100000001"/>	<input checked="" type="checkbox"/> Read <input type="checkbox"/> Read ACL ⓘ <input type="checkbox"/> Write ACL ⓘ <input type="checkbox"/> Full control	<a href="#">Save</a> <a href="#">Delete</a>
<a href="#">Add User</a>			

**Note:**

If the message **You have no access to it** appears when you access a bucket or object using a sub-account, grant the sub-account the access to the bucket through the root account. For more information, see [Accessing Bucket List Using Sub-Account](#).

### Using ACLs via APIs

**Bucket ACL**

API	Operation	Description
<a href="#">PUT Bucket acl</a>	Setting bucket ACL	Sets the ACL for a specified bucket
<a href="#">GET Bucket acl</a>	Querying bucket ACL	Queries the ACL of a bucket

**Object ACL**

API	Operation	Description
<a href="#">PUT Object acl</a>	Setting object ACL	Sets the ACL for a specified object in a bucket
<a href="#">GET Object acl</a>	Querying object ACL	Queries the ACL of an object

# CAM Practices

Last updated : 2024-03-25 15:11:18

## Overview

Cloud Access Management (CAM) is a Tencent Cloud authentication and authorization service that helps you manage the access to your Tencent Cloud resources. You can manage authorized objects, resources, and operations and set policies to control access when granting permissions.

## Features

### Granting access to resources under the root account

You can grant the access to resources under your root account to other users, including sub-accounts and other root accounts, without sharing the identity credentials of your root account.

### Granular permission management

Different access permissions of different resources can be granted to different users. For example, some sub-accounts can be granted the read access to a COS bucket, while some other sub-accounts or root accounts can be granted the write access to a COS object. Such resources, access permissions, and users can all be managed in batch.

### Eventual consistency

CAM currently supports data sync across Tencent Cloud regions by copying policies. CAM policies can be modified timely, but it may take some time for those policies synced across regions to take effect. In addition, CAM uses cache (currently valid for one minute) to improve the performance, and any policy update does not take effect until the cache expires.

## Use Cases

### Enterprise sub-account permission management

Enterprises may need to grant the minimal access permission of their cloud resources to all kinds of employees.

Assume that an enterprise owns many cloud resources (CVM/VPC/CDN instances, COS buckets/objects, etc.) and many employees (developers, testers, Ops engineers, etc.).

Developers and testers need the read/write permissions of project-specific cloud resources on development servers and test servers respectively, while Ops engineers are responsible for the purchase and daily operations of such servers. If the duty or project of an employee changes, the corresponding permissions should be terminated.

## Cross-enterprise permission management

There are cases where enterprises may need to share their cloud resources. For example, enterprise A has many cloud resources and wants to focus on product R&D, so it outsources the operations of its cloud resources to enterprise B, and it needs to revoke all the granted permissions as soon as the outsourcing service contract is terminated.

## Policy Syntax

A CAM policy consists of several elements and is used to describe specific information on authorization. Core elements include principal, action, resource, condition, and effect. For more information, see [Overview](#).

### Note:

There is no particular sequence in the description of policy syntax. However, note that the action element is case-sensitive.

If there are no particular conditions required, the condition element is optional.

You can define the principal element only through policy management APIs or policy syntax parameters but not in the console.

### Core elements

Core Element	Description	Required
version	Specifies the version of the policy syntax. Valid value: <code>2.0</code> .	Yes
principal	Describes the entity to be authorized by the policy, including users (developers, sub-accounts, anonymous users) and user groups	This element can be used only in policy management APIs and policy syntax parameters.
statement	Describes the details of a permission or a permission set defined by other elements including effect, action, resource, and condition. One policy has only one statement.	Yes
action	Describes the action to be allowed or denied, which can be an API operation or a set of API operations. This element is case-sensitive, such as <code>name/cos:GetService</code> .	Yes
resource	Describes the resource to which the permission applies. A resource is described in a six-segment format. Detailed resource definitions vary by product. For more information on how to specify a resource, see the documentation of the product for which you are writing a resource statement.	Yes

condition	Describes the condition for the policy to take effect. A condition consists of the operator, action key, and action value. A condition value may be time, IP address, etc. Some services allow you to specify additional values in a condition.	No
effect	Describes whether the statement result is "allow" or "deny".	Yes

### Policy limits

Item	Upper Limit
Number of user groups under a root account	300
Number of sub-accounts under a root account	1,000
Number of roles under a root account	1,000
Number of user groups that a sub-account can be added to	10
Number of root accounts with which a collaborator can be associated	10
Number of sub-accounts in a user group	100
Number of custom policies that can be created under a root account	1,500
Number of policies that can be directly associated with a user, user group, or role	200
Number of characters in a policy syntax	4,096

### Sample policy

The policy in this example allows the sub-account with ID 100000000011 under the root account with ID 100000000001 (APPID: 1250000000) to **upload** and **download** objects regarding the `examplebucket-bj` bucket in Beijing region and the `exampleobject` object in the `examplebucket-gz` bucket in Guangzhou region, on condition that the access IP falls within the IP range `10.*.*.10/24`.

```
{
  "version": "2.0",
  "principal": {
    "qcs": ["qcs::cam::uin/100000000001:uin/100000000011"]
  },
  "statement": [{
    "effect": "allow",
    "action": ["name/cos:PutObject", "name/cos:GetObject"],
    "resource": ["qcs::cos:ap-beijing:uid/1250000000:examplebucket-bj-1",
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-gz-12500000"],
  }],
}
```

```
        "condition": {
            "ip_equal": {
                "qcs:ip": "10.*.*.10/24"
            }
        }
    }
}
```

# Granting Sub-Accounts Access to COS

Last updated : 2024-03-25 15:11:17

## Overview

You can set different operation permissions on COS buckets or objects through CAM, so that different teams or users in different companies or departments can better collaborate with each other .

To start with, you need to understand several terms: root account, sub-account (user), and user group. For CAM terms and the detailed description of configurations, please see CAM [Glossary](#).

### Root account

A root account is also known as a developer. When you sign up for a Tencent Cloud account, the system creates a root account identity for you to log in to the Tencent Cloud services. Tencent Cloud records your usage and bills you based on the root account.

By default, a root account has full access to the resources in the account. A root account can access billing information, change user passwords, create users and user groups, access other Tencent Cloud service resources, etc. By default, only a root account can access such resources. Any other users can only access them after they are authorized by a root account.

### Sub-account (user) and user group

A sub-account is an entity created by the root account. It has an ID and identity credentials, as well as permission to log in to the Tencent Cloud console.

By default, a sub-account does not own resources. It needs to be authorized by a root account.

One root account can create multiple sub-accounts (users).

One sub-account can belong to multiple root accounts to assist them in managing their Tencent Cloud resources.

However, at any specific time point, one sub-account can only log in under one root account to manage its Tencent Cloud resources.

A sub-account can switch between developers (root accounts) in the console.

When a sub-account logs in to the console, it is under its default root account and has the access permissions granted by the root account.

After a sub-account switches from one root account to another, it will only have the access permissions granted by the current root account, but not the permissions granted by the previous one.

A user group is a collection of multiple users (sub-accounts) with the same functions. You can create different user groups based on your business needs and associate them with appropriate policies to grant them different permissions.



## Directions

You can grant a sub-account permission to access COS in three steps: creating a sub-account, granting permissions to the sub-account, and accessing COS resources with the sub-account.

### Step 1. Create a sub-account

You can create a sub-account in the CAM console and grant it access permissions. The specific operations are shown as below:

1. Log in to the [CAM console](#).
2. Select **User > User List > Create User** to enter the user creation page.
3. Select **Custom Creation**, set **Access Resources and Receive Messages**, and click **Next**.
4. Enter the user information as required.

**User Information:** Set the username (for example, Sub\_user) and email address of the sub-account. The email address is needed to receive the email sent by Tencent Cloud to bind the sub-account with his/her Weixin account.

**Access Method:** Select **Programming access** and **Tencent Cloud console access**. Other configuration items can be set as needed.

5. Click **Next** and start identity verification.
6. Grant permissions to the sub-account. You can configure simple policies, such as granting the sub-account permission to access the COS bucket list or read-only permission, with the policy options provided. To configure a more complex policy, proceed to [Step 2. Grant permissions to the sub-account](#).
7. Set the user tag optionally and click **Next**.
8. Click **Finish**. In this way, the sub-account is created.

### Step 2. Grant permissions to the sub-account

Create a custom policy or select an existing policy and associate it with the sub-account.

1. Log in to the [CAM console](#).
2. Choose **Policy > Create Custom Policy > Create by Policy Syntax** to go to the policy creation page.
3. You can select **Blank Template** to customize a permission policy as needed. You can also select a COS-associated **System Template**. Then, click **Next**.
4. Enter an easy-to-remember policy name. If you have selected **Blank Template**, enter the policy syntax. For more information, see [\[Sample Policies\]\(#Sample policies\)](#). You can copy and paste the policy content into the **Policy Content** box and click **Finish**.
5. After you create a policy, associate it with the sub-account.

**Policy** Custom Policy ▾

Bind users or user groups with the policy to assign them related permissions.

Create Custom Policy Delete Support search by policy

Policy Name	Description	Service Type ▾	Operation
policygen-20190627105115	-	-	Delete   Bind User/

6. Select the sub-account and click **Confirm** to complete the authorization.

**Bind User/User Group**

**Bind User**

Support multi-keywords search by user name/ID/SecretID In Q

User	Switch to User ... ▾
<input checked="" type="checkbox"/> Jason	User

(1) selected

User Name/Group Name	Type
Jason	User

Press Shift to select multiple items

OK Cancel

### Step 3. Access COS resources using the sub-account

The access methods (programming access and Tencent Cloud console access) mentioned in Step 1 are described as follows:

(1) Programming access

To use the programming access method (for example, using APIs, SDKs, or tools) to access COS resources with a sub-account, you need to obtain the `APPID` of the root account first. Besides, you need to go to the CAM console to generate `SecretId` and `SecretKey` of the sub-account as follows:

1. Log in to the [CAM console](#) with the root account.
  2. Click **User > User List**.
  3. Click the name of the sub-account to go to the **User Details** page of the sub-account.
  4. Click the **API Key** tab. Then, click **Create Key** to create `SecretId` and `SecretKey` for the sub-account.
- After this, you can use this sub-account's `SecretId` and `SecretKey`, as well as the root account's `APPID` to access COS resources.

**Note:**

To access COS resources with a sub-account, you need to use XML APIs or SDKs based on XML APIs.

**Example of access using XML Java SDK**

The following parameters need to be set if you use the XML Java SDK:

```
// Initialize the user authentication information
COSCredentials cred = new BasicCOSCredentials("<root account's APPID>", "<sub-account's SecretId>", "<sub-account's SecretKey>");
```

**Example:**

```
String secretId = System.getenv("secretId");// Sub-account's `SecretId`. Follow the principle of least privilege to reduce risks. For information about how to obtain a sub-account key, visit https://cloud.tencent.com/document/product/598/37140.
String secretKey = System.getenv("secretKey");// Sub-account's `SecretKey`. Follow the principle of least privilege to reduce risks. For information about how to obtain a sub-account key, visit https://cloud.tencent.com/document/product/598/37140.
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// Initialize the user authentication information
COSCredentials cred = new BasicCOSCredentials("<root account's APPID>", secretId, secretKey);
```

**Example of access using COSCMD command line tool**

The following parameters need to be set if you use COSCMD:

```
coscmd config -u <root account's APPID> -a <sub-account's SecretId> -s <sub-account's SecretKey>
```

**Example:**

```
coscmd config -u 1250000000 -a AKIDasdfmRxHPa9oLhJp**** -s e8Sdeasdfas2238Vi**** -b
```

## (2) Tencent Cloud console access

After the sub-user is granted permissions, they can enter the root account ID, sub-user name, and sub-user password on the [Sub-user Login](#) page to log in to the console. Then, they can click **Cloud Object Storage** in **Products** to access storage resources under the root account.

## Sample Policies

The following typical sample policies are provided herein. When configuring a policy, you can refer to the following code, copy and paste it into the **Policy Content** box, and modify it as needed. For more policy syntax for other common COS scenarios, see [Overview](#) or the business use cases parts of [Cloud Access Management](#).

### Sample 1. Granting the sub-account full read/write permissions for COS access

#### Note:

This policy grants a large range of permissions to the sub-account. Please configure it with caution.

The policy is as follows:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:*"
      ],
      "resource": "*",
      "effect": "allow"
    },
    {
      "effect": "allow",
      "action": "monitor:*",
      "resource": "*"
    }
  ]
}
```

### Sample 2. Granting the sub-account read-only permission

The following policy grants the sub-account read-only permission:

```
{
  "version": "2.0",
  "statement": [
```

```
{
  "action": [
    "name/cos:List*",
    "name/cos:Get*",
    "name/cos:Head*",
    "name/cos:OptionsObject"
  ],
  "resource": "*",
  "effect": "allow"
},
{
  "effect": "allow",
  "action": "monitor:*",
  "resource": "*"
}
]
```

### Sample 3. Granting the sub-account write-only permission (excluding deletion)

The policy is as follows:

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "cos:ListParts",
        "cos:PostObject",
        "cos:PutObject*",
        "cos:InitiateMultipartUpload",
        "cos:UploadPart",
        "cos:UploadPartCopy",
        "cos:CompleteMultipartUpload",
        "cos:AbortMultipartUpload",
        "cos:ListMultipartUploads"
      ],
      "resource": "*"
    }
  ]
}
```

### Sample 4. Granting the sub-account read/write permission for a certain IP range

The following sample grants read/write permission only for the `192.168.1.0/24` and `192.168.2.0/24` IP address ranges:

To enter more conditions, see [Conditions](#).

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "cos:*"
      ],
      "resource": "*",
      "effect": "allow",
      "condition": {
        "ip_equal": {
          "qcs:ip": ["192.168.1.0/24", "192.168.2.0/24"]
        }
      }
    }
  ]
}
```

# Authorization Cases

Last updated : 2024-03-25 15:11:17

## Granting Permission via Bucket Policy

### Preparations

1. Create a bucket.

Granting permissions through a bucket policy involves specific buckets, so you need to [create a bucket](#) first. For authorization at the account level, see [Granting Permission via CAM](#) in this document.

2. Prepare the UIN of the account to be authorized.

This document assumes that the root account that owns the target bucket has a UIN of 100000000001 and its sub-account has a UIN of 100000000011. The sub-account needs to be authorized to access the destination bucket.

#### Note:

To query sub-accounts created under the root account, log in to the CAM console and view them in the [User List](#).

To create a sub-account, see [Creating Sub-user](#).

3. Open the **Add Policy** dialog.

Click the destination bucket and select **Permission Management** > **Permission Policy Settings** > **Visual Editor** > **Add Policy**. Then, you can configure a policy as instructed in the authorization cases below. For detailed directions, see [Adding a Bucket Policy](#).

The following lists several authorization cases, which you can configure as needed.

### Authorization cases

#### Case 1: Granting a sub-account full permissions for a specified directory

The configuration information is as follows:

Configuration Item	Description
Effect	Select <b>Allow</b> .
User	Select <b>Sub-account</b> and enter a sub-account UIN, which must be a sub-account under the current root account, such as <code>1000000000011</code> .
Resource	Select a specific resource path, such as <code>folder/sub-folder/*</code> .
Actions	Select <b>All Actions</b> .

#### Case 2: Granting a sub-account read permission for all files in a specified directory

The configuration information is as follows:

Configuration Item	Description
Effect	Select <b>Allow</b> .
User	Select <b>Sub-account</b> and enter a sub-account UIN, which must be a sub-account under the current root account, such as <code>100000000011</code> .
Resource	Select a specific resource path, such as <code>folder/sub-folder/*</code> .
Actions	Select <b>Read Operation (listing objects is included)</b> .

### Case 3: Granting a sub-account read/write permission for specified files

The configuration information is as follows:

Configuration Item	Description
Effect	Select <b>Allow</b> .
User	Select <b>Sub-account</b> and enter a sub-account UIN, which must be a sub-account under the current root account, such as <code>100000000011</code> .
Resource	Select a specific object key, such as <code>folder/sub-folder/example.jpg</code> .
Actions	Select <b>All Actions</b> .

### Case 4: Granting a sub-account read/write permission for all files in a specified directory while denying read/write permission for specified files in the directory

For this case, you need to add an **allow** policy and a **deny** policy.

1. First, add the **allow** policy. The configuration information is as follows:

Configuration Item	Description
Effect	Select <b>Allow</b> .
User	Select <b>Sub-account</b> and enter a sub-account UIN, which must be a sub-account under the current root account, such as <code>100000000011</code> .
Resource	Specify a directory prefix, such as <code>folder/sub-folder/*</code> .
Actions	Select <b>All Actions</b> .

2. Then, add the **deny** policy. The configuration information is as follows:

Configuration Item	Description



Effect	Select <b>Deny</b> .
User	Select <b>Sub-account</b> and enter a sub-account UIN, which must be a sub-account under the current root account, such as <code>1000000000011</code> .
Resource	Specify the object key to be denied access, such as <code>folder/sub-folder/privateobject</code> .
Actions	Select <b>All Actions</b> .

### Case 5: Granting a sub-account read/write permission for files with a specified prefix

The configuration information is as follows:

Configuration Item	Description
Effect	Select <b>Allow</b> .
User	Select <b>Sub-account</b> and enter a sub-account UIN, which must be a sub-account under the current root account, such as <code>1000000000011</code> .
Resource	Specify a prefix, such as <code>folder/sub-folder/prefix</code> .
Actions	Select <b>All Actions</b> .

## Granting Permission via CAM

If you need to grant permissions at the account level, see the following documents:

[Authorizing Sub-account Full Access to Specific Directory](#)

[Authorizing Sub-account Read-only Access to Files in Specific Directory](#)

[Authorizing Sub-account Read/Write Access to Specific File](#)

[Authorizing Sub-account Read-only Access to COS Resources](#)

[Authorizing a Sub-account Read/Write Access to All Files in Specified Directory Except Specified Files](#)

[Authorizing Sub-account Read/Write Access to Files with Specified Prefix](#)

[Authorizing Another Account Read/Write Access to Specific Files](#)

# Working with COS API Authorization Policies

Last updated : 2024-03-25 15:11:17

## Note:

When granting API access permissions to a sub-user or collaborator, be sure to follow the principle of least privilege and grant the minimum set of permissions necessary to satisfy business needs. There may be data security risks if you grant excessive access to all of your resources (`resource:*`) or all operations (`action:*`).

## Overview

When using a temporary key to access COS, the operation permissions required vary by API or series of APIs that you specify.

A COS API authorization policy is a JSON string. For example, below is a policy that grants the permission to perform uploads (including simple upload, upload through an HTML form, and multipart upload) for objects prefixed with `doc` and downloads for objects prefixed with `doc2` for the bucket `examplebucket-1250000000` in the region "ap-beijing" under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [{
    "action": [
      // Upload an object by using simple upload
      "name/cos:PutObject",
      // Upload an object by using an HTML form
      "name/cos:PostObject",
      // Initialize a multipart upload
      "name/cos:InitiateMultipartUpload",
      // List all ongoing multipart uploads
      "name/cos:ListMultipartUploads",
      // List uploaded parts
      "name/cos:ListParts",
      // Upload parts
      "name/cos:UploadPart",
      // Complete a multipart upload
      "name/cos:CompleteMultipartUpload",
      // Abort a multipart upload
      "name/cos:AbortMultipartUpload"
    ],
    "effect": "allow",
    "resource": [
      "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
```

```

    ]
  },
  {
    "action": [
      // Download
      "name/cos:GetObject"
    ],
    "effect": "allow",
    "resource": [
      "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
    ]
  }
]
}

```

## Authorization Policy Elements

Name	Description
version	Policy syntax version, which is 2.0 by default.
effect	Allow or deny.
resource	Specific data of the authorized operation, which can be any resources, resources with a specified path prefix, resource in a specified absolute path, or their combination.
action	COS API. You can specify one, several, or all ( * ) COS APIs as needed, such as <code>name/cos:GetService</code> . <b>Note that this value is case-sensitive.</b>
condition	Optional condition. For more information, see <a href="#">Element Reference</a> .

Examples of authorization policy settings for each COS API are as listed below.

## Service APIs

### Querying bucket list

To grant access to the `GET Service` API, the `action` field in the policy should be set to `name/cos:GetService` , and the `resource` field to `*` .

### Sample

The following policy grants the permission to query the bucket list:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetService"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
    }
  ]
}
```

## Bucket APIs

The `resource` field for bucket API policies is outlined in further detail below:

To allow access to buckets in all regions

The `resource` field should be set to `*`. **Use this option with caution as it may present data security risks due to excessive permissions.**

To allow access only to buckets in a specified region

For example, to grant access to `examplebucket-1250000000` under the APPID `1250000000` in the region `ap-beijing`, the `resource` field should be set to `qcs::cos:ap-beijing:uid/1250000000:*`.

To allow access only to a bucket with a specified name in a specified region

For example, to grant access to the bucket named `examplebucket-1250000000` under the APPID `1250000000` in the region `ap-beijing`, the `resource` field should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*`.

The `action` field in bucket API policies varies by operation. The following lists several bucket API policies for your reference.

### Creating bucket

To grant access to the `PUT Bucket` API, the `action` field in the policy should be set to `name/cos:PutBucket`.

### Sample

The following policy grants the user with the APPID `1250000000` permission to create a bucket named `examplebucket-1250000000` in Beijing region:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

**Note:**

For bucket naming rules, see [Bucket Overview](#).

## Extracting bucket and its permissions

To grant the access to the `HEAD Bucket` API, the `action` field in the policy should be set to `name/cos:HeadBucket`.

### Sample

The following policy grants the permission to extract only the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:HeadBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

## Querying object list

To grant access to the `GET Bucket` API, the `action` field in the policy should be set to `name/cos:GetBucket` .

### Sample

The following policy grants the permission to query only the list of objects in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

### Deleting bucket

To grant access to the `Delete Bucket` API, the `action` field in the policy should be set to `name/cos>DeleteBucket` .

### Sample

The following policy grants the permission to delete only the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos>DeleteBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

```
}
```

## Setting bucket ACL

To grant access to the `Put Bucket ACL` API, the `action` field in the policy should be set to

```
name/cos:PutBucketACL .
```

### Sample

The following policy grants the permission to set an ACL only for the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

## Querying bucket ACL

To grant access to the `GET Bucket acl` API, the `action` field in the policy should be set to

```
name/cos:GetBucketACL .
```

### Sample

The following policy grants the permission to get the ACL only of the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketACL"
      ],
      "effect": "allow",
      "resource": [
```

```
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
    ]
}
]
```

## Setting CORS configuration

To grant access to the `PUT Bucket cors` API, the `action` field in the policy should be set to `name/cos:PutBucketCORS`.

### Sample

The following policy grants the permission to set a CORS configuration only for the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

## Querying CORS configuration

To grant access to the `GET Bucket cors` API, the `action` field in the policy should be set to `name/cos:GetBucketCORS`.

### Sample

The following policy grants the permission to query the CORS configuration only of the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
```



```
    "name/cos:GetBucketCORS"
  ],
  "effect": "allow",
  "resource": [
    "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
  ]
}
]
```

## Deleting CORS configuration

To grant access to the `DELETE Bucket cors` API, the `action` field in the policy should be set to `name/cos:DeleteBucketCORS`.

### Sample

The following policy grants the permission to delete the CORS configuration only of the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

## Setting lifecycle configuration

To grant access to the `PUT Bucket lifecycle` API, the `action` field in the policy should be set to `name/cos:PutBucketLifecycle`.

### Sample

The following policy grants the permission to set a lifecycle configuration only for the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
```

```
"version": "2.0",
"statement": [
  {
    "action": [
      "name/cos:PutBucketLifecycle"
    ],
    "effect": "allow",
    "resource": [
      "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
    ]
  }
]
```

## Querying lifecycle configuration

To grant access to the `GET Bucket lifecycle` API, the `action` field in the policy should be set to `name/cos:GetBucketLifecycle`.

### Sample

The following policy grants the permission to query the lifecycle configuration only of the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

## Deleting lifecycle configuration

To grant access to the `DELETE Bucket lifecycle` API, the `action` field in the policy should be set to `name/cos>DeleteBucketLifecycle`.

### Sample

The following policy grants the permission to delete the lifecycle configuration only of the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

## Object APIs

The `resource` field for object API policies is outlined in further detail below:

To grant access to all objects, the `resource` field should be set to `*` .

To grant access only to objects in a specified bucket, such as objects in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` , the `resource` field should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*` .

To grant access only to objects with a specified path prefix in a specified bucket, such as objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` , the `resource` field should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*` .

To grant access only to an object in a specified absolute path, such as the object in the absolute path `doc/audio.mp3` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` , the `resource` field should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/audio.mp3` .

The `action` field in object API policies varies by operation. All object API policies are as listed below.

### Uploading object by using simple upload

To grant access to the `PUT Object` API, the `action` field in the policy should be set to `name/cos:PutObject` .

## Sample

The following policy grants the permission to use simple upload to upload only objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Multipart upload

Multipart upload APIs include `Initiate Multipart Upload` , `List Multipart Uploads` , `List Parts` , `Upload Part` , `Complete Multipart Upload` , and `Abort Multipart Upload` . To grant access to these APIs, the `action` field in the policy should be a collection of

```
"name/cos:InitiateMultipartUpload", "name/cos:ListMultipartUploads", "name/cos:ListParts", "name/cos:UploadPart", "name/cos:CompleteMultipartUpload", "name/cos:AbortMultipartUpload" .
```

## Sample

The following policy grants the permission to use multipart upload to upload only objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:InitiateMultipartUpload",
        "name/cos:ListMultipartUploads",
        "name/cos:ListParts",
        "name/cos:UploadPart",
        "name/cos:CompleteMultipartUpload",
        "name/cos:AbortMultipartUpload"
      ],

```

```
    "effect": "allow",
    "resource": [
      "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
    ]
  }
]
```

## Querying multipart upload

To grant access to this API, the `action` field in the policy should be set to `name/cos:ListMultipartUploads`.

### Sample

The following policy grants the permission to query ongoing multipart uploads only in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:ListMultipartUploads"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
    }
  ]
}
```

## Uploading object by using HTML form

To grant access to the `POST Object` API, the `action` field in the policy should be set to `name/cos:PostObject`.

### Sample

The following policy grants the permission to use the `POST` method to upload only objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
```

```
"statement": [
  {
    "action": [
      "name/cos:PostObject"
    ],
    "effect": "allow",
    "resource": [
      "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
    ]
  }
]
```

## Appending parts

To grant access to the `Append Object` API, the `action` field in the policy should be set to `name/cos:AppendObject`.

## Sample

The following policy grants permission to append parts to objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:AppendObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Querying object metadata

To grant access to the `HEAD Object` API, the `action` field in the policy should be set to `name/cos:HeadObject`.

## Sample

The following policy grants the permission to query objects only with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:HeadObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Downloading object

To grant access to the `GET Object` API, the `action` field in the policy should be set to `name/cos:GetObject` .

## Sample

The following policy grants the permission to download only objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Copying object

To grant access to the `Put Object Copy` API, the `action` field for the destination object should be set to `name/cos:PutObject`, and the `action` field for the source object should be set to `name/cos:GetObject`.

## Sample

The following policy grants the permission to use multipart copy to copy objects from the path prefixed with `doc` to the path prefixed with `doc2` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    },
    {
      "action": [
        "name/cos:GetObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
      ]
    }
  ]
}
```

Here, `"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"` is the source object.

## Copying part

To grant access to the `Upload Part - Copy` API, the `action` field for the destination object should be a collection of

`"name/cos:InitiateMultipartUpload", "name/cos:ListMultipartUploads", "name/cos:ListParts", "name/cos:PutObject", "name/cos:CompleteMultipartUpload", "name/cos:AbortMultipartUpload"`, and the `action` field for the source object should be set to `name/cos:GetObject`.



## Sample

The following policy grants the permission to use multipart copy to copy objects from the path prefixed with `doc` to the path prefixed with `doc2` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:InitiateMultipartUpload",
        "name/cos:ListMultipartUploads",
        "name/cos:ListParts",
        "name/cos:PutObject",
        "name/cos:CompleteMultipartUpload",
        "name/cos:AbortMultipartUpload"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    },
    {
      "action": [
        "name/cos:GetObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
      ]
    }
  ]
}
```

Here, `"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"` is the source object.

## Setting object ACL

To grant access to the `Put Object ACL` API, the `action` field in the policy should be set to `name/cos:PutObjectACL` .

## Sample

The following policy grants the permission to set an ACL only for objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutObjectACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Querying object ACL

To grant access to the `Get Object ACL` API, the `action` field in the policy should be set to `name/cos:GetObjectACL`.

## Sample

The following policy grants the permission to query the ACL only of objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetObjectACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Checking CORS configuration

To grant access to the `OPTIONS Object` API, the `action` field in the policy should be set to `name/cos:OptionsObject`.

## Sample

The following policy grants the permission to send an `OPTIONS` request only for objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:OptionsObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Restoring archived object

To grant access to the `Post Object Restore` API, the `action` field in the policy should be set to `name/cos:PostObjectRestore` .

## Sample

The following policy grants the permission to restore archived objects only with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000` :

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PostObjectRestore"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

## Deleting object

To grant access to the `DELETE Object` API, the `action` field in the policy should be set to `name/cos:DeleteObject`.

### Sample

The following policy grants the permission to delete only the object `audio.mp3` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3"
      ]
    }
  ]
}
```

### Deleting multiple objects

To grant access to the `DELETE Multiple Objects` API, the `action` field in the policy should be set to `name/cos:DeleteObject`.

### Sample

The following policy grants the permission to batch delete only the objects `audio.mp3` and `video.mp4` in the bucket `examplebucket-1250000000` in the region `ap-beijing` under the APPID `1250000000`:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3",
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/video.mp4"
      ]
    }
  ]
}
```

```
]
}
```

## Authorization Policies for Common Scenarios

### Granting full access to all resources

The following policy grants full access to all resources:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "*"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
    }
  ]
}
```

### Granting read-only access to all resources

The following policy grants read-only access to all resources:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:HeadObject",
        "name/cos:GetObject",
        "name/cos:GetBucket",
        "name/cos:OptionsObject"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
    }
  ]
}
```

## Granting read-write access to resources with specified path prefix

The following policy grants the permission to access only files with the path prefix `doc` in the bucket

`examplebucket-1250000000` and does not allow any operations on files in other paths:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "*"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-shanghai:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
    }
  ]
}
```

# Security Guidelines for Using Temporary Credentials for Direct Upload from Frontend to COS

Last updated : 2024-11-20 15:48:24

## Overview

In mobile and web applications, you can directly initiate requests to COS on the frontend through the SDK for iOS, Android, or JavaScript. In this way, data upload and download do not pass through your backend server, which reduces the bandwidth usage and load of your backend server and makes full use of various capabilities of COS, such as bandwidth and global acceleration, to improve the user experience of your application.

In actual usage, you need to use a temporary key as the signature for frontend COS requests to avoid problems such as leakage of the permanent key and unauthorized access. However, even with a temporary signature, if you specify excessive permissions or paths when generating it, such problems may still occur, which brings certain risks to your application. This document describes some bad examples and security regulations that you need to comply with for your application to securely use COS.

## Prerequisites

This document assumes that you have a good understanding of the concepts related to temporary key and can generate and use a temporary key to send requests to COS. For more information on how to generate and use a temporary key, see [Generating and Using Temporary Keys](#).

### Note:

When authorizing access with a temporary key, ensure that you follow the principle of the least privilege as needed. If you grant excessive permissions, such as granting permissions to all resources ( `resource: *` ) or all operations ( `action: *` ), data security risks may arise.

## Bad Examples and Security Regulations

### Bad example 1. Excessive `resource`

Application A uses COS when registered users upload profile photos. The profile photo of each user has a fixed object key `app/avatar/<Username>.jpg` and object keys `app/avatar/<Username>_m.jpg` and

`app/avatar/<Username>_s.jpg` for different photo sizes. When you generate a temporary key on the backend, you specify `resource` as `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/*` for convenience. In this case, when a malicious user gets the generated temporary key through methods such as packet capture, they can upload an image to overwrite any user's profile photo and thus gain unauthorized access, and the user's valid profile photo will be overwritten and lost.

### Security regulation

`resource` indicates the resource path that a temporary key can access, and end users covered by this path need to be taken into full account. In principle, resources specified by `resource` should be used only by a single user. In this example, the specified `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/*` will apparently cover all users, resulting in security vulnerabilities.

In this example, the path to user's profile photo can be modified to `app/avatar/<Username>/<size>.jpg`, and `resource` can be specified as `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>/*` then to meet the security regulations. In addition, multiple values can be passed in to the `resource` field as an array. Therefore, you can explicitly specify multiple `resource` values to fully limit the final resource paths that users can access; for example:

```
"resource": [
  "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>.jpg",
  "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>_m.jpg",
  "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>_s.jpg"
]
```

### Bad example 2. Excessive `action`

Application B provides a public photo wall feature, where all photos are stored in `app/photos/*`, and the client needs to perform `GET Bucket` and `GET Object` operations (i.e., `action`). When you generate a temporary key on the backend, you specify `action` as `name/cos:*` for convenience. In this case, a malicious user can get the generated temporary key through methods such as packet capture to perform all object operations (such as upload and deletion) on any object under the resource path and thus gain unauthorized access, which causes data loss and affects your online business.

### Security regulation

`action` indicates operations allowed for the temporary key. In principle, a temporary key with `name/cos:*` that allows all operations should not be distributed to the frontend; instead, all needed operations must be explicitly listed. If each operation needs different resource paths, you should match the `**operation**` and `resource` path separately rather than specifying them in batches.

In this example, you should use `"action": [ "name/cos:GetBucket", "name/cos:GetObject" ]` to specify operations. For detailed directions on authorization, see [Working with COS API Access Policies](#).



### Bad example 3. Excessive `action` and `resource`

Application C provides a management tool that allows a user to list and download all users' files ( `app/files/*` ) but only upload and delete files in their personal directory ( `app/files/<Username>/*` ). When you generate a temporary key on the backend, you mix four operations (i.e., `action` ) in two permissions as well as the resource paths corresponding to the two permissions. In this case, the temporary key will have the greater permissions specified in the resource paths, that is, the user can list, download, upload, and delete all users' files. Through this vulnerability, a malicious user can tamper with or delete other users' files and thus gain unauthorized access, which exposes valid user data to risks.

### Security regulation

For a combination of multiple `action` and `resource` values, you should not simply mix them in pair; instead, you should use multiple statements to match an `action` with the corresponding `resource` to avoid granting excessive permissions.

In this example, you should use the following code:

```
"statement": [
  {
    "effect": "allow",
    "action": [
      "name/cos:GetBucket",
      "name/cos:GetObject"
    ],
    "resource": "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/files/*"
  },
  {
    "effect": "allow",
    "action": [
      "name/cos:PutObject",
      "name/cos>DeleteObject"
    ],
    "resource": "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/files/<Us
  ]
```

### Bad example 4. Unauthorized access to temporary key

Application D provides a forum service, and attachments to posts in the forum are stored in COS. The forum application has different user levels, and only active users with a certain number of posts can view posts and attachments in certain subforums. For some public subforums, all users can view the posts and attachments. The temporary key generation API on the COS backend will generate a temporary key that allows downloading attachments in the corresponding subforum based on the subforum ID passed in by the frontend. However, in the implementation, the backend does not check whether the requesting user has access to the subforum of the specified

ID; therefore, anyone can request this API to get a temporary key that allows access to attachments in private subforums and thus gain unauthorized access. The limitation on access to private resources does not take effect, leading to potential data leakage.

### Security regulation

In addition to making sure that the permissions of the obtained temporary key are granted as expected, the API for getting the temporary key also needs to check whether the correct permissions are requested by correct users based on the actual business scenario, so as to prevent users with low permissions from getting the temporary key for high permissions.

In this example, the backend API should also check whether the requesting user has access to the specified subforum; and if not, it should not return a temporary key.

## Summary

The examples above illustrate the security risks that may occur if permissions of a temporary key are more excessive than expected. In the scenario where files can be directly uploaded to COS on the frontend, as a malicious user can get a temporary key more easily than in the scenario where COS is accessed on the backend, you should pay more attention to permission control.

The security regulations mentioned in this document are based on the principle of least privilege. In actual usage, you can enumerate all possible permissions based on `action` and `resources`. For example, three `action` values and two `resource` values can form  $3 * 2 = 6$  accessible resources and corresponding operations. You can evaluate whether the permissions are granted as expected based on this enumeration; and if not, you should consider enumerating multiple statements to separate permissions.

In addition, you should take authentication into full account for the temporary key generation API. Only when the operation of getting the temporary key is secure can the obtained temporary key be truly secure. There must be no omissions on the security chain.

# Generating and Using Temporary Keys

Last updated : 2024-03-25 15:11:17

## Note:

When authorizing access with a temporary key, ensure that you follow the principle of the least privilege as needed. If you grant excessive permissions, such as granting permissions to all resources (`resource: *`) or all operations (`action: *`), data security risks may arise.

If you have specified a permission scope for the temporary key when applying for it, you can only use the key within the scope. For example, if you have limited the permissions to only uploading objects to the `examplebucket-1-1250000000`, you **can't** upload objects to the `examplebucket-2-1250000000` bucket or download objects from the `examplebucket-1-1250000000` bucket.

## Temporary Key

[Temporary keys \(temporary access credentials\)](#) are access-limited keys requested through CAM APIs.

To call the COS API, you use temporary keys to calculate a signature for identity authentication.

When a COS API request uses a temporary key to calculate the signature for authentication, the following three fields in the message returned by the temporary key API are required:

TmpSecretId

TmpSecretKey

Token

## Benefits to Use a Temporary Key

When using COS on web, iOS, and Android applications, fixed keys are less ideal for permission management and less secure if stored in your client-side code, as the key may be disclosed. In this case, you can use a temporary key. For example, when applying for a temporary key, you can specify the action and resource by setting the [policy](#) field to grant limited access permissions.

For COS API authorization policies, see:

[Working with COS API Access Policies](#)

[Examples of Temporary Key Authorization Policies in Common Scenarios.](#)

## Getting a Temporary Key

You can get a temporary key via the [COS STS SDK](#) or by calling the STS API [GetFederationToken](#) directly.

**Note:**

The Java SDK is used as an example in this document. To use it, you need to get the SDK code (version number) on GitHub. If you cannot find the SDK version number, check whether this SDK version is available on GitHub.

## COS STS SDK

COS provides SDKs and samples in various languages (e.g., Java, Node.js, PHP, Python, and Go) for STS. For more information, please see [COS STS SDK](#). To learn about how to use each SDK, see the README files and samples on GitHub by referring to the following table:

Language	Download Address	Sample
Java	<a href="#">Download</a>	<a href="#">View Sample</a>
.NET	<a href="#">Download</a>	<a href="#">View Sample</a>
Go	<a href="#">Download</a>	<a href="#">View Sample</a>
Node.js	<a href="#">Download</a>	<a href="#">View Sample</a>
PHP	<a href="#">Download</a>	<a href="#">View Sample</a>
Python	<a href="#">Download</a>	<a href="#">View Sample</a>

**Note:**

To avoid the differences between versions of the STS API, STS SDKs take a return parameters structure that may be different from that of the STS API. For more information, see [Java SDK documentation](#).

Here is an example to obtain a temporary key using the downloaded [Java SDK](#):

### Sample codes

```
// Import `java sts sdk` using the integration method with Maven as described on Gi
public class Demo {
    public static void main(String[] args) {
        TreeMap<String, Object> config = new TreeMap<String, Object>();

        try {
            // `SecretId` and `SecretKey` represent permanent identities (root acco
            String secretId = System.getenv("secretId");// User `SecretId`. We reco
            String secretKey = System.getenv("secretKey");// User `SecretKey`. We
            // Replace it with your Cloud API key SecretId
            config.put("secretId", secretId);
            // Replace it with your Cloud API key SecretKey
            config.put("secretKey", secretKey);
        }
    }
}
```

```
// Set a domain:
// If you use Tencent Cloud CVMs, you can set an internal domain.
//config.put("host", "sts.internal.tencentcloudapi.com");

// Validity period of the key, in seconds (default: 1800). The value ca
config.put("durationSeconds", 1800);

// Replace it with your own bucket
config.put("bucket", "examplebucket-1250000000");
// Replace it with the region where your bucket resides
config.put("region", "ap-guangzhou");

// Change it to an allowed path prefix. You can determine the upload pa
// Examples of several typical prefix authorization scenarios:
// 1. Allow access to all objects: "*"
// 2. Allow access to specified objects: "a/a1.txt", "b/b1.txt"
// 3. Allow access to objects with specified prefixes: "a*", "a/*", "b/
// If "*" is entered, you allow the user to access all resources. Unles
config.put("allowPrefixes", new String[] {
    "exampleobject",
    "exampleobject2"
});

// A list of permissions needed for the key (required)
// The following permissions are required for simple, form-based, and m
String[] allowActions = new String[] {
    // Simple upload
    "name/cos:PutObject",
    // Upload using a form or Weixin Mini Program
    "name/cos:PostObject",
    // Multipart upload
    "name/cos:InitiateMultipartUpload",
    "name/cos:ListMultipartUploads",
    "name/cos:ListParts",
    "name/cos:UploadPart",
    "name/cos:CompleteMultipartUpload"
};
config.put("allowActions", allowActions);
/**
 * Set `condition` (if necessary)
 *// # Condition for the temporary key to take effect. For the detailed c
final String raw_policy = "{\n" +
    "  \"version\": \"2.0\",\n" +
    "  \"statement\": [\n" +
    "    {\n" +
    "      \"effect\": \"allow\", \n" +
```

```

    \\ "action\\": [\\n" +
    \\ "name/cos:PutObject\\", \\n" +
    \\ "name/cos:PostObject\\", \\n" +
    \\ "name/cos:InitiateMultipartUpload\\", \\n" +
    \\ "name/cos:ListMultipartUploads\\", \\n" +
    \\ "name/cos:ListParts\\", \\n" +
    \\ "name/cos:UploadPart\\", \\n" +
    \\ "name/cos:CompleteMultipartUpload\\" \\n" +
    ], \\n" +
    \\ "resource\\": [\\n" +
    \\ "qcs::cos:ap-shanghai:uid/1250000000:examplebucket-125000
    ], \\n" +
    \\ "condition\\": { \\n" +
    \\ "ip_equal\\": { \\n" +
    \\ "qcs:ip\\": [ \\n" +
    \\ "192.168.1.0/24\\", \\n" +
    \\ "101.226.100.185\\", \\n" +
    \\ "101.226.100.186\\" \\n" +
    ] \\n" +
    } \\n" +
    } \\n" +
    ] \\n" +
    " }";

    config.put("policy", raw_policy);
    */

    Response response = CosStsClient.getCredential(config);
    System.out.println(response.credentials.tmpSecretId);
    System.out.println(response.credentials.tmpSecretKey);
    System.out.println(response.credentials.sessionToken);
} catch (Exception e) {
    e.printStackTrace();
    throw new IllegalArgumentException("no valid secret !");
}
}
}

```

## FAQs

### What should I do if NoSuchMethodError occurs due to JSONObject package conflict?

Use 3.1.0 or a later version.

## Accessing COS using a temporary key

When a COS API accesses COS with a temporary key, it passes the temporary `sessionToken` through the `x-cos-security-token` field, and calculates the signature using the temporary `SecretId` and `SecretKey`.

The following example shows how to use a temporary key obtained by use of COS Java SDK to access COS:

**Note:**

Before you run the sample, go to the [GitHub project](#) to download the Java SDK installation package.

```
// Import `cos xml java sdk` using the integration method with Maven as described o
import com.qcloud.cos.*;
import com.qcloud.cos.auth.*;
import com.qcloud.cos.exception.*;
import com.qcloud.cos.model.*;
import com.qcloud.cos.region.*;
public class Demo {
    public static void main(String[] args) throws Exception {

        // Basic user information
        String tmpSecretId = "COS_SECRETID"; // Replace it with the temporary Sec
        String tmpSecretKey = "COS_SECRETKEY"; // Replace it with the temporary Se
        String sessionToken = "Token"; // Replace it with the temporary token retu

        // 1. Initialize user authentication information (`secretId`, `secretKey`).
        COSCredentials cred = new BasicCOSCredentials(tmpSecretId, tmpSecretKey);
        // 2. Set the bucket region. For more information on COS regions, visit htt
        ClientConfig clientConfig = new ClientConfig(new Region("ap-guangzhou"));
        // 3. Generate a COS client
        COSClient cosclient = new COSClient(cred, clientConfig);
        // The bucket name must contain `appid`.
        String bucketName = "examplebucket-1250000000";

        String key = "exampleobject";
        // Upload an object. You are advised to call this API to upload objects sma
        File localFile = new File("src/test/resources/text.txt");
        PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, l

        // Set the `x-cos-security-token` header field.
        ObjectMetadata objectMetadata = new ObjectMetadata();
        objectMetadata.setSecurityToken(sessionToken);
        putObjectRequest.setMetadata(objectMetadata);

        try {
            PutObjectResult putObjectResult = cosclient.putObject(putObjectRequest)
            // Success: PutObjectResult returns the file ETag.
            String etag = putObjectResult.getETag();
        } catch (CosServiceException e) {
            //Failure: CosServiceException is thrown.
            e.printStackTrace();
        }
    }
}
```

```
    } catch (CosClientException e) {  
        //Failure: CosClientException is thrown.  
        e.printStackTrace();  
    }  
  
    // Disable the client  
    cosclient.shutdown();  
  
    }  
}
```



# Authorizing Sub-Account to Get Buckets by Tag

Last updated : 2024-03-25 15:11:18

## Overview

COS allows you to filter buckets by tag in the console or via the API, which is implemented based on authorization by tag.

## Authorization steps

1. Log in to the [CAM console](#) with the root account `Owner` and enter the policy configuration page.
2. Grant sub-account `SubUser` access to buckets with the specified tag through the **policy generator** or **policy syntax** as follows:

Policy generator

Policy syntax

1. Go to the [CAM policy configuration](#) page.
2. Click **Create Custom Policy > Create by Policy Generator**.
3. On the permission configuration page, configure the following:

**Effect:** use the default option Allow.

**Service:** Select COS.

**Operation:** Select **Read > GetService** (pulling the bucket list).

**Resource:** Select **All resources**.

**Condition:** Click **Add other conditions**. On the panel, configure the following:

**Condition Key:** Select `qcs:resource_tag`.

**Operator:** Select `string_equal`.

**Condition Value:** Enter a tag in the format of `key&val`. Here, replace `key` and `value` with the tag key and value respectively.

4. Click **Next** and enter the policy name.

5. Click **OK** to complete the process.

1. Go to the [CAM policy configuration](#) page.
2. Click **Create Custom Policy > Create by Policy Syntax**.
3. Select **Blank Template** and click **Next**.

4. Enter a policy in the following format. Here, replace `key` and `value` with the specified tag key and value respectively.

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "name/cos:GetService"
      ],
      "resource": "*",
      "condition": {
        "for_any_value:string_equal": {
          "qcs:resource_tag": [
            "key&value"
          ]
        }
      }
    }
  ]
}
```

5. Click **OK** to complete the process.

3. Associate the policy with the sub-account `SubUser` by locating the policy created in step 2 on the **Policies** page and clicking **Associate User/User Group/Role** on the right.

4. In the pop-up window, select the sub-account `SubUser` and click **OK**.

## Viewing in the console

1. Log in to the [COS console](#) with the sub-account `SubUser`.

2. The **Bucket List** page **automatically displays the list of buckets to which the sub-account has access**.

At this point, you have granted the sub-account access to buckets with the specified tag (key and value).

## Calling the API

### Note:

Unlike the console, the `GetService` API cannot automatically display the list of buckets to which the sub-account has access and requires you to pass in tag parameters.

The `GetService` API currently allows you to pass in only one tag.

1. Initiate a request with the key of the sub-account `SubUser` .
2. Call the `GetService` API to pass in the tag filtering parameters such as `(key, value)` . Below is a sample request. For more information, see [GET Service \(List Buckets\)](#).

```
GET /?tagkey=key1&tagvalue=value1 HTTP/1.1  
Date: Fri, 24 May 2019 11:59:51 GMT  
Authorization: Auth String
```

# Descriptions and Use Cases of Condition Keys

Last updated : 2024-03-25 15:11:17

When using access policies to grant permissions, you can specify policy conditions to restrict user access sources and the storage classes of uploaded files as instructed in [Access Policy Language > Overview](#).

This document provides common examples of using COS condition keys in bucket policies. You can view all the condition keys supported by COS and applicable requests [here](#).

## Note:

When using condition keys in writing a bucket policy, comply with the principle of least privilege, add the corresponding condition keys only applicable requests (actions), and avoid using the \* wildcard when specifying the actions. Using the wildcard will cause the requests to fail. For more information about condition keys, see [here](#).

When you create a policy in the CAM console, pay attention to the syntax format. The syntax elements of `version`, `principal`, `statement`, `effect`, `action`, `resource`, and `condition` must all begin with a letter in the same letter case.

## Use Cases of Condition Keys

### Restricting user access IPs (qcs:ip)

#### Condition key qcs:ip

You can use the `qcs:ip` condition key to restrict user access IPs. This condition key is applicable to all requests.

#### Example: allowing only user access from specified IPs

The policy in this example allows the sub-account with ID 100000000002 under the root account with ID 100000000001 (APPID: 1250000000) to upload and download objects regarding the `examplebucket-bj` bucket in Beijing region and the `exampleobject` object in the `examplebucket-gz` bucket in Guangzhou region, on condition that the access IP falls within the IP range `192.168.1.0/24` or is `101.226.100.185` or `101.226.100.186`.

```
{
  "version": "2.0",
  "principal": {
    "qcs": [
      "qcs::cam::uin/100000000001:uin/100000000002"
    ]
  },
}
```

```
"statement": [
  {
    "effect": "allow",
    "action": [
      "name/cos:PutObject",
      "name/cos:GetObject"
    ],
    "resource": [
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-gz-1250000000/exampleobject"
    ],
    "condition": {
      "ip_equal": {
        "qcs:ip": [
          "192.168.1.0/24",
          "101.226.100.185",
          "101.226.100.186"
        ]
      }
    }
  }
]
```

## Restricting VPC IDs (vpc:requester\_vpc)

### Condition key vpc:requester\_vpc

You can use the condition key `vpc:requester_vpc` to restrict the `vpcid` of user access. For more information on `vpcid`, see [Virtual Private Cloud](#).

### Example: restricting the VPC ID to aqp5jrc1

The policy in this example allows the sub-account with ID 100000000002 that belongs to the root account with ID 100000000001 (APPID: 1250000000) to access the bucket `examplebucket-1250000000`, on condition that the VPC ID is `aqp5jrc1`.

```
{
  "statement": [
    {
      "action": [
        "name/cos:*"
      ],
      "condition": {
        "string_equal": {
          "vpc:requester_vpc": [
```

```

        "vpc-aqp5jrc1"
      ]
    }
  },
  "effect": "allow",
  "principal": {
    "qcs": [
      "qcs::cam::uin/100000000001:uin/100000000002"
    ]
  },
  "resource": [
    "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*"
  ]
}
],
"version": "2.0"
}

```

## Allowing access only to the latest or specified version of an object (cos:versionid)

### Request parameter versionid

The `versionid` request parameter specifies the version number of the object. For more information on versioning, see [Overview](#). When downloading an object ( `GetObject` ) or deleting an object ( `DeleteObject` ), you can use `versionid` to specify the object version to be manipulated. There are three different cases with `versionid` :

If `versionid` is not carried, requests will apply to the latest version of the object by default.

If `versionid` is an empty string, this is equivalent to the case where `versionid` is not carried.

If `versionid` is `"null"` , for objects that are uploaded before versioning is enabled for a bucket, their version numbers will become the `"null"` string after versioning is enabled.

### Condition key cos:versionid

You can use the `cos:versionid` condition key to restrict the `versionid` request parameter.

### Example 1: allowing users to get objects of a specified version

Assume that the root account with UIN 100000000001 that owns the bucket `examplebucket-1250000000` uses the following bucket policy to allow the sub-account with UIN 100000000002 to get objects of a specified version only.

According to the policy, object download requests sent the sub-account with UIN 100000000002 can be successful only when they carry the `versionid` parameter and the value of `versionid` is the version number

```
Tg0NDUxNTc1NjIzMTQ1MDAwODg .
```

```

{
  "version": "2.0",

```

```

"statement": [
  {
    "principal": {
      "qcs": [
        "qcs::cam::uin/100000000001:uin/100000000002"
      ]
    },
    "effect": "allow",
    "action": [
      "name/cos:GetObject"
    ],
    "condition": {
      "string_equal": {
        "cos:versionid": "MTg0NDUxNTc1NjIzMTQ1MDAwODg"
      }
    },
    "resource": [
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
    ]
  }
]
}

```

### Adding a deny policy

If you use the above policy to grant the sub-user permissions, and the sub-user obtains the same permissions without any conditions attached through other means, the broader authorization policy takes effect. For example, if a sub-user is in a user group and the root account grants the GetObject permission to the user group without any conditions attached, the restriction on the version number of the above policy does not take effect.

To cope with this, you can add an explicit deny policy based on the above policy to achieve tighter permission restrictions. The following deny policy specifies that, when a sub-user initiates an object download request that does not carry the `versionid` parameter or that the version number specified by `versionid` is not `MTg0NDUxNTc1NjIzMTQ1MDAwODg`, the request will be denied. Because the priority of the deny policy is higher than other policies, adding a deny policy can avoid permission vulnerabilities to the maximum extent.

```

{
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",

```

```

    "action": [
      "name/cos:GetObject"
    ],
    "condition": {
      "string_equal": {
        "cos:versionid": "MTg0NDUxNTc1NjIzMTQ1MDAwODg"
      }
    },
    "resource": [
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
    ]
  },
  {
    "principal": {
      "qcs": [
        "qcs::cam::uin/100000000001:uin/100000000002"
      ]
    },
    "effect": "deny",
    "action": [
      "name/cos:GetObject"
    ],
    "condition": {
      "string_not_equal_if_exist": {
        "cos:versionid": "MTg0NDUxNTc1NjIzMTQ1MDAwODg"
      }
    },
    "resource": [
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
    ]
  }
],
"version": "2.0"
}

```

### Example 2: allowing users to get objects of the latest version

Assume that the root account with UIN 100000000001 that owns the bucket `examplebucket-1250000000` uses the following bucket policy to allow the sub-account with UIN 100000000002 to get objects of the latest version only.

According to the policy, if `versionid` is not carried or its value is an empty string, a `GetObject` request will download an object of the latest version by default. Therefore, you can use `string_equal_if_exsit` in the condition:



1. If `versionid` is not carried, it is considered that the condition is met ( `True` ) by default, the `allow` policy is hit, and requests are allowed.
2. If `versionid` is an empty string ( `""` ), the `allow` policy will also be hit, and only requests for downloading objects of the latest version will be authorized.

```
"condition":{
  "string_equal_if_exist":{
    "cos:versionid": ""
  }
}
```

After the explicit deny policy is added, the complete bucket policy is as follows:

```
{
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "name/cos:GetObject"
      ],
      "condition": {
        "string_equal_if_exist": {
          "cos:versionid": ""
        }
      },
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ]
    },
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "deny",
      "action": [
        "name/cos:GetObject"
      ],
      "condition": {
        "string_not_equal": {
```

```

        "cos:versionid":""
      }
    },
    "resource": [
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
    ]
  }
],
"version": "2.0"
}

```

### Example 3: disallowing users from deleting objects uploaded before versioning is enabled

Your bucket may have some objects uploaded before versioning is enabled, and the version numbers of these objects become "null" after versioning is enabled. Sometimes, you may need to enable additional protection for these objects, for example, to prevent users from permanently deleting these objects, that is, to deny deletion of objects with version numbers.

In the example below, there are two bucket policies:

1. Authorize the sub-account to use `DeleteObject` requests to delete objects in the bucket.
2. Restrict the condition for `DeleteObject` requests. If a `DeleteObject` request carries the request parameter `versionid` with the value `"null"`, the request will be denied.

Therefore, if object A was uploaded to the bucket `examplebucket-1250000000` before versioning is enabled, the version number of object A becomes a "null" string after versioning is enabled.

After the bucket policy is added, object A will be protected. If a `DeleteObject` request initiated by a sub-user to delete object A does not carry a version number, object A will not be permanently deleted because versioning is enabled. Instead, a delete marker will be added for object A. If the request contains the "null" version number of object A, the request will be denied, and object A will not be permanently deleted.

```

{
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "name/cos:DeleteObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ]
    }
  ]
}

```

```
    ],
  },
  {
    "principal":{
      "qcs":[
        "qcs::cam::uin/100000000001:uin/100000000002"
      ]
    },
    "effect":"deny",
    "action":[
      "name/cos:DeleteObject"
    ],
    "condition":{
      "string_equal":{
        "cos:versionid":"null"
      }
    },
    "resource":[
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
    ]
  }
],
"version":"2.0"
}
```

## Restricting the size of the file to upload (cos:content-length)

### Request header Content-Length

The length of the content of an HTTP request in bytes defined in RFC 2616 is often used in PUT and POST requests. For more information, see [Common Request Headers](#).

### Condition key cos:content-length

When uploading an object, you can use the `cos:content-length` condition key to restrict the `Content-Length` request header to limit the file size of the uploaded object. In this way, you can flexibly manage storage space and avoid wasting storage space and network bandwidth by uploading files that are too large or too small. In the two examples below, the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the `cos:content-length` condition key to restrict the value of the `Content-Length` header in upload requests initiated by the sub-account with UIN 100000000002.

### Example 1: restricting the maximum value of the request header Content-Length

Require that `PutObject` and `PostObject` upload requests carry the `Content-Length` header with a value less than or equal to 10 bytes.

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "name/cos:PutObject",
        "name/cos:PostObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-
1250000000/*"
      ],
      "condition": {
        "numeric_less_than_equal": {
          "cos:content-length": 10
        }
      }
    },
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "deny",
      "action": [
        "name/cos:PutObject",
        "name/cos:PostObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-
1250000000/*"
      ],
      "condition": {
        "numeric_greater_than_if_exist": {
          "cos:content-length": 10
        }
      }
    }
  ]
}
```

```
}
```

## Example 2: restricting the minimum value of the request header Content-Length

Require that `PutObject` and `PostObject` upload requests carry the `Content-Length` header with a value not less than 2 bytes.

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "name/cos:PutObject",
        "name/cos:PostObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "numeric_greater_than_equal": {
          "cos:content-length": 2
        }
      }
    },
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "deny",
      "action": [
        "name/cos:PutObject",
        "name/cos:PostObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
```

```

        "numeric_less_than_if_exist":{
            "cos:content-length":2
        }
    }
}
]
}

```

## Restricting the type of the file to upload (cos:content-type)

### Request header Content-Type

`Content-Type` must be an HTTP request content type as defined in RFC 2616 (MIME), such as `application/xml` and `image/jpeg`. For more information, see [Common Request Headers](#).

### Condition key cos:content-type

You can use the `cos:content-type` condition key to restrict the `Content-Type` request header.

### Example 1: restricting Content-Type of PutObject requests to "image/jpeg"

Assume that the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the `cos:content-type` condition key to restrict the content of the `Content-Type` header in upload requests initiated by the sub-account with UIN 100000000002.

The bucket policy in this example is to restrict that object upload requests ( `PutObject` ) must carry the `Content-Type` header and with the value `image/jpeg`.

Note that `string_equal` requires that the request must carry the `Content-Type` header with a value exactly the same as the specified value. In a real request, you need to **explicitly specify the Content-Type header of the request**. Otherwise, if your request does not carry the `Content-Type` header, the request will fail. In addition, if you use a certain tool to initiate a request and do not explicitly specify `Content-Type`, the tool may automatically add an unexpected `Content-Type` header to the request, the request may also fail.

```

{
  "version":"2.0",
  "statement":[
    {
      "principal":{
        "qcs":[
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect":"allow",
      "action":[
        "name/cos:PutObject"
      ],
    }
  ]
}

```

```

        "resource": [
            "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-
1250000000/*"
        ],
        "condition": {
            "string_equal": {
                "cos:content-type": "image/jpeg"
            }
        }
    },
    {
        "principal": {
            "qcs": [
                "qcs::cam::uin/100000000001:uin/100000000002"
            ]
        },
        "effect": "deny",
        "action": [
            "name/cos:PutObject"
        ],
        "resource": [
            "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-
1250000000/*"
        ],
        "condition": {
            "string_not_equal_if_exist": {
                "cos:content-type": "image/jpeg"
            }
        }
    }
]
}

```

## Restricting the file type returned by download request (cos:response-content-type)

### Request parameter response-content-type

The `GetObject` API allows you to add the `response-content-type` request parameter to specify the value of the `Content-Type` header in the response.

### Condition key cos:response-content-type

You can use the `cos:response-content-type` condition key to specify whether requests need to carry `response-content-type`.

### Example 1: restricting the GetObject request parameter response-content-type to be "image/jpeg"

Assume that the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the following bucket policy to require that `GetObject` requests initiated by the sub-account with UIN 100000000002 carry the `response-content-type` request parameter with the value `image/jpeg`. `response-content-type` is a request parameter and needs to be URL-encoded when the request is initiated (encoded value: `response-content-type=image%2Fjpeg`). Therefore, when you set the policy, "image/jpeg" also needs to be URL-encoded, that is, `image%2Fjpeg` needs to be entered.

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "name/cos:GetObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "string_equal": {
          "cos:response-content-type": "image%2Fjpeg"
        }
      }
    },
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "deny",
      "action": [
        "name/cos:GetObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
```



```
        "string_not_equal_if_exist":{
            "cos:response-content-type":"image%2Fjpeg"
        }
    }
}
]
```

## Allowing only HTTPS requests (cos:secure-transport)

### Condition key cos:secure-transport

You can use the `cos:secure-transport` condition key to require requests to use the HTTPS protocol.

### Example 1: restricting download requests to use HTTPS

Assume that the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the following bucket policy to allow only HTTPS-based `GetObject` requests sent by the sub-account with UIN 100000000002.

```
{
  "version":"2.0",
  "statement":[
    {
      "principal":{
        "qcs":[
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect":"allow",
      "action":[
        "name/cos:GetObject"
      ],
      "resource":[
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition":{
        "bool_equal":{
          "cos:secure-transport":"true"
        }
      }
    }
  ]
}
```

## Example 2: denying any non-HTTPS request

Assume that the root account with UIN 100000000001 that owns the bucket `examplebucket-1250000000` uses the following bucket policy to deny any non-HTTPS requests sent by the sub-account with UIN 100000000002.

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "deny",
      "action": [
        "*"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "bool_equal": {
          "cos:secure-transport": "false"
        }
      }
    }
  ]
}
```

## Allowing setting a specified storage class (cos:x-cos-storage-class)

### Request header x-cos-storage-class

You can use the `x-cos-storage-class` request parameter to specify or modify the storage class of an object when uploading the object.

### Condition key cos:x-cos-storage-class

You can use the `cos:x-cos-storage-class` condition key to restrict the `x-cos-storage-class` request header to restrict storage class modification requests.

COS's storage class fields include `STANDARD` , `MAZ_STANDARD` , `STANDARD_IA` , `MAZ_STANDARD_IA` , `INTELLIGENT_TIERING` , `MAZ_INTELLIGENT_TIERING` , `ARCHIVE` , and `DEEP_ARCHIVE` .

## Example 1: requiring PutObject requests to set the storage class to STANDARD

Assume that the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the following bucket policy to require `PutObject` requests sent by the sub-account with UIN 100000000002 to carry the `x-cos-storage-class` header with the value `STANDARD` .

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "name/cos:PutObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "string_equal": {
          "cos:x-cos-storage-class": "STANDARD"
        }
      }
    },
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "deny",
      "action": [
        "name/cos:PutObject"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "string_not_equal_if_exist": {
          "cos:x-cos-storage-class": "STANDARD"
        }
      }
    }
  ]
}
```

```
    }  
  ]  
}
```

## Allowing setting a specified bucket/object ACL (`cos:x-cos-acl`)

### Request header `x-cos-acl`

When uploading an object or creating a bucket, you can use the `x-cos-acl` request header to specify an ACL or modify the object or bucket ACL. For more information, see [ACL](#).

Preset ACLs for buckets: `private`, `public-read`, `public-read-write`, `authenticated-read`

Preset ACLs for objects: `default`, `private`, `public-read`, `authenticated-read`, `bucket-owner-read`, `bucket-owner-full-control`

### Condition key `cos:x-cos-acl`

You can use the `cos:x-cos-acl` condition key to restrict the `x-cos-acl` request header to restrict object/bucket ACL modification requests.

### Example 1: the object ACL must be set to private in a PutObject request

Assume that the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the following bucket policy to allow the sub-account with UIN 100000000002 to upload private objects only. The policy requires that all `PutObject` requests carry the `x-cos-acl` header with the value `private`.

```
{  
  "version": "2.0",  
  "statement": [  
    {  
      "principal": {  
        "qcs": [  
          "qcs::cam::uin/100000000001:uin/100000000002"  
        ]  
      },  
      "effect": "allow",  
      "action": [  
        "name/cos:PutObject"  
      ],  
      "resource": [  
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-  
1250000000/*"  
      ],  
      "condition": {  
        "string_equal": {  
          "cos:x-cos-acl": "private"  
        }  
      }  
    }  
  ]  
}
```

```

    }
  },
  {
    "principal":{
      "qcs":[
        "qcs::cam::uin/100000000001:uin/100000000002"
      ]
    },
    "effect":"deny",
    "action":[
      "name/cos:PutObject"
    ],
    "resource":[
      "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
    ],
    "condition":{
      "string_not_equal_if_exist":{
        "cos:x-cos-acl":"private"
      }
    }
  }
]
}

```

## Allowing listing objects in a specified directory only (cos:prefix)

### Condition key cos:prefix

You can use the `cos:prefix` condition key to restrict the `prefix` request parameter.

#### Note:

If the value of `prefix` contains special characters such as `/`, the value must be URL-encoded before being written into the bucket policy.

### Example 1: allowing listing only objects in a specified directory of the bucket

Assume that the root account with UIN 100000000001 that owns the bucket `examplebucket-1250000000` uses the following bucket policy to restrict the sub-account with UIN 100000000002 to list only the objects in the `folder1` directory of the bucket. The policy requires that all `GetBucket` requests carry the `prefix` parameter with the value `folder1`.

```

{
  "statement": [
    {
      "principal": {
        "qcs": [

```

```

        "qcs::cam::uin/100000000001:uin/100000000002"
    ]
},
"effect": "allow",
"action": [
    "name/cos:GetBucket"
],
"resource": [
    "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-
1250000000/*"
],
"condition": {
    "string_equal": {
        "cos:prefix": "folder1"
    }
}
},
{
    "principal": {
        "qcs": [
            "qcs::cam::uin/100000000001:uin/100000000002"
        ]
    },
    "effect": "deny",
    "action": [
        "name/cos:GetBucket"
    ],
    "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-
1250000000/*"
    ],
    "condition": {
        "string_equal_if_exist": {
            "cos:prefix": "folder1"
        }
    }
}
],
"version": "2.0"
}

```

## Allowing using the TLS protocol of a specified version only (cos:tls-version)

### Condition key cos:tls-version

You can use the `cos:tls-version` condition key to restrict the TLS version of HTTPS requests. Its value is of the numeric type and supports floating points, such as 1.0, 1.1, or 1.2.

### Example 1: authorizing only HTTP requests that use TLS v1.2

Request Scenario	Expected Result
HTTPS request using TLS v1.0	403, failed
HTTPS request using TLS v1.2	200, successful

A policy example is as follows:

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "*"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "numeric_equal": {
          "cos:tls-version": 1.2
        }
      }
    }
  ]
}
```

### Example 2: rejecting HTTP requests that use TLS earlier than v1.2

Request Scenario	Expected Result
HTTPS request using TLS v1.0	403, failed
HTTPS request using TLS v1.2	200, successful

A policy example is as follows:

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "*"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "numeric_greater_than_equal": {
          "cos:tls-version": 1.2
        }
      }
    },
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "deny",
      "action": [
        "*"
      ],
      "resource": [
        "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
      ],
      "condition": {
        "numeric_less_than_if_exist": {
          "cos:tls-version": 1.2
        }
      }
    }
  ]
}
```



## Forcibly setting a specified bucket tag when creating a bucket (qcs:request\_tag)

### Note:

The `request_tag` condition key is only applicable to `PutBucket` and `PutBucketTagging` operations but not `GetService`, `PutObject`, or `PutObjectTagging`.

### Condition key `qcs:request_tag`

You can use the `qcs:request_tag` condition key to restrict that a user must include a specified bucket tag when initiating a `PutBucket` or `PutBucketTagging` request.

### Example: restricting that a user must include a specified bucket tag when creating a bucket

Many users may manage their buckets using bucket tags. The following policy example indicates that the user can get authorization only after the user sets the specified bucket tags `<a,b>` and `<c,d>` when creating a bucket.

Multiple bucket tags can be set. Different bucket tag keys/values and tag quantities will be used as different combinations. Assume that multiple parameter values carried by the user in the request form set A and multiple parameter values specified in the condition form set B. With this condition key, the user can use different combinations of qualifiers `for_any_value` and `for_all_value` to indicate different meanings.

`for_any_value:string_equal` indicates that the request takes effect if A and B have an intersection.

`for_all_value:string_equal` indicates that the request takes effect if A is a subset of B.

If `for_any_value:string_equal` is used, the corresponding policy and request are as shown below:

Request Scenario	Expected Result
PutBucket, request header <code>x-cos-tagging:</code> <code>a=b&amp;c=d</code>	200, successful
PutBucket, request header <code>x-cos-tagging:</code> <code>a=b</code>	200, successful
PutBucket, request header <code>x-cos-tagging:</code> <code>a=b&amp;c=d&amp;e=f</code>	200, successful

A policy example is as follows:

```
{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
```

```

        "qcs::cam::uin/100000000001:uin/100000000002"
    ]
},
"effect": "allow",
"action": [
    "name/cos:PutBucket"
],
"resource": "*",
"condition": {
    "for_any_value:string_equal": {
        "qcs:request_tag": [
            "a&b",
            "c&d"
        ]
    }
}
}
]
}

```

If `for_all_value:string_equal` is used, the corresponding policy and request are as shown below:

Request Scenario	Expected Result
PutBucket, request header <code>x-cos-tagging:</code> <code>a=b&amp;c=d</code>	200, successful
PutBucket, request header <code>x-cos-tagging:</code> <code>a=b</code>	200, successful
PutBucket, request header <code>x-cos-tagging:</code> <code>a=b&amp;c=d&amp;e=f</code>	403, failed

A policy example is as follows:

```

{
  "version": "2.0",
  "statement": [
    {
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "effect": "allow",
      "action": [
        "name/cos:PutBucket"
      ]
    }
  ]
}

```

```
    ],
    "resource": "*",
    "condition": {
      "for_all_value:string_equal": {
        "qcs:request_tag": [
          "a&b",
          "c&d"
        ]
      }
    }
  }
]
}
```

# Granting Bucket Permissions to a Sub-Account that is Under Another Root Account

Last updated : 2024-06-03 11:10:51

## Overview

There are two buckets under root account A ( APPID : 1250000000 ): examplebucket1-1250000000 and examplebucket2-1250000000 , which sub-account B0 under root account B wants to manipulate to meet its business needs. This document describes how to authorize it to do so.

## Directions

### Authorizing root account B to manipulate buckets under root account A

1. Log in to the [COS console](#) with root account A.
2. Click **Bucket List**, find the bucket to be authorized, and click its name to enter the bucket details page.
3. On the left sidebar, click **Permission Management** to enter the bucket's permission management page.
4. Find the **Permission Policy Settings** configuration item, click **Add Policy**, select a custom policy, and click **Next**.
5. Fill in the form as shown below:

**Policy ID:** Optional.

**Effect:** Allowed.

**User:** Click "Add User", select root account for user type, and enter root account B's UIN for account ID, such as 100000000002.

**Resource:** Select an option as needed (the entire bucket by default).

**Resource Path:** It needs to be entered only for specified resources.

**Operation:** Click "Add Operation" and select all operations. If you want to grant root account B permissions to only certain operations, you can also select one or more operations as needed.

**Filter:** Add a filter or leave it blank as needed.

6. Click **Finish** to grant root account B specified permissions to the bucket.
7. If you need to authorize root account B to manipulate other buckets, repeat the above steps.

### Authorizing sub-account B0 to manipulate buckets under root account A

1. Log in to the CAM Console with root account B and go to the [Policy](#) page.
2. Select **Create by Policy Syntax > Create by Policy Syntax > Blank Template** and click **Next**.

**Note:**

Root account B can grant its sub-account B0 permissions only using a custom policy, but not a preset policy.

3. Fill in the form as shown below:

**Policy Name:** Designate a unique and meaningful name for the policy, such as `cos-child-account` .

**Remarks:** Optional; add remarks as needed.

**Policy Content:**

```
{
  "version": "2.0",
  "statement": [
    {
      "action": "cos:*",
      "effect": "allow",
      "resource": [
        "qcs::cos::uid/1250000000:examplebucket1-1250000000/*",
        "qcs::cos::uid/1250000000:examplebucket2-1250000000/*"
      ]
    }
  ]
}
```

The policy above grants sub-account B0 the permissions to operate on all the buckets under account A that root account B is authorized to access. Here, `1250000000` in `uid/1250000000` refers to the APPID of root account A, and `examplebucket1-1250000000` and `examplebucket2-1250000000` are the buckets under account A that root account B is authorized to operate.

4. Click **Complete**.

5. Locate the created policy in the **policy list** and click **Associate User/User Group/Role** on the right.

6. In the pop-up window, select sub-account B0 and click **OK**.

7. Then, the authorization is completed, and you can use the key of sub-account B0 to manipulate the bucket under root account A.

# Performance Optimization

## Request Rate and Performance Optimization

Last updated : 2024-03-25 15:20:11

### Note:

Currently, COS can achieve a high QPS through the underlying index distribution mechanism. If you need a higher QPS, [contact us](#) for assistance. In the daily process of organizing files, we still recommend you follow the guidelines in this document and avoid excessively centralized index storage.

## Overview

This document describes the best practices for optimizing the request rate performance in COS.

COS supports a typical workload capacity of 30,000 PUT or GET requests per second. If your workload exceeds the threshold, you can follow this guide to expand and optimize your request rate performance.

### Note:

The request load refers to the number of requests initiated per second rather than the number of concurrent connections. In other words, you can still send hundreds of new connection requests per second while maintaining thousands of existing connections.

COS supports performance expansion to provide a higher request rate. In case of a high GET request load, we recommend you use COS in combination with CDN. For more information, see [Overview](#). If the overall request rate of a bucket is expected to exceed 30,000 PUT/LIST/DELETE requests per second, [contact us](#) to prepare for the workload and avoid exceeding the request limit.

### Note:

If you have a mixed request load that only occasionally reaches 30,000 per second and does not exceed 30,000 per second during bursts, you can ignore this guide.

## Directions

### Mixed request load

When a large number of objects need to be uploaded, the object key you select may cause performance issues.

Below is a brief description of how COS stores object key values.

Tencent Cloud maintains bucket and object key values as indexes in each service region of COS. Object keys are stored in the UTF-8 binary order in multiple index partitions. Due to such a large number of key values, using timestamps or alphabetical order, for example, may exhaust the read/write performance capacity of the partition

where the key values are located. Taking the bucket path `examplebucket-1250000000.cos.ap-beijing.myqcloud.com` as an example, below are some cases that may exhaust the index performance capacity:

```
20170701/log120000.tar.gz
20170701/log120500.tar.gz
20170701/log121000.tar.gz
20170701/log121500.tar.gz
...
image001/indexpage1.jpg
image002/indexpage2.jpg
image003/indexpage3.jpg
...
```

If your typical workload exceeds 30,000 requests per second, you should avoid using sequential key values as shown in the above case. When you need to use characters such as sequential numbers, dates, or time values as object keys, you can add random prefixes to key names, so as to manage key values in multiple index partitions and improve the centralized load performance. Below are some methods for adding a random element to key values.

#### Note:

All the following methods can be used to improve the access performance of a single bucket. If the typical load of your business exceeds 30,000 requests per second, you still need to [contact us](#) to prepare for your business load in advance.

### Adding hexadecimal hash prefixes

The most direct way to increase the object key randomness is to add a hash string prefix at the beginning of the key name. For example, when uploading an object, calculate the SHA1 or MD5 hash of the path key value and add a few characters as a prefix to the key name. Generally, a hash prefix with a length of 2–4 characters will suffice.

```
faf1-20170701/log120000.tar.gz
e073-20170701/log120500.tar.gz
333c-20170701/log121000.tar.gz
2c32-20170701/log121500.tar.gz
```

#### Note:

As key values in COS are indexed in the UTF-8 binary order, you may need to initiate 65,536 GET Bucket operations to get the original complete `20170701` prefix structure.

### Adding enumerated value prefixes

If you still want to ensure that your object keys are easily retrievable, you can enumerate prefixes based on file type to help group your objects. Prefixes with the same enumeration value share the performance of the index partition where they are located.

```
logs/20170701/log120000.tar.gz
logs/20170701/log120500.tar.gz
```

```
logs/20170701/log121000.tar.gz
...
images/image001/indexpage1.jpg
images/image002/indexpage2.jpg
images/image003/indexpage3.jpg
...
```

If the access load for an enumerated prefix remains at over 30,000 requests per second, you can refer to the previous method to add a hash prefix after the enumerated value to implement multiple index partitions. This can further improve the read/write performance.

```
logs/faf1-20170701/log120000.tar.gzlogs/
e073-20170701/log120500.tar.gz
logs/333c-20170701/log121000.tar.gz
...
images/0165-image001/indexpage1.jpg
images/a349-image002/indexpage2.jpg
images/ac00-image003/indexpage3.jpg
...
```

### Reversing the key name string

When you need to use incremental IDs or dates or upload a large number of objects with successive prefixes in a single request, refer to the following method:

```
20170701/log0701A.tar.gz
20170701/log0701B.tar.gz
20170702/log0702A.tar.gz
20170702/log0702B.tar.gz
...
id16777216/album/hongkong/img20170701121314.jpg
id16777216/music/artist/tony/anythinggoes.mp3
id16777217/video/record20170701121314.mov
id16777218/live/show/date/20170701121314.mp4
...
```

The naming method for key values shown above easily exhausts the performance capacity of index partitions where the key values prefixed with `2017` and `id` are located. In this case, reverse part of the key prefix to allow for a certain degree of randomness.

```
10707102/log0701A.tar.gz
10707102/log0701B.tar.gz
20707102/log0702A.tar.gz
20707102/log0702B.tar.gz
...
```



```
61277761di/album/hongkong/img20170701121314.jpg  
61277761di/music/artist/tony/anythinggoes.mp3  
71277761di/video/record20170701121314.mov  
81277761di/live/show/date/20170701121314.mp4  
...
```

## High GET request load

If your workload primarily involves GET requests (i.e., download requests), in addition to the above guidelines, we recommend you use COS in combination with CDN.

CDN has edge cache nodes around the globe that can be used to minimize the latency and improve the speed of content delivery to users. Frequently accessed files can be cached with the prefetch feature, thus reducing the number of GET requests forwarded to the COS origin. For more information, see [Overview](#).

# Working with COSBench

Last updated : 2024-11-18 14:12:26

## COSBench Overview

COSBench is an open-source stress test tool developed by Intel for testing the performance of cloud object storage systems. As a cloud storage system compatible with S3 protocol, COSBench can be used to perform benchmark tests on the read/write performance of COS.

## System Environment

It is recommended that you run COSBench in CentOS 7.0 or a later version. If you run it in Ubuntu, unexpected issues may occur.

## Performance Factors

**CPU cores:** a small number of CPU cores coupled with a large number of running workers is very likely to cause high overheads for context switching. Therefore, 32 or 64 cores are recommended for the test.

**NIC:** the outbound traffic from the server is limited. To test the traffic for large files, an NIC above 10 GB is recommended.

**Network link:** public network links vary in quality. Charges will incur for public network downstream traffic for downloads over public network. Therefore, private network is recommended for access within the same region.

**Test duration:** in order to get a reliable value, we recommend a longer test period.

**Test environment:** the version of the JDK running on your program is also a key performance factor. For example, when testing on HTTPS, with an earlier JDK version, [GCM bugs](#) may occur for the encryption algorithm, as well as the locking issues for the random number generator.

## Directions

1. Download COSBench 0.4.2.c4.zip from [GitHub](#) and decompress it on your server.
2. Install the COSBench dependent library and run the following command.

For CentOS, run the following command to install the dependencies:

```
sudo yum install nmap-ncat java curl java-1.8.0-openjdk-devel -y
```

For Ubuntu, run the following command to install the dependencies:

```
sudo apt install nmap openjdk-8-jdk
```

3. Edit the file `s3-config-sample.xml` and configure a test job. The test job is divided into the following five stages.

init: creates a bucket.

prepare: uses parallel threads (or workers) to PUT (upload) objects with specified size to read in the main stage.

main: uses parallel workers to read and write objects for a specified period of time.

cleanup: deletes the created objects.

dispose: deletes buckets.

The sample configuration is as shown below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<workload name="s3-50M-sample" description="sample benchmark for s3">

  <storage type="s3" config="accesskey=*****;secretk

<workflow>

  <workstage name="init">
    <work type="init" workers="10" config="cprefix=examplebucket;csuffix=-1250000
  </workstage>

  <workstage name="prepare">
    <work type="prepare" workers="100" config="cprefix=examplebucket;csuffix=-125
  </workstage>

  <workstage name="main">
    <work name="main" workers="100" runtime="300">
      <operation type="read" ratio="50" config="cprefix=examplebucket;csuffix=-12
      <operation type="write" ratio="50" config="cprefix=examplebucket;csuffix=-1
    </work>
  </workstage>

  <workstage name="cleanup">
    <work type="cleanup" workers="10" config="cprefix=examplebucket;csuffix=-1250
  </workstage>

  <workstage name="dispose">
    <work type="dispose" workers="10" config="cprefix=examplebucket;csuffix=-1250
  </workstage>

</workflow>
```

```
</workload>
```

**Parameter description:**

Parameter	Description
accesskey, secretkey	Key information. We recommend you use a sub-account key and follow the <a href="#">principle of least privilege</a> to reduce risks. For information about how to obtain a sub-account key, see <a href="#">Access Key</a> .
cprefix	Bucket name prefix, such as examplebucket.
containers	Value range for bucket names. A bucket name is made up of cprefix and containers, such as examplebucket1 or examplebucket2.
csuffix	Your APPID. Note that APPID is prefixed with a -, such as -1250000000.
runtime	Duration of the stress test
ratio	Ratio of reads to writes
workers	Number of stress test threads

4. Edit the file `cosbench-start.sh` and add the following parameter to the Java startup command line to disable S3 MD5 verification.

```
-Dcom.amazonaws.services.s3.disableGetObjectMD5Validation=true
```

```
#-----
# MAIN
#-----

rm -f $BOOT_LOG
mkdir -p log

echo "Launching osgi framework ... "

/usr/bin/nohup java -Dcom.amazonaws.services.s3.disableGetObjectMD5Validation=true -Dcosbe
.tomcat.config=$TOMCAT_CONFIG -server -cp main/* org.eclipse.equinox.launcher.Main -config
tion $OSGI_CONFIG -console $OSGI_CONSOLE_PORT 1> $BOOT_LOG 2>&1 &
```

5. Start the COSBench service.

For CentOS, run the following command:

```
sudo bash start-all.sh
```

For Ubuntu, run the following command:

```
sudo bash start-driver.sh &sudo bash start-controller.sh &
```

6. Run the following command to submit the job.

```
sudo bash cli.sh submit conf/s3-config-sample.xml
```

Check the test status at `http://ip:19088/controller/index.html` (replace the IP in this link with the IP of your own testing server).

COSBENCH - CONTROLLER WEB CONSOLE
time: Tue Jun 30 20:32:59 CST 20  
version: 0.4.2.201606

### Controller Overview 1

Name: *not configured* URL: *not configured*

Driver	Name	URL	IsAlive	Link
1	driver1	http://127.0.0.1:18088/driver	●	<a href="#">view details</a>

[submit new workloads](#)  
[config workloads](#)  
[advanced config for workloads](#)

### Active Workloads 1

ID	Name	Submitted-At	State	Order	Link
<input type="checkbox"/> w1	s3-50M-sample	2020-6-30 20:32:47	processing		<a href="#">view details</a>

### Historical Workloads 0

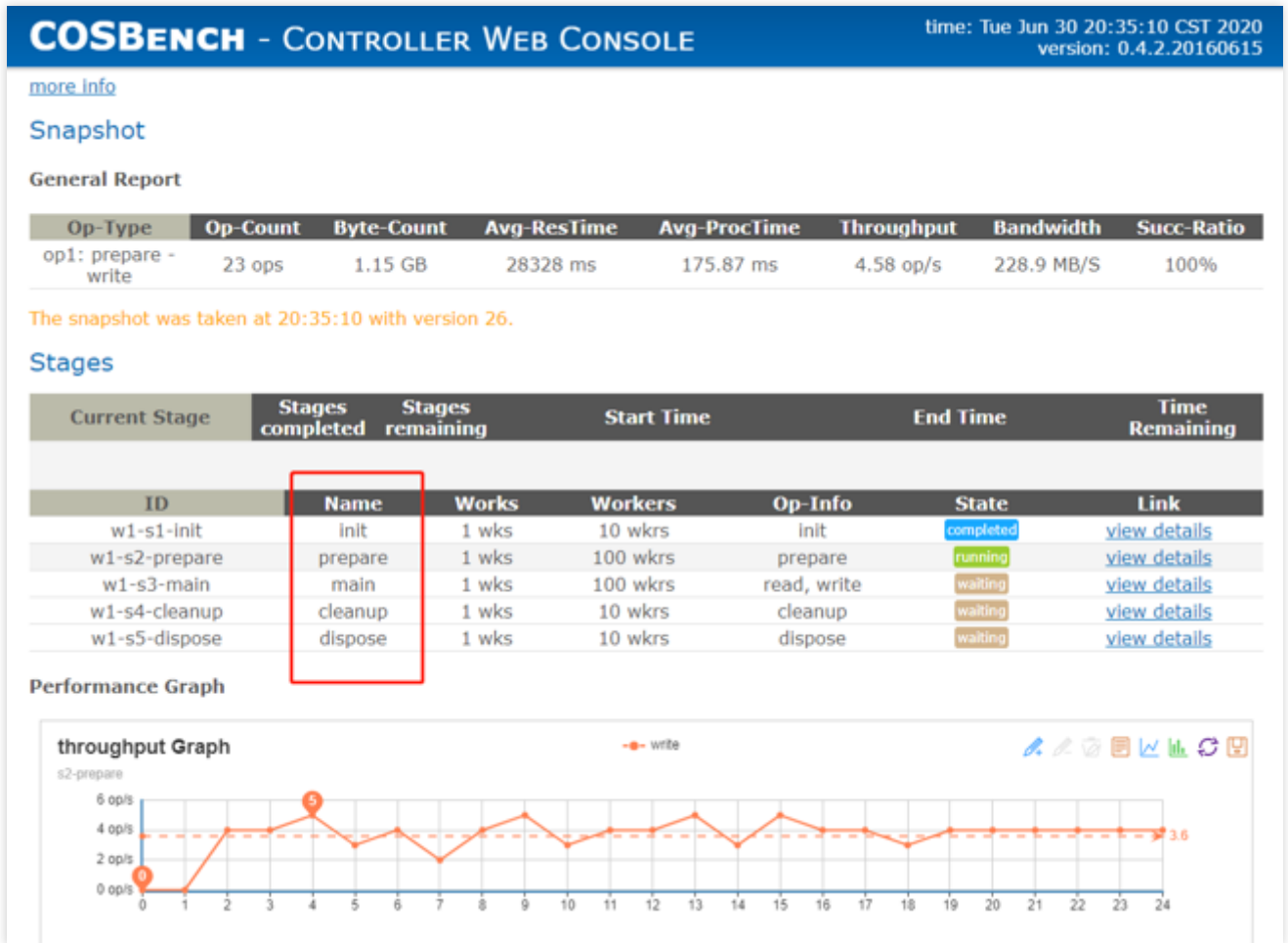
[view performance matrix](#)

ID	Name	Duration	Op-Info	State	Link
----	------	----------	---------	-------	------

### Archived Workloads 0

[view performance matrix](#)  
[load archived workloads](#)

You can see the five work stages as shown below.



7. The following example shows the performance tests of uploads and downloads on a CVM instance with 32 cores and 17 Gbps private network bandwidth in Beijing region. The test includes the following two stages.

prepare: 100 workers threads are run to upload 1,000 objects (50 MB each).

main: 100 workers read and write objects in parallel for 300 seconds.

The test results after two stages above are as shown below.

## Basic Info

ID: w27 Name: s3-50M-sample Current State: finished

Submitted At: 2019-3-20 10:39:53 Started At: 2019-3-20 10:39:53 Stopped At: 2019-3-20 10:50:29

[more info](#)

## Final Result

### General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Rat
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: prepare -write	10 kops	500 GB	2460.65 ms	181.83 ms	41.27 op/s	2.06 GB/S	100%
op1: read	10.22 kops	510.75 GB	2012.74 ms	118.33 ms	34.15 op/s	1.71 GB/S	100%
op2: write	10.28 kops	514.2 GB	908.98 ms	317.71 ms	34.38 op/s	1.72 GB/S	100%
op1: cleanup -delete	20 kops	0 B	14.19 ms	14.19 ms	704.74 op/s	0 B/S	100%
op1: dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

[show performance details](#)

8. Run the following command to stop the test.

```
sudo bash stop-all.sh
```

# Data Migration

## Migrating Local Data to COS

Last updated : 2024-12-18 14:18:02

### Practice Scenario

If you have a local IDC, COS offers the following migration methods to help you quickly migrate massive amounts of local data to COS.

Migration Method	Description
<a href="#">COS Migration</a> (online migration)	COS Migration is an all-in-one tool integrating COS data migration features. You can migrate data to COS quickly after completing simple configurations.

## Migration Practice

### COS Migration

The steps are as follows:

1. Install the Java environment.
2. Install the COS Migration tools.
3. Modify the configuration file.
4. Launch the tool.

For more information, please see [COS Migration Tool](#).

### Tips

Below describes how to configure COS Migration to maximize the migration speed:

1. Adjust the threshold and concurrence of large/small files based on your network environment to implement optimal migration where large files are split into multiple parts and small files are transferred concurrently. Adjust the tool execution time and set a bandwidth limit to ensure that bandwidth usage by data migration will not affect your business operations. Such adjustments can be made by modifying the following parameters in the `[common]` section of the `config.ini` configuration file:

Parameter Name	Parameter Description
<code>smallFileThreshold</code>	Threshold for small files. If the size of a file is higher than or equal to this threshold, multipart upload will be used; otherwise, simple upload will be used. The default value



	is 5 MB.
bigFileExecutorNum	Concurrence of large files, which is 8 by default. If COS is connected to over the public network with low bandwidth, reduce this value.
smallFileExecutorNum	Concurrence of small files, which is 64 by default. If COS is connected to over the public network with low bandwidth, reduce this value.
executeTimeWindow	This parameter defines the time range when the migration tool will execute a task every day, which will enter sleep mode at other times. In sleep mode, the migration will be paused and the migration progress will be retained until the next time window when the migration will be resumed automatically.

2. Distributed parallel transfer can further speed up the migration. You can install COS Migration on multiple servers and perform migration tasks separately for different data sources.

# Migrating Data from Third-Party Cloud Storage Service to COS

Last updated : 2024-03-25 15:11:17

## Background

If you use a third-party cloud storage platform, COS can help you quickly migrate your data from that platform to COS.

Migration Method	Interactivity	Threshold for Large/Small Files	Concurrency	HTTPS Secure Transfer
<a href="#">Migration Service Platform (MSP)</a>	Visual operations	Default configuration	Same for all files	Enabled

This tool allows you to view the data migration progress, check file consistency, upload again after failure, and use checkpoint restart and other features, which can meet your basic migration requirements.

## Migration Practices

### MSP

MSP is a platform that integrates multiple migration tools and provides visual UIs to help you monitor and manage large-scale data migration tasks with ease. The File Migration Tool on it enables you to migrate data from various public clouds or data origins to COS.

The steps are as follows:

1. Log in to the [MSP console](#).
2. Click **Object Storage Migration** on the left sidebar.
3. Click **Create Task** to create a task and configure it as needed.
4. Start the task.

For more information, see the following documents:

[Migrating from Alibaba Cloud OSS](#)

Migrating from Huawei OBS

Migrating from Qiniu KODO

Migrating from UCloud UFile

Migrating from Kingsoft Cloud KS3

Migrating from Baidu AI Cloud BOS

[AWS S3 Migration Tutorial](#)**Tips**

During data migration, how fast the data source can be read depends on the network connection, and selecting a higher QPS concurrency value during file migration task creation helps speed up the migration.

# Migrating Data from URL to COS

Last updated : 2024-03-25 15:11:17

## Background

If you want to migrate data using a URL list as the data source address, COS supports the following three ways:

Migration Method	Description
Migration Service Platform (MSP)	MSP is a platform that integrates multiple migration tools and provides visual interfaces to help you monitor and manage large-scale data migration jobs with ease. The File Migration Tool on it can help you migrate data from various public clouds or data origin servers to COS.
<a href="#">COS Origin-Pull</a>	COS origin-pull automatically migrates data with read/write access requests from the data origin server to COS. This can help you separate hot data from cold data quickly and speed up reads/writes of hot data in your business system.
<a href="#">COS Migration</a>	COS Migration is an all-in-one tool integrating COS data migration features. You can migrate data to COS quickly after completing simple configurations.

In the process of migrating data from the origin server to the cloud, if you only want to migrate hot data while leaving the cold data in the origin server, you can use [COS origin-pull](#), which migrates the hot data accessed by read/write requests to COS and automatically separates the business data with low access frequency.

## Migration Practice

### Migration Service Platform (MSP)

The steps are as follows:

1. Log in to [MSP](#).
2. Click **Migration Tool** on the left sidebar, locate **File Migration Tool**, and click **Use Now**.
3. Create a migration task and configure task information.
4. Start the task.

For more information, please see [Migrating from a URL List](#).

### Tips

During data migration, how fast the data source can be read from depends on the network connection, and selecting a higher QPS concurrence value when creating a file migration task will help speed up the migration.

## COS origin-pull

The steps are as follows:

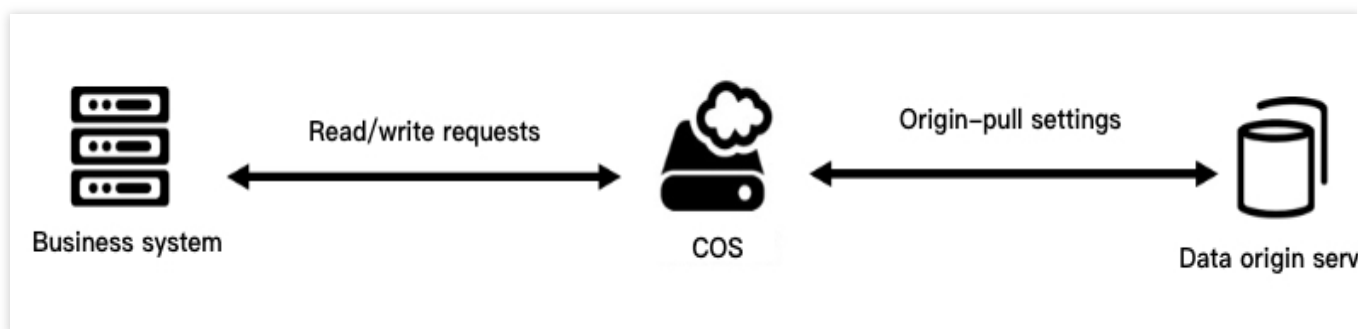
1. Log in to the COS Console and enable the origin-pull setting in the destination bucket.
2. Configure the bucket origin-pull address and save the configuration.
3. Redirect the read/write requests from your business system to COS.

For more information, please see [Setting Origin-Pull](#).

### Tips

The following steps demonstrate how to separate hot data and cold data. Hot data can be seamlessly migrated to COS for faster reads.

1. Redirect the read/write requests in your business system to COS, enable the origin-pull setting in the COS Console, and configure the data origin server as the source address. The system structure is as shown below.



2. After a period of time, the cold data remains in the origin server, while the hot data has been migrated to COS without any impact on your business system during migration.

## COS Migration

The steps are as follows:

1. Install the Java environment.
2. Install the COS Migration tools.
3. Modify the configuration file.
4. Launch the tool.

For more information, please see [COS Migration Tool](#).

### Tips

Below describes how to configure COS Migration to maximize the migration speed:

1. Adjust the threshold and concurrence of large/small files based on your network environment to implement optimal migration where large files are split into multiple parts and small files are transferred concurrently. Adjust the tool

execution time and set a bandwidth limit to ensure that bandwidth usage by data migration will not affect your business operations. Such adjustments can be made by modifying the following parameters in the `[common]` section of the `config.ini` configuration file:

Parameter Name	Parameter Description
<code>smallFileThreshold</code>	Threshold for small files. If the size of a file is higher than or equal to this threshold, multipart upload will be used; The default value is 5 MB.
<code>bigFileExecutorNum</code>	Concurrency of large files, which is 8 by default. If COS is connected to over the public network with low bandwidth, reduce this value.
<code>smallFileExecutorNum</code>	Concurrency of small files, which is 64 by default. If COS is connected to over the public network with low bandwidth, reduce this value.
<code>executeTimeWindow</code>	This parameter defines the time range when the migration tool will execute a task every day, which will enter sleep mode at other times. In sleep mode, the migration will be paused and the migration progress will be retained until the next time window when the migration will be resumed automatically.

2. Distributed parallel transfer can further speed up the migration. You can install COS Migration on multiple servers and perform migration tasks separately for different data sources.

# Migrating Data Within COS

Last updated : 2024-03-25 15:11:19

## Background

If you are using COS and need to migrate data in a bucket within COS, you are advised to use either of the following migration methods:

Online migration: [Migration Service Platform \(MSP\)](#)

Online migration: [Cross-bucket replication](#)

## Migration Practice

### MSP

MSP is a platform that integrates multiple migration tools and provides a visual interface to help you easily monitor and manage large-scale data migration tasks. The "File Migration Tool" can help you migrate data from various public clouds and data origin servers to COS.

The steps are as follows:

1. Log in to the [MSP console](#).
2. Click **Object Storage Migration** in the left sidebar.
3. Click **Create a Task** to create a task and configure the task as needed.
4. Start the task.

For more information, please see [Migrating Between Tencent Cloud COS](#).

During data migration, how fast the source data can be read depends on the network, and selecting a higher QPS concurrence when creating a file migration task will speed up the migration.

### Cross-Bucket replication

COS introduces the cross-bucket replication feature, with which you can set a rule to automatically and asynchronously replicate **incremental objects** in buckets residing in different regions. If cross-bucket replication is enabled, COS will precisely replicate object content (for example, object metadata and version ID) in the source bucket to the destination bucket, where the replica's attributes will be identical to those of the source. Moreover, operations on the source objects, for example, object upload and object deletion, will also be replicated to the destination bucket. For more information, please see [Cross-Bucket Replication](#).

# Migrating Data Between HDFS and COS

Last updated : 2024-03-25 15:11:17

## Overview

Hadoop DistCp (Distributed copy) is a tool used for large inter- and intra-cluster copying. It uses MapReduce to perform distribution, error handling, recovery, and reporting. With the parallel processing capabilities of MapReduce, Hadoop DistCp can quickly perform large-scale data migration through map tasks, each of which copies a portion of the files specified under the source path.

Since Hadoop-COS implements the semantics of the Hadoop Distributed File System (HDFS), Hadoop DistCp can help you easily migrate data between COS and HDFS. This document outlines this process below.

## Prerequisites

1. The [Hadoop-COS](#) plugin has been installed on the Hadoop cluster, and the access key for COS has been configured correctly. You can use the following Hadoop commands to check if COS access is normal:

```
hadoop fs -ls cosn://examplebucket-1250000000/
```

If the files in the COS bucket can be listed correctly, it means that Hadoop-COS has been installed and configured correctly, and the following steps can be performed.

2. The COS access account must have read and write permission for the destination path of the COS bucket.

### Note:

You can authorize sub-accounts read/write permissions for resources in the COS bucket as needed. You are advised to authorize by referring to [Notes on Principle of Least Privilege](#) and [Setting Sub-user Permissions](#). The common preset policies are as follows:

[DataFullControl](#): full control over data, including the permission to read, write, as well as delete files, and list the file list.

[QcloudCOSDataReadOnly](#): read-only permission

[QcloudCOSDataWriteOnly](#): write-only permission

The custom monitoring feature requires Cloud Monitoring to have permission to report metrics and read API operations. Please grant the [QcloudMonitorFullAccess](#) permission with caution.

## Directions



## Copying files from HDFS to a COS bucket

Use Hadoop DistCp to migrate files in the `/test` directory of the local HDFS cluster to the COS bucket `hdfs-test-1250000000`.

```
[iainyu@VM_38_97_centos ~]$ hadoop fs -ls -R /
drwxr-xr-x  - iainyu supergroup          0 2019-12-13 20:48 /test
-rw-r--r--  1 iainyu supergroup       167225 2019-12-13 20:47 /test/test-1
-rw-r--r--  1 iainyu supergroup       167225 2019-12-13 20:47 /test/test-2
-rw-r--r--  1 iainyu supergroup       167225 2019-12-13 20:47 /test/test-3
```

1. Run the following command to start the migration:

```
hadoop distcp hdfs://10.0.0.3:9000/test cosn://hdfs-test-1250000000/
```

Hadoop DistCp starts MapReduce tasks to copy the files, and outputs a brief report like so:

```

Merged Map outputs=0
GC time elapsed (ms)=0
Total committed heap usage (bytes)=253755392
File Input Format Counters
  Bytes Read=640
File Output Format Counters
  Bytes Written=8
DistCp Counters
  Bytes Copied=501675
  Bytes Expected=501675
  Files Copied=4
19/12/13 21:03:35 INFO mapred.LocalJobRunner: Finishing task: attempt_local1986740758_0001_m_000000_0
19/12/13 21:03:35 INFO mapred.LocalJobRunner: map task executor complete.
19/12/13 21:03:35 INFO mapred.CopyCommittee: Cleaning up temporary work folder: file:/tmp/hadoop-iainyu/mapred/staging/iainyu1750342510/.staging/_distcp-13859
19/12/13 21:03:35 INFO mapreduce.Job: map 100% reduce 0%
19/12/13 21:03:35 INFO mapreduce.Job: Job job_local1986740758_0001 completed successfully
19/12/13 21:03:35 INFO mapreduce.Job: Counters: 28
File System Counters
  COSN: Number of bytes read=0
  COSN: Number of bytes written=501675
  COSN: Number of read operations=0
  COSN: Number of large read operations=0
  COSN: Number of write operations=0
  FILE: Number of bytes read=155880
  FILE: Number of bytes written=537819
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=501675
  HDFS: Number of bytes written=0
  HDFS: Number of read operations=17
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=0
Map-Reduce Framework
  Map input records=4
  Map output records=0
  Input split bytes=161
  Spilled Records=0
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=0
  Total committed heap usage (bytes)=253755392
File Input Format Counters
  Bytes Read=640
File Output Format Counters
  Bytes Written=8
DistCp Counters
  Bytes Copied=501675
  Bytes Expected=501675
  Files Copied=4

```

2. Run the `hadoop fs -ls -R cosn://hdfs-test-1250000000/` command to list the directories and files that have just been migrated to the bucket `hdfs-test-1250000000`.

```
[iainyu@VM_38_97_centos ~]$ hadoop fs -ls -R cosn://hdfs-test-125 /
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
drwxrwxrwx - iainyu iainyu 0 1970-01-01 08:00 cosn://hdfs-test-125 /test
-rw-rw-rw- 1 iainyu iainyu 167225 2019-12-13 21:03 cosn://hdfs-test-125 /test/test-
-rw-rw-rw- 1 iainyu iainyu 167225 2019-12-13 21:03 cosn://hdfs-test-125 /test/test-
-rw-rw-rw- 1 iainyu iainyu 167225 2019-12-13 21:03 cosn://hdfs-test-125 /test/test-
```

## Copying files from a COS bucket to a local HDFS cluster

Hadoop DistCp is a tool that supports copying data between different clusters and file systems. To copy COS files to a local HDFS cluster, simply use the object path in the COS bucket as the source path and the HDFS file path as the destination path.

```
hadoop distcp cosn://hdfs-test-1250000000/test hdfs://10.0.0.3:9000/
```

## Using the DistCp command line to migrate data between HDFS and COS

### Note:

With this command line, you can migrate data from HDFS to COS, and vice versa.

Run the following command:

```
hadoop distcp -Dfs.cosn.impl=org.apache.hadoop.fs.CosFileSystem -Dfs.cosn.bucket.re
```

Parameter description:

Dfs.cosn.impl: Always set it to `org.apache.hadoop.fs.CosFileSystem`.

Dfs.cosn.bucket.region: bucket region. You can view the region on the bucket list page of the COS console.

Dfs.cosn.userinfo.secretId: Enter the `SecretId` of the bucket owner's account. The value can be obtained at [Manage API Key](#).

Dfs.cosn.userinfo.secretKey: Enter the `SecretKey` of the bucket owner's account. The value can be obtained at [Manage API Key](#).

libjars: specifies the location of the Hadoop-COS JAR package. You download the package from the `dep` directory at [GitHub repository](#).

### Note:

For other parameters, please see [Hadoop-COS](#).

## Additional Hadoop DistCp Parameters

Hadoop DistCp supports a variety of parameters. For example, you can use `-m` to specify the maximum number of concurrent Map tasks, and `-bandwidth` to limit the maximum bandwidth used by each map. For more

information, please see [Apache Hadoop DistCp Guide](#).

# Accessing COS with AWS S3 SDK

Last updated : 2025-01-24 13:07:58

## Overview

COS offers AWS S3-compatible APIs. Therefore, after your data is migrated from S3 to COS, you can make your client application conveniently compatible with the COS service by simply modifying the configurations. This document describes how to adapt the S3 SDK to different development platforms. After adaptation, you can use the APIs of S3 SDK to access files in COS.

### Preparations

You have signed up for a Tencent Cloud account as instructed in [Signing Up](#) and obtained the Tencent Cloud `SecretID` and `SecretKey` from the [CAM console](#).

You have a client application that has been integrated with the S3 SDK and runs properly.

## Android

The following describes how to adapt the AWS Android SDK 2.14.2 to COS. If COS is accessed from a device, a permanent key placed into the client code is at great risk of being leaked; therefore, we recommend you connect to the STS service to obtain a temporary key. For more information, see [Generating and Using Temporary Keys](#).

### Initialization

When initializing an instance, you need to set the temporary key provider and `Endpoint`. Suppose the bucket region is `ap-guangzhou`:

```
AmazonS3Client s3 = new AmazonS3Client(new AWSCredentialsProvider() {
    @Override
    public AWSCredentials getCredentials() {
        // Here, the backend requests for a temporary key from STS.
        return new BasicSessionCredentials(
            "<TempSecretID>", "<TempSecretKey>", "<STSSessionToken>"
        );
    }

    @Override
    public void refresh() {
        //
    }
});
```

```
s3.setEndpoint("cos.ap-guangzhou.myqcloud.com");
```

## iOS

The following describes how to adapt the AWS iOS SDK 2.10.2 to COS. If COS is accessed from a device, a permanent key placed into the client code is at great risk of being leaked; therefore, we recommend you connect to the STS service to obtain a temporary key. For more information, see [Generating and Using Temporary Keys](#).

### 1. Implement the AWS CredentialsProvider protocol

```
-(AWSTask<AWSCredentials *> *)credentials{
    // Here, the backend requests for a temporary key from STS.
    AWSCredentials *credential = [[AWSCredentials alloc] initWithAccessKey:@"<TempSecretID>"
    secretKey:@"<TempSecretKey>" sessionKey:@"<STSSessionToken>"
    expiration:[NSDate dateWithTimeIntervalSince1970:1565770577]];

    return [AWSTask taskWithResult:credential];
}

-(void)invalidateCachedTemporaryCredentials{
}
}
```

### 2. Provide a temporary key provider and Endpoint

Suppose the bucket region is `ap-guangzhou` :

```
NSURL* bucketURL = [NSURL URLWithString:@"https://cos.ap-guangzhou.myqcloud.com"];

AWSEndpoint* endpoint = [[AWSEndpoint alloc] initWithRegion:AWSRegionUnknown
service:AWSServiceS3 URL:bucketURL];
AWSServiceConfiguration* configuration = [[AWSServiceConfiguration alloc]
initWithRegion:AWSRegionUSEast2 endpoint:endpoint
credentialsProvider:[MyCredentialProvider new]]; // `MyCredentialProvider`
implements the `AWSCredentialsProvider` protocol.

[[AWSServiceManager defaultManager]
setDefaultServiceConfiguration:configuration];
```

# Node.js

The following describes how to adapt the AWS JS SDK 2.509.0 to COS.

## Initialization

When initializing an instance, set the Tencent Cloud key and `Endpoint` . Supposing the bucket region is `ap-guangzhou` , the sample code is as follows:

```
var AWS = require('aws-sdk');

AWS.config.update({
  accessKeyId: "COS_SECRETID",
  secretAccessKey: "COS_SECRETKEY",
  region: "ap-guangzhou",
  endpoint: 'https://cos.ap-guangzhou.myqcloud.com',
});

s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

# Java

The following describes how to adapt the AWS Java SDK 1.11.609 to COS.

## 1. Modify the configuration and certificate files of AWS

### Note :

Below is an example of modifying the configuration and certificate files of AWS on Linux.

The default configuration file of the AWS SDK is typically located under the user directory. For more information, see [Configuration and credential file settings](#).

Add the following configuration information to the configuration file (located in `~/.aws/config`):

```
[default]
s3 =
addressing_style = virtual
```

Configure the Tencent Cloud key in the certificate file (located in `~/.aws/credentials`):

```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

## 2. Set the Endpoint in the code

Supposing the bucket region is `ap-guangzhou`, the sample code is as follows:

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
    .withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(
        "http://cos.ap-guangzhou.myqcloud.com",
        "ap-guangzhou"))
    .build();
```

If you are using version 2 of the AWS Java SDK, the code example is as follows:

```
S3Client s3Client = S3Client.builder()
    .endpointOverride(URI.create("http://cos.ap-guangzhou.myqcloud.com"))
    .region(Region.of("ap-guangzhou"))
    .build();
```

## Python

The following describes how to adapt the AWS Python SDK 1.9.205 to COS.

### 1. Modify the configuration and certificate files of AWS

#### Note :

Below is an example of modifying the configuration and certificate files of AWS on Linux.

The default configuration file of the AWS SDK is typically located under the user directory. For more information, see [Configuration and credential file settings](#).

Add the following configuration information to the configuration file (located in `~/.aws/config`):

```
[default]
s3 =
    signature_version = s3
    addressing_style = virtual
```

Configure the Tencent Cloud key in the certificate file (located in `~/.aws/credentials`):

```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

### 2. Set the `Endpoint` in the code

Suppose the bucket region is `ap-guangzhou`:

```
client = boto3.client('s3', endpoint_url='https://cos.ap-guangzhou.myqcloud.com')
```

# PHP

The following describes how to adapt the AWS PHP SDK 3.109.3 to COS.

## 1. Modify the configuration and certificate files of AWS

### Note :

Below is an example of modifying the configuration and certificate files of AWS on Linux.

The default configuration file of the AWS SDK is typically located under the user directory. For more information, see [Configuration and credential file settings](#).

Add the following configuration information to the configuration file (located in `~/.aws/config`):

```
[default]
s3 =
addressing_style = virtual
```

Configure the Tencent Cloud key in the certificate file (located in `~/.aws/credentials`):

```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

## 2. Set the `Endpoint` in the code

Suppose the bucket region is `ap-guangzhou` :

```
$S3Client = new S3Client([
    'region'          => 'ap-guangzhou',
    'version'         => '2006-03-01',
    'endpoint'        => 'https://cos.ap-guangzhou.myqcloud.com'
]);
```

# .NET

The following describes how to adapt the AWS .NET SDK 3.3.104.12 to COS.

### Initialization

When initializing an instance, set the Tencent Cloud key and `Endpoint` . Supposing the bucket region is `ap-guangzhou` , the sample code is as follows:



```
string sAccessKeyId = "COS_SECRETID";
string sAccessKeySecret = "COS_SECRETKEY";
string region = "ap-guangzhou";

var config = new AmazonS3Config() { ServiceURL = "https://cos." + region +
".myqcloud.com" };
var client = new AmazonS3Client(sAccessKeyId, sAccessKeySecret, config);
```

## Go

### aws-sdk-go

The following describes how to adapt the AWS Go SDK 1.21.9 to COS.

#### 1. Create a session based on the key

Suppose the bucket region is `ap-guangzhou` :

```
func newSession() (*session.Session, error) {
    creds := credentials.NewStaticCredentials("COS_SECRETID", "COS_SECRETKEY", "")
    region := "ap-guangzhou"
    endpoint := "http://cos.ap-guangzhou.myqcloud.com"
    config := &aws.Config{
        Region:           aws.String(region),
        Endpoint:         &endpoint,
        S3ForcePathStyle: aws.Bool(true),
        Credentials:      creds,
        // DisableSSL:     &disableSSL,
    }
    return session.NewSession(config)
}
```

#### 2. Create a server initiation request based on the session

```
sess, _ := newSession()
service := s3.New(sess)

// Take file upload as an example.
fp, _ := os.Open("yourLocalFilePath")
defer fp.Close()

ctx, cancel := context.WithTimeout(context.Background(), time.Duration(30)*time.Second)
defer cancel()
```

```
service.PutObjectWithContext(ctx, &s3.PutObjectInput{
    Bucket: aws.String("examplebucket-1250000000"),
    Key:    aws.String("exampleobject"),
    Body:   fp,
})
```

## aws-sdk-go-v2

The following describes how to adapt the aws-sdk-go-v2 to upload objects to COS.

```
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "os"
    "strings"
)

func main() {
    // Get the key through environment variables SECRETID and SECRETKEY
    creds := credentials.NewStaticCredentialsProvider(os.Getenv("SECRETID"), os
    customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region
        return aws.Endpoint{
            PartitionID: "aws",
            URL:         "http://cos.ap-guangzhou.myqcloud.com",
            SigningRegion: "ap-guangzhou",
        }, nil
    })

    cfg, _ := config.LoadDefaultConfig(
        context.TODO(),

        config.WithCredentialsProvider(creds),
        config.WithEndpointResolverWithOptions(customResolver))

    s3Client := s3.NewFromConfig(cfg, func(o *s3.Options) {
        o.UsePathStyle = false // Access using virtual-host style
    })

    input := &s3.PutObjectInput{
```

```
        Body:          strings.NewReader("xxxxxxx"),
        Bucket:        aws.String("test-1250000000"), //Bucket name
        Key:           aws.String("test"),           //Object key
        StorageClass: "STANDARD",
    }

    result, err := s3Client.PutObject(context.Background(), input)
    if err != nil {
        panic(err)
    }
    fmt.Println(result)
```

## C++

The following describes how to adapt the AWS C++ SDK 1.7.68 to COS.

### 1. Modify the configuration and certificate files of AWS

#### Note :

Below is an example of modifying the configuration and certificate files of AWS on Linux.

The default configuration file of the AWS SDK is typically located under the user directory. For more information, see [Configuration and credential file settings](#).

Add the following configuration information to the configuration file (located in `~/.aws/config`):

```
[default]
s3 =
addressing_style = virtual
```

Configure the Tencent Cloud key in the certificate file (located in `~/.aws/credentials`):

```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

### 2. Set the Endpoint in the code

Supposing the bucket region is `ap-guangzhou`, the sample code is as follows:

```
Aws::Client::ClientConfiguration awsCC;
awsCC.scheme = Aws::Http::Scheme::HTTP;
awsCC.region = "ap-guangzhou";
awsCC.endpointOverride = "cos.ap-guangzhou.myqcloud.com";
Aws::S3::S3Client s3_client(awsCC);
```



# Data Disaster Recovery and Backup Disaster Recovery and High Availability Architecture Based on Cross-Bucket Replication

Last updated : 2024-03-25 15:11:17

## Overview

Tencent Cloud Object Storage (COS) offers a service availability of 99.95% and reliability of 99.999999999%. Due to uncontrollable factors such as natural disasters and fiber-optic cable failures, neither the availability nor the reliability can reach 100% for in-cloud data; however, extremely high availability and reliability are required for certain businesses like finance.

Given this background, COS provides a high-availability disaster recovery solution based on cross-bucket replication. When using COS, you are advised to make disaster recovery plans and backups for your in-cloud data based on your actual needs to keep your business uninterrupted.

This document describes a COS backup and disaster recovery solution (i.e., master/slave switch for cloud-based businesses) as well as a COS high-availability solution based on cross-bucket replication. Different COS products and features such as cross-bucket replication, origin-pull, SCF, and CDN achieve high availability.

## Backup and Disaster Recovery Solution Based on Cross-Bucket Replication

Disaster recovery entails three elements: redundancy, remote, and replication.

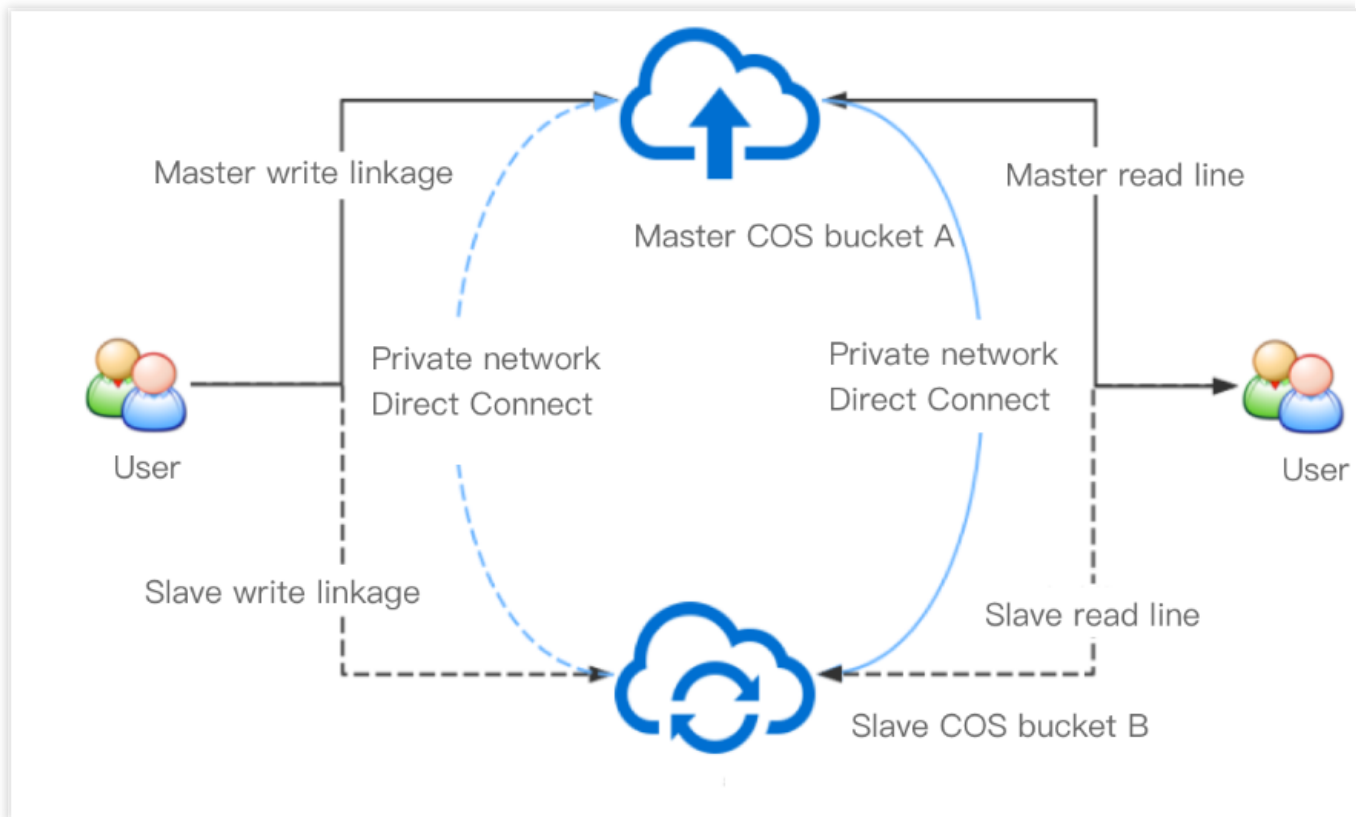
**Redundancy:** data should be backed up simultaneously to another available system.

**Remote:** data backups should be stored in another remote region, as disasters often extend geographically and only a long enough distance can guarantee the availability of redundant data.

**Replication:** data loss during backup should be reduced down to zero.

COS cross-bucket replication enables cross-bucket syncing of incremental data. Data uploaded to a bucket can be replicated to another bucket in seconds or minutes, depending on file size and distance. Cross-bucket replication allows you to make remote redundant backups of your data for disaster recovery and business continuity. For more information, please see [Cross-Bucket Replication Overview](#). To enable this feature, you need to enable versioning first. For more information on versioning, please see [Versioning Overview](#).

The schematic diagram of the backup and disaster recovery architecture based on cross-bucket replication is as shown below:



Under this architecture, your bucket A and bucket B mutually back up each other. If your data is stored in bucket A, then bucket B is the backup bucket. In order to ensure business continuity and stability, you have configured cross-bucket replication rules for bucket A and bucket B respectively. According to the rules, incremental data in bucket A will be automatically replicated to bucket B, and vice versa.

**Note:**

After the incremental data in bucket A is replicated to bucket B, although it is "incremental" in bucket B, it will not be replicated to bucket A.

Normally, all your read/write requests point to bucket A where all incremental data will be automatically replicated to bucket B as backups. You can add a network quality detection module to your upload or download program at the business side, allowing you to quickly switch to bucket B when a failure is detected in bucket A.

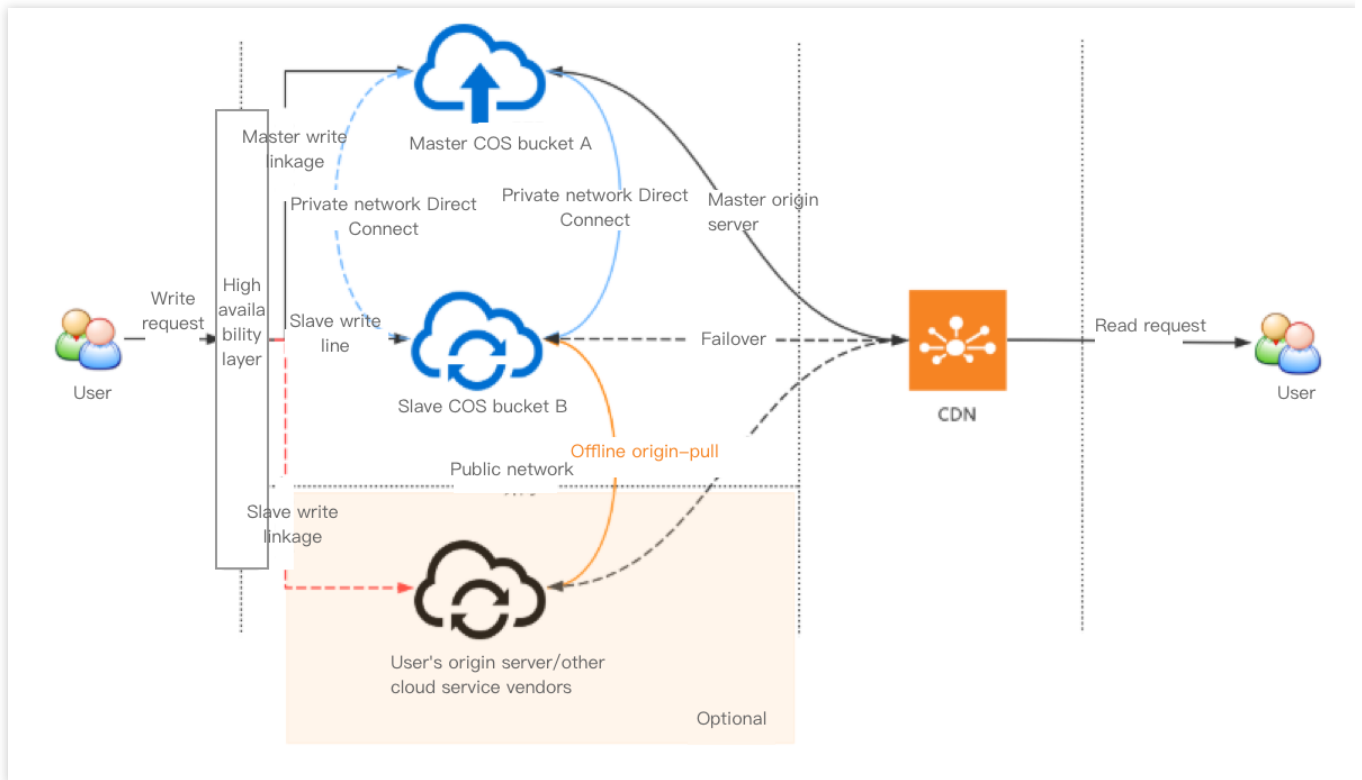
**Note:**

Network quality can be tested based on Serverless Cloud Function (SCF) by changing the automated testing addresses to the domain names of master and slave buckets, and modifying the alarm code snippets as needed.

## High-Availability Solution Based on Cross-Bucket Replication

Despite all the benefits, the aforementioned solution may not always be able to guarantee high availability due to the complex, ever-changing real businesses. This section proposes a high-availability solution based on cross-bucket replication and used with different COS products and features such as origin-pull, SCF, and CDN.

The schematic diagram of the high availability architecture based on cross-bucket replication is as shown below:



This architecture consists of the following layers:

**High availability layer:** integrates network detection and service scheduling and switches links based on metrics such as link connectivity, which can be implemented with the aid of SCF (as described in the previous section) or on the client side according to your business needs.

**Storage layer:** typically consists of COS buckets in different regions. You can also **introduce buckets from external origin servers or other cloud vendors** by [setting origin-pull policies](#) so as to further guarantee data consistency.

**CDN layer:** provides the nearest access through a massive number of edge servers in Tencent Cloud CDN, eliminating the need to directly access data on origin servers and thus ensuring data security.

The following explains how this architecture guarantees high availability:

1. Normally, all your write requests point to bucket A where all incremental data will be automatically replicated to bucket B as backups.
2. When the links to bucket A fail (for example, the quality of automated testing declines or an upload fails), the client can switch the links to bucket B. In this case, all incremental data in bucket B will also be automatically replicated to bucket A.

3. You can also choose to make a redundant backup of your data on an external origin server or in another cloud first and then configure an origin-pull policy for bucket B. If, in extreme cases, the links to both buckets A and B fail, bucket B can pull data from the origin server when the attempt to upload data to bucket B fails.

**Note:**

As full redundant backups are costly, you can choose to make redundant backups of only hot data (such as files uploaded in just a few hours) so as to reduce data storage costs.

If you choose an origin server as part of the high availability architecture, please be sure to assess the bandwidth of the origin server and the possible impact of the limit on it when designing the architecture.

4. You can read data from your bucket by directly accessing it or by [binding a CDN acceleration domain name](#) to your bucket, the latter of which enables nearest access through the edge servers in Tencent Cloud CDN. If your business data involves content delivery, or you don't want your end users to directly access your bucket, you are advised to use [Tencent Cloud CDN](#).

**Note:**

If you want to read data from your bucket directly, your client should be able to follow 302 redirects in the HTTP protocol.

Tencent Cloud CDN boasts nearly a thousand edge servers which provide adjacent access nodes to increase the data read speed. You can bind multiple origin servers to CDN as master and slave servers in order to ensure high availability. For more information, please see [Origin Server Configuration](#).

If you want to secure your origin servers as much as possible, you can set private-read/write permission for them and enable CDN origin-pull authentication so as to allow your end users to anonymously access the data cached on the CDN edge servers while protecting the security of the data on the origin servers.

## References

The following documents can help you easily implement the high-availability disaster recovery architecture:

[Versioning Overview](#)

[Cross-Bucket Replication Overview](#)

[Setting Origin-Pull](#)

[Setting CDN Acceleration](#)

[Origin Server Configuration](#)

[Adding Domain Names](#)



# Cloud Data Backup

Last updated : 2024-03-25 15:11:19

## Overview

This document describes how to back up data stored in the cloud. COS provides the following three methods to facilitate backup of data stored in COS:

Backup and disaster recovery scheme based on cross-region replication

Batch data replication scheme (COS batch operation feature)

Migration and backup scheme based on Migration Service Platform (MSP)

## Backup and Disaster Recovery Scheme Based on Cross-region Replication

With the cross-region replication feature, COS can automatically and asynchronously replicate **new objects** added to one bucket to another bucket in a different region. When cross-region replication is enabled, COS will accurately replicate the object content (such as object metadata and version ID) in the source bucket to the destination bucket, and the object copies contain exactly the same attribute information.

For more information, please see [High-Availability Disaster Recovery Architecture Based on Cross-region Replication](#).

### Use cases

**Remote disaster recovery:** COS has 11 nines of availability for object data, but there is still a slight chance of data loss due to force majeure such as wars and natural disasters. If you want to avoid data loss by explicitly having a separate copy in a different region, you can use cross-region replication to achieve remote disaster recovery. In this way, when the IDC in one region is damaged due to force majeure, the IDC in the other region can still provide data copies for your use.

**Compliance:** COS ensures data availability by providing multiple copies and erasure codes for data in physical disks by default. However, there may be compliance requirements in some industries stipulating that you keep copies in another region. Cross-region replication allows data to be replicated across regions to meet such requirements.

**Access latency reduction:** when you have end users accessing objects from different regions, with cross-region replication, you can maintain object copies in the available storage regions closest to them, which can minimize access latency to deliver a better user experience.

**Special technical requirements:** if you have computing clusters in two different regions and the clusters need to process the same set of data, with cross-region replication, you can maintain object copies in both regions.

**Data migration and backup:** you can copy your business data from one availability region to another one as needed to implement data migration and backup.

### Usage

For more information, please see [Setting Cross-Region Replication](#).

## Batch Data Replication Scheme

The COS batch operation feature enables you to perform large-scale batch replication operations at the object level.

### Use cases

For some business reasons, you may need to back up all data in an existing bucket to another bucket to ensure data availability and reliability.

### Usage

For more information, please see [Batch Operation](#).

### Instructions

The batch replication operation should be used in conjunction with the inventory feature. For more information on inventory, please see [Inventory Overview](#). This operation allows you to replicate the specified objects in batches from the source bucket to the destination bucket in the same region or in different regions. It supports customizing parameters for the `PUT Object-copy` operation, and the metadata and storage class of the copy are subject to the configuration information. For more information, please see [PUT Object - copy](#).

### Note:

All objects to be replicated must be in the same bucket.

Only one destination bucket can be configured for a batch replication job.

You need to have permission to read objects from the source bucket and write objects into the destination bucket.

The total size of the objects cannot exceed 5 GB.

Verification via ETags and server-side encryption using custom keys are not supported.

If the destination bucket does not have versioning enabled and contains an object file with the same name as a file to be replicated, COS will overwrite the object file.

## Data Migration and Backup Scheme

Tencent Cloud Migration Service Platform (MSP) provides easy and fast resource migration schemes. After you create a migration job, you can view the migration progress, migration report, and much more in the MSP Console. For detailed directions, please see [COS Migration](#).

# Local Data Backup

Last updated : 2024-03-25 15:11:17

## Overview

This document describes how to back up local data to the cloud. COS provides the following three methods to facilitate backup of local data to a COS bucket:

Backup based on file sync through COSBrowser

Backup based on online migration through COS Migration

Backup based on offline migration through CDM

## Backup Based on File Sync

COSBrowser is a visual cloud file manager launched by COS that allows you to easily view, transfer, and manage COS resources through simple interactions. Currently, COSBrowser is available in Desktop Edition and Mobile Edition. For more information, please see [COSBrowser Overview](#).

COSBrowser Desktop Edition has a file sync feature that can be used to sync and upload local files to the cloud by associating local folders with buckets.

Sync Setting Sync Logs

---

Local Folder \*

Change

Bucket Path \*

Auto Sync

---

Support incremental upload only. See the [documentation](#) for details.

## Usage

For more information, please see [COSBrowser Instructions](#).

## Online Migration Scheme

COS Migration is an all-in-one tool integrating COS data migration features. You can migrate data to COS quickly after completing simple configurations.

### Use cases

If you have a local IDC, COS Migration can help you migrate massive amounts of local data to COS.

## Usage

For more information, please see [COS Migration](#).

## Offline Migration Scheme

CDM uses a dedicated offline migration device provided by Tencent Cloud to help you migrate your local data to the cloud, solving the problems that arise from online migration from a local IDC to cloud, such as long time, high costs, and low security.

You can determine how to migrate data based on the data volume, IDC egress bandwidth, IDC idle server resources, acceptable completion time, and other factors. The estimated time needed for online migration is shown in the figure below. As can be seen, if data migration needs to take more than 10 days or the amount of data to be migrated exceeds 50 TB, you are recommended to use [CDM](#) for offline migration.

Data Volume	Bandwidth			
	10Mbps	100Mbps	1Gbps	10Gbps
10GB	3 hours	17 minutes	2 minutes	10 seconds
100GB	30 hours	3 hours	17 minutes	2 minutes
1TB	12.5 days	30 hours	3 hours	17 minutes
10TB	125 days	12.5 days	30 hours	3 hours
100TB	3.5 years	125 days	12.5 days	30 hours
1PB	35 years	3.5 years	125 days	12.5 days
10PB	350 years	35 years	3.5 years	125 days

### Usage

For more information, please see [CDM](#)

# Domain Name Management Practice

## Switch bucket to custom domain

Last updated : 2024-07-05 18:50:41

### Background

To ensure overall service security and stability, for buckets created after January 1, 2024, if the default domain of Cloud Object Storage (COS) is used to access objects, preview of any type of file and download of apk/ipa type files are not supported. For details, see [Implementation Notice on Security Management of COS Bucket Domain](#).

For buckets created after January 1, 2024, if you want to preview files directly or download apk/ipa type objects within the bucket through a browser, it is recommended to use a custom domain to access objects. Buckets created before January 1, 2024 are not affected by this change when the default domain is used for preview and download. However, for better service stability, it is recommended to prioritize using a custom domain.

This document describes how to configure a custom domain for a bucket, switching from accessing the bucket's default domain to accessing a custom domain.

### Step 1: Registering and Filing a Domain

First, you need to prepare a custom domain that has been filed.

Domain registration: If you do not have a custom domain, you can purchase one at [Domains](#).

Domain filing: If your custom domain is to be configured for a bucket in the Chinese mainland region, it must be filed.

### Step 2: Configuring a Custom Domain for the Bucket

1. After preparing the custom domain, log in to the [COS console](#), go to the bucket list, and select the bucket you need to configure.

2. Enter the bucket details page, and choose **Domain Name and Transport Management > Custom Origin Server Domain**.

3. Click **Add Domain**, and configure the domain information:

Domain: Enter the prepared custom domain.

Origin Server Type: The following types are available.

**Default Origin Server:** If you want to use the custom domain as the default origin server, select Default Origin Server.

**Static Website Origin Server:** If you want to use the custom domain for a static website, first enable the static website feature for the bucket, and then select Static Website Origin Server.

**Global Acceleration Origin Server:** If you want to use the custom domain for global acceleration, first enable the global acceleration feature for the bucket, and then select Global Acceleration Origin Server.

4. Configure an HTTPS certificate. If you want to access using the HTTPS protocol, you need to configure a certificate for the custom domain.

If you need to use your own certificate, paste the certificate content and private key content into the specified input boxes.

If you are using a certificate from Tencent Cloud, you can directly select an existing Tencent Cloud certificate under the current account in the pop-up window.

5. Upon completing the custom domain configuration, record the CNAME information (e.g., bucket-1250000000.cos.ap-beijing.myqcloud.com) for subsequent domain name resolution configuration.

## Step 3: Configuring Domain Name Resolution

### Tencent Cloud Domain

If your domain DNS provider is Tencent Cloud, go to the [DNS console](#) to configure the CNAME resolution record.

1. Go to the [DNS console](#), find the corresponding domain, and click the **Resolution** button.

2. Click **Quick Add Resolution** to add a resolution record for the domain.

3. In the pop-up window, select **Website Resolution**, and choose **Domain Name Mapping (CNAME)** for the website address. Enter the CNAME information recorded in Step 2, for example, bucket-1250000000.cos.ap-beijing.myqcloud.com.

4. It takes some time for the resolution record to take effect. You can use the dig command or check in the [COS console](#) to see whether the resolution is successfully applied. The verification methods are as follows:

Enter the command `dig mydomain.com` in the command line window, and check whether the CNAME record is correctly applied. (Replace `mydomain.com` with your actual domain when using it.)

```

-MB0 ~ % dig test .com
dig mydomain.com

; <<>> DiG 9.10.6 <<>> test .com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45215
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4000
;; QUESTION SECTION:
;test .com. IN      A

; ANSWER SECTION:
test .com. 599 IN  CNAME  .cos.ap-beijing.myqcloud.com

```

Log in to the [COS console](#) and check the bucket's custom domain. If the domain's CNAME is not successfully applied, a corresponding prompt will appear.

## Other Vendors' Domains

If your domain DNS provider is not Tencent Cloud, you need to go to the corresponding DNS service to configure the CNAME resolution record.

## Step 4: Accessing Your Custom Domain

After the above steps are completed, the configuration of the custom domain is finished. Below is an explanation of how to use the custom domain to access COS.

### Viewing the Object Access Link

1. Log in to the [COS console](#), find the bucket with the configured custom domain, and click to enter the **File List**. Select an object to enter the object details. For operation instructions, see [Viewing Object Information](#).
2. Switch the designated domain to the **custom origin server domain name**. The object address and temporary link below will be correspondingly switched to the custom domain link. To access public read objects, you can use the object address (without signature). To access private read objects, you can use the temporary link (with signature).

### Switching to the Custom Domain for API Access

For users who access COS directly via API, simply change the request Host to the custom domain when accessing.

```

GET /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com      # Replace with the user
Date: GMT Date

```



```
Authorization: Auth String
```

## Switching to the Custom Domain for SDK Access

For users using the SDK, simply set the domain parameter to the custom domain when initializing the client. Taking the Python SDK as an example, the code example is as follows.

```
domain = 'user-define.example.com' # User's custom domain
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

For code examples of switching to a custom domain in COS SDKs of various languages, refer to the following documentation:

[Go SDK](#)

[Java SDK](#)

[Python SDK](#)

[PHP SDK](#)

[Android SDK](#)

[iOS SDK](#)

[JS SDK](#)

[Node.js SDK](#)

[.NET SDK](#)

[C++ SDK](#)

[C Language SDK](#)

[Mini Program SDK](#)

# Supporting HTTPS for Custom Endpoints

Last updated : 2024-03-25 15:11:17

## Overview

You can access the objects under a bucket using your own endpoint (the custom endpoint, for example, `test.cos.com` ). Detailed directions are as follows:

[Supporting HTTPS for a custom endpoint with CDN acceleration enabled](#)

[Supporting HTTPS for a custom endpoint with CDN acceleration disabled](#)

## Directions

### Enabling CDN Acceleration

#### Step 1. Bind a custom domain name

Bind the bucket to your own endpoint and enable CDN acceleration. For detailed directions, please see [Enabling Custom Accelerated Domain Name](#).

#### Step 2. Perform HTTPS configuration

You can configure HTTPS access in the [CDN console](#). For detailed directions, please see [HTTPS Configuration Guide](#).

### Disabling CDN Acceleration

This section uses an example to describe how to support HTTPS access in COS by configuring custom endpoints through a reverse proxy (with CDN acceleration disabled). In this example, we use the custom endpoint

`https://test.cos.com` to directly access the `testhttps-1250000000` bucket in the Guangzhou region with CDN acceleration disabled. The specific steps are as follows:

#### Step 1. Bind a custom domain name

HTTPS certificate hosting for custom origin server domain names of COS is supported in public cloud regions in the Chinese mainland and in Singapore. You can bind the certificate to the added custom origin server domain names via the console. For details, see [Method 1](#). If no HTTPS certificate is available for your domain name, click [Apply for Free Certificate](#).

This feature is currently not supported in other regions. To use an HTTPS certificate, see [Method 2](#).

#### Method 1

: Bind a custom origin server domain name via the COS console

Bind the `testhttps-1250000000` bucket to the `https://test.cos.com` domain and disable CDN acceleration. For detailed directions, please see [Enabling Custom Accelerated Domain Name](#).

Method 2:

Configure a reverse proxy for the domain name

Configure a reverse proxy for the `https://test.cos.com` endpoint on the server, as shown below (the Nginx configuration is for reference only):

```
server {
    listen          443;
    server_name    test.cos.com ;

    ssl on;
    ssl_certificate /usr/local/nginx/conf/server.crt;
    ssl_certificate_key /usr/local/nginx/conf/server.key;

    error_log logs/test.cos.com.error_log;
    access_log logs/test.cos.com.access_log;
    location / {
        root /data/www/;
        proxy_pass http://testhttps-1250000000.cos.ap-guangzhou.myqcloud.com; // Con
    }
}
```

`Server.crt;` and `server.key` are HTTPS certificates for your own (custom) domain. If no HTTPS certificate is available for your domain, you can apply for one at [Tencent Cloud SSL Certificate Service](#).

If no certificate is available, the following configuration information can be deleted, but an alarm will occur during access. Click Continue to access the bucket:

```
ssl on;
ssl_certificate /usr/local/nginx/conf/server.crt;
ssl_certificate_key /usr/local/nginx/conf/server.key;
```

## Step 2. Resolve the domain name at a server

Resolve your endpoint at your endpoint's DNS provider.

## Step 3. Perform advanced configurations

### Opening the web page in a browser directly

After configuring the custom endpoint to support HTTPS, you can download objects in the bucket using your domain. If your business requires directly accessing web pages and images in a browser, you can use the static website feature. For detailed directions, please see [Setting Up a Static Website](#).

After the configuration is completed, add the following code to the Nginx configuration file, restart Nginx, and refresh the browser cache.

```
proxy_set_header Host $http_host;
```

### Configuring referer hotlink protection

Public buckets might be hotlinked. You can use the hotlink protection feature to set a referer allowlist to prevent malicious hotlinking as follows:

1.1 Log in to the [COS console](#), enable the hotlink protection feature, and configure an allowlist. For detailed directions, please see [Setting Hotlink Protection](#).

1.2 Add the following code to the Nginx configuration file, restart Nginx, and refresh the browser cache.

```
proxy_set_header Referer www.test.com;
```

1.3 After the configuration, if you open the file directly, the error `errorcode: -46616` (error message: `not hit white refer`) will be reported. In this case, you can access the custom endpoint with a proxy to open the page.

```
{
  errorcode: -46616,
  errormsg: "not hit white refer, retcode:-46616"
}
```

# Setting CORS

Last updated : 2024-03-25 15:11:17

## One-Origin Policy

The one-origin policy is a key security mechanism for isolating potentially malicious files. It restricts the way files/scripts loaded from one origin interacts with resources from another origin. Resources with the same protocol, domain name (or IP), and port are considered to belong to the same origin. Scripts on one origin only have permissions to read/write resources on the origin but cannot access resources on other origins.

### Definition of one-origin resources

Webpages from a single origin should have the same protocol, domain name, and port (if specified). The following table shows how to test whether a webpage belongs to the same origin as

`http://www.example.com/dir/page.html` :

URL	Result	Reason
<code>http://www.example.com/dir2/other.html</code>	Yes	Same protocol, domain name, and port
<code>http://www.example.com/dir/inner/another.html</code>	Yes	Same protocol, domain name, and port
<code>https://www.example.com/secure.html</code>	No	Different protocols (HTTPS)
<code>http://www.example.com:81/dir/etc.html</code>	No	Different ports (81)
<code>http://news.example.com/dir/other.html</code>	No	Different domain names

## CORS

Cross-Origin Resource Sharing (CORS) is also known as cross-origin access. It allows web application servers to perform cross-origin access control to ensure secure cross-origin data transfer. Both the browser and server need to support this feature before you can use it. The feature is compatible with all browsers (for IE, IE 10 or later is required). The CORS communication process is automatically completed by the browser without any manual intervention required. For developers, CORS communication and one-origin AJAX communication work in the same way and use the same code. Once the browser identifies an AJAX request for cross-origin access, it automatically adds additional header information. In some cases, an additional request is made, but you will not perceive it.

Therefore, the key to CORS communication lies in the server. As long as the server implements CORS APIs, cross-origin communication can be implemented.

## CORS Use Cases

CORS is used when you are using a browser. This is because access permissions are controlled by the browser but not the server. Therefore, if you use other clients, you don't need to care about cross-origin access.

With CORS, you can use AJAX in a browser to directly access, upload, and download COS data without using your app server for data transfer. If your website adopts both COS and AJAX technologies, we recommend you use CORS for direct communication with COS.

## COS Support for CORS

COS supports configuring CORS rules to allow or deny cross-origin requests as needed. CORS rules are configured at the bucket level.

COS authentication and whether a CORS request is allowed are independent of each other. In other words, CORS rules of COS are only used to decide whether to add CORS-related headers. It is up to the browser whether to block the request.

All object and multipart APIs of COS support CORS authentication.

### Note:

When two webpages ( `www.a.com` and `www.b.com` ) running in the same browser request the same cross-origin resource at the same time, if the request from `www.a.com` arrives at the server first, the server will return the resource to the user of `www.a.com` with the `Access-Control-Allow-Origin` header. If `www.b.com` sends a request later, the browser will return the cached response of the last request to the user. In this case, the header content does not match the CORS-required content, so the `www.b.com` request will fail.

## CORS Configuration Example

The following example shows how to configure CORS to get data from COS by using AJAX. The bucket permission used in the example is set to public. For a bucket with private access permission, a signature needs to be added in the request, while other configurations are the same.

The bucket used in the following example is named `corstest` , with the access permission of public read/private write.

### Preparations

#### 1. Verify whether the file can be accessed.

Upload the `test.txt` file to `corstest` . The access address of this file is `http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt` .

Access the text file by using curl, with the following address replaced with your file address:

```
curl http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt
```

If "test" (the file content) is returned, the file can be accessed normally.

```
[root@VM_139_240_centos ~]# curl http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt
test
```

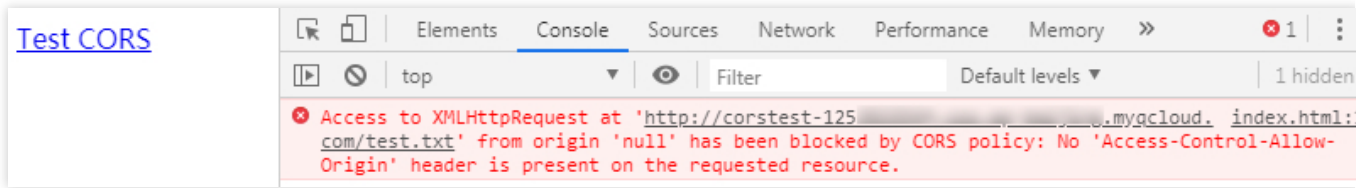
## 2. Access the file by using AJAX

You can access the `test.txt` file directly by using AJAX.

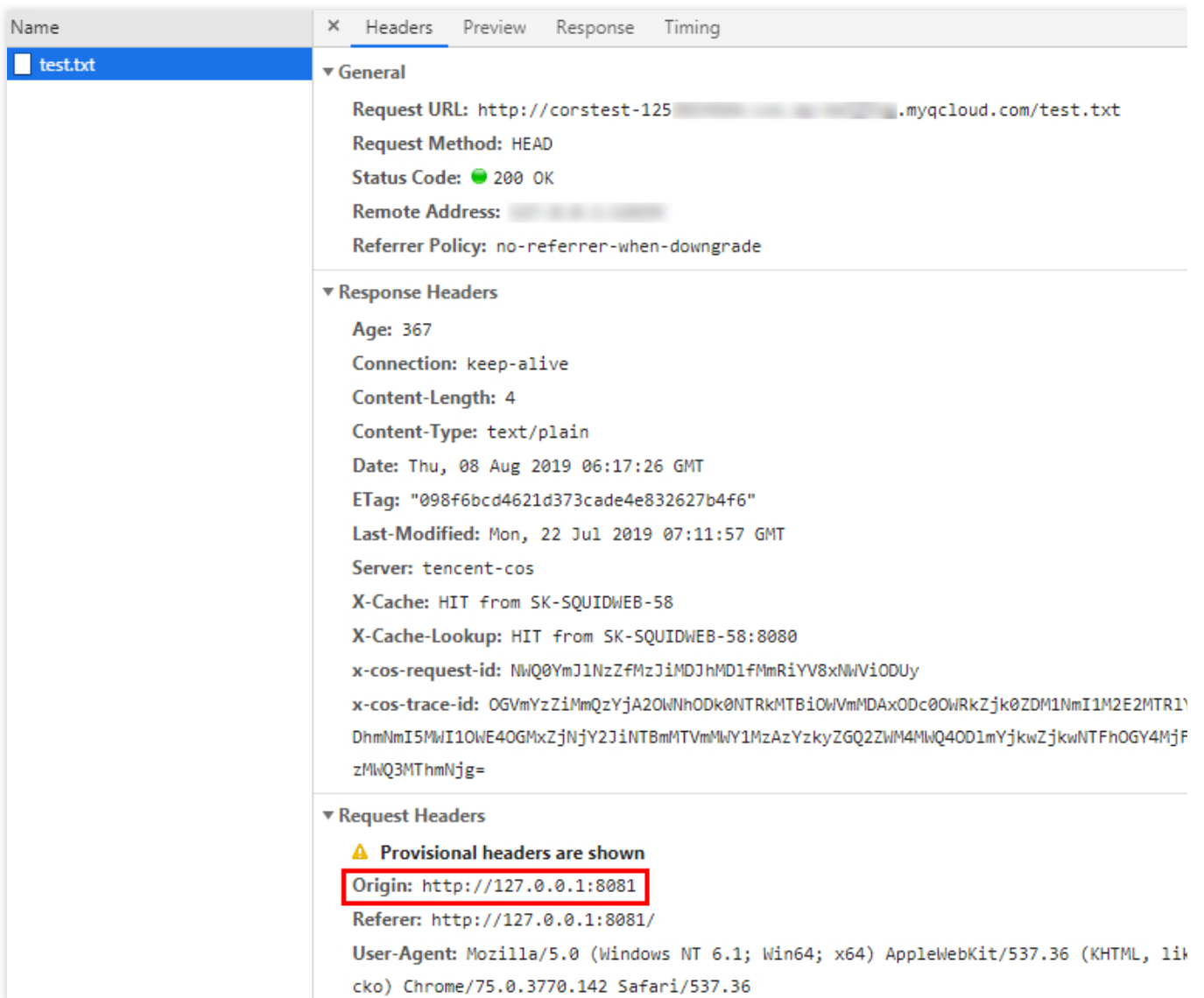
(1) Copy the following code to a local HTML file and then open it with a browser. As no custom header is set, no preflight request is required.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
</head>
<body>
<a href="javascript:test()">Test CORS</a>
<script>
  function test() {
    var url = 'http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt';
    var xhr = new XMLHttpRequest();
    xhr.open('HEAD', url);
    xhr.onload = function () {
      var headers = xhr.getAllResponseHeaders().replace(/\r\n/g, '\n');
      alert('request success, CORS allow.\n' +
        'url: ' + url + '\n' +
        'status: ' + xhr.status + '\n' +
        'headers:\n' + headers);
    };
    xhr.onerror = function () {
      alert('request error, maybe CORS error.');
```

(2) Open the HTML file in the browser and click **Test CORS** to send the request. The following error occurs with the message "Access denied. No Access-Control-Allow-Origin header is found". This is because CORS has not been configured on the server.



(3) When the access fails, go to the **Headers** page to find out the cause. You can see that the browser sent a request with **Origin** specified, meaning it is a cross-origin request.



#### Note:

The webpage is set up on the server with the address `http://127.0.0.1:8081`. Therefore, the **Origin** is `http://127.0.0.1:8081`.



## Configuring CORS

Now that you have identified the cause of the access failure, you can solve the problem by configuring CORS for the bucket. This example configures CORS in the COS console as follows, which is recommended for easy configurations:

1. Log in to the [COS console](#), click **Bucket List**, and click the name of the target bucket. Then, select the **Security Management** tab and find **CORS (Cross-Origin-Resource Sharing)** in the drop-down list.
2. Click **Add a Rule** to add the first rule with the following least restricted configuration:

**Add rules** ×

Origin \*

A line can contain at most one \* wildcard character

Allow-Methods \*  PUT  GET  POST  DELETE  HEAD

Allow-Headers

Expose-Headers

Max-age \*

### Note:

The CORS configuration consists of multiple rules, which are matched individually from top to bottom. The first matched rule will be applied.

### Verifying result

After the configuration is completed, try accessing the `test.txt` file again. If the result is as follows, the file can be accessed normally.

[Test CORS](#)

```
127.0.0.1:8081
request success, CORS allow.
url: http://corstest-125...myqcloud.com/
test.txt
status: 200
headers:
date: Thu, 08 Aug 2019 07:06:30 GMT
x-cache-lookup: HIT from SK-SQUIDWEB-58:8080
last-modified: Mon, 22 Jul 2019 07:11:57 GMT
server: tencent-cos
age: 190
```

Status Code: 200 OK  
Remote Address: ...  
Referrer Policy: no-referrer-when-downgrade

## Troubleshooting and suggestions

To avoid problems related to cross-origin access, you can set the least restricted CORS rule as described above to allow all cross-origin requests. If an error still occurs under this configuration, the root cause may lie in other factors rather than CORS.

In addition to configuring the least restricted rule, you can also configure a more refined rule. For example, in this example, you can use the following most restricted configuration to ensure a successful match:

### Add rules ✕

Origin \*

A line can contain at most one \* wildcard character

Allow-Methods \*  PUT  GET  POST  DELETE  HEAD

Allow-Headers

Expose-Headers

Max-age \*

Therefore, for most scenarios, we recommend you use the most restricted configuration as needed to ensure security.

## CORS Configuration Items

CORS configuration items are as follows:

### Origin

This refers to the origin allowing cross-origin requests.

Multiple domain names can be specified, with one domain name per line.

Asterisk (\*) is supported, meaning that all domain names are allowed. This is not recommended.

A single specific domain name such as `http://www.abc.com` is supported.

Second-level wildcard domain names such as `http://*.abc.com` are supported. Note that each line can contain only one `*`.

Do not omit the protocol name HTTP or HTTPS. If the port is not the default port 80, the port should also be specified.

## Allow-Methods

Enumerate one or multiple allowed cross-origin request methods.

Examples: GET, PUT, POST, DELETE, and HEAD.

## Allow-Headers

Allowed cross-origin request header.

Multiple domain names can be specified, with one domain name per line.

Headers may be easily omitted. Therefore, if there is no special requirement, we recommend you set this field to `*`, meaning that all headers are allowed.

The values are case-insensitive.

Each header specified in `Access-Control-Request-Headers` must correspond to a value in `Allow-Headers`.

## Expose-Headers

This is a list of headers exposed to the browser, i.e., the response headers that you access from the application (for example, JavaScript's `XMLHttpRequest` object).

The configuration should be specific to the requirements of the application. `ETag` is recommended by default.

Wildcard is not allowed. The headers are case-insensitive, with one header per line.

## Max-Age

This is the time (in seconds) the browser can cache the results of a preflight request (OPTIONS request) for specific resources. In general cases, you can set it to a bigger value, for example, 60 seconds. This configuration item is optional.

# Building a Frontend Single-Page Application with COS's Static Website Feature

Last updated : 2024-03-25 15:11:17

## What Is a Single-Page Application?

A single-page application (SPA), is a model of a web application or website that interacts with the user by dynamically rewriting the current page, instead of the traditional method of reloading entire new pages from the server. This approach avoids interruptions to the user experience by switching between pages and makes the application more like a desktop application. In an SPA, all necessary code (HTML, JavaScript, and CSS) is either retrieved with a single page load or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions.

At present, in the field of frontend development, common SPA development frameworks include React, Vue, and Angular.

This document uses two popular frameworks to illustrate how to use the **static website** feature provided by **Tencent Cloud's Cloud Object Storage (COS)** to quickly build an online available SPA, and provides solutions to some common problems.

## Preparations

1. Install the Node.js environment.
2. Sign up for a Tencent Cloud account and verify your identity to ensure that you can log in to the [Tencent Cloud COS console](#).
3. Create a bucket (to facilitate testing, set the bucket permission to **Public read & Private write**).

## Writing Frontend Code

### Note:

If you have already implemented the code, skip to [Configuring the Bucket Static Website](#).

### Quickly building an SPA with Vue

1. Run the following command to install the Vue CLI:

```
npm install -g @vue/cli
```

2. Run the following command in the Vue CLI to quickly create a Vue project. For more information, see the [official document](#).

```
vue create vue-spa
```

3. Run the following command to install `vue-router` in the root directory of the project:

```
npm install vue-router -S (Vue 2.x)
```

Or

```
npm install vue-router@4 -S (Vue 3.x)
```

4. Modify the `main.js` and `App.vue` files in the project.

Modify `main.js` as follows:



```
import { createApp } from 'vue'
import { createRouter, createWebHistory } from 'vue-router'
import App from './App.vue'
import Home from './components/Home.vue'
import Foo from './components/Foo.vue'
import Bar from './components/Bar.vue'
import Default from './components/404.vue'

const routes = [
  { path: '/', component: Home },
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar },
  { path: '/*', component: Default }
]

const router = createRouter({
  history: createWebHistory(),
  routes
})

const app = createApp(App)
app.use(router)
app.mount('#app')
```

In `App.vue`, mainly modify the component template. See the figure below.

```
<template>
  <div>
    <ul>
      <li>
        <router-link to="/">Home</router-link>
      </li>
      <li>
        <router-link to="/foo">Foo</router-link>
      </li>
      <li>
        <router-link to="/bar">Bar</router-link>
      </li>
    </ul>
    
    <router-view></router-view>
  </div>
</template>
```

**Note:**

For simplicity, only some key code is shown here. For the full code, [click here](#) to download.

5. After modifying the code, run the following command for local preview:

```
npm run serve
```

6. After debugging and preview check, run the following command to package the production environment code:



```
npm run build
```

The `dist` directory is generated in the root directory of the project, and the Vue program code is ready.

## Quickly building an SPA with React

1. Run the following command to install `create-react-app` :

```
npm install -g create-react-app
```

2. Use `create-react-app` to quickly create a React project. For more information, please see the [official document](#).

3. Run the following command to install `react-router-dom` in the root directory of the project:

```
npm install react-router-dom -S
```

4. Modify the `App.js` file in the project.

```
import React from 'react';
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';
import './App.css';

function Home() {
  return <h2>Home</h2>;
}

function About() {
  return <h2>About</h2>;
}

function NoMatch() {
  return <h2>404 Page</h2>
}

function App() {
  return (
    <Router>
      <div className="App">
        <nav>
          <ul>
```

```
        <li>
          <Link to="/">Home</Link>
        </li>
        <li>
          <Link to="/about">About</Link>
        </li>
      </ul>
    </nav>
    <Switch>
      <Route exact path="/">
        <Home />
      </Route>
      <Route path="/about">
        <About />
      </Route>
      <Route path="*">
        <NoMatch />
      </Route>
    </Switch>
  </div>
</Router>
);
}

export default App;
```

**Note:**

For simplicity, only some key code is shown here. For the full code, [click here](#) to download.

5. After modifying the code, run the following command for local preview:

```
npm run start
```

6. After debugging and preview check, run the following command to package the production environment code:

```
npm run build
```

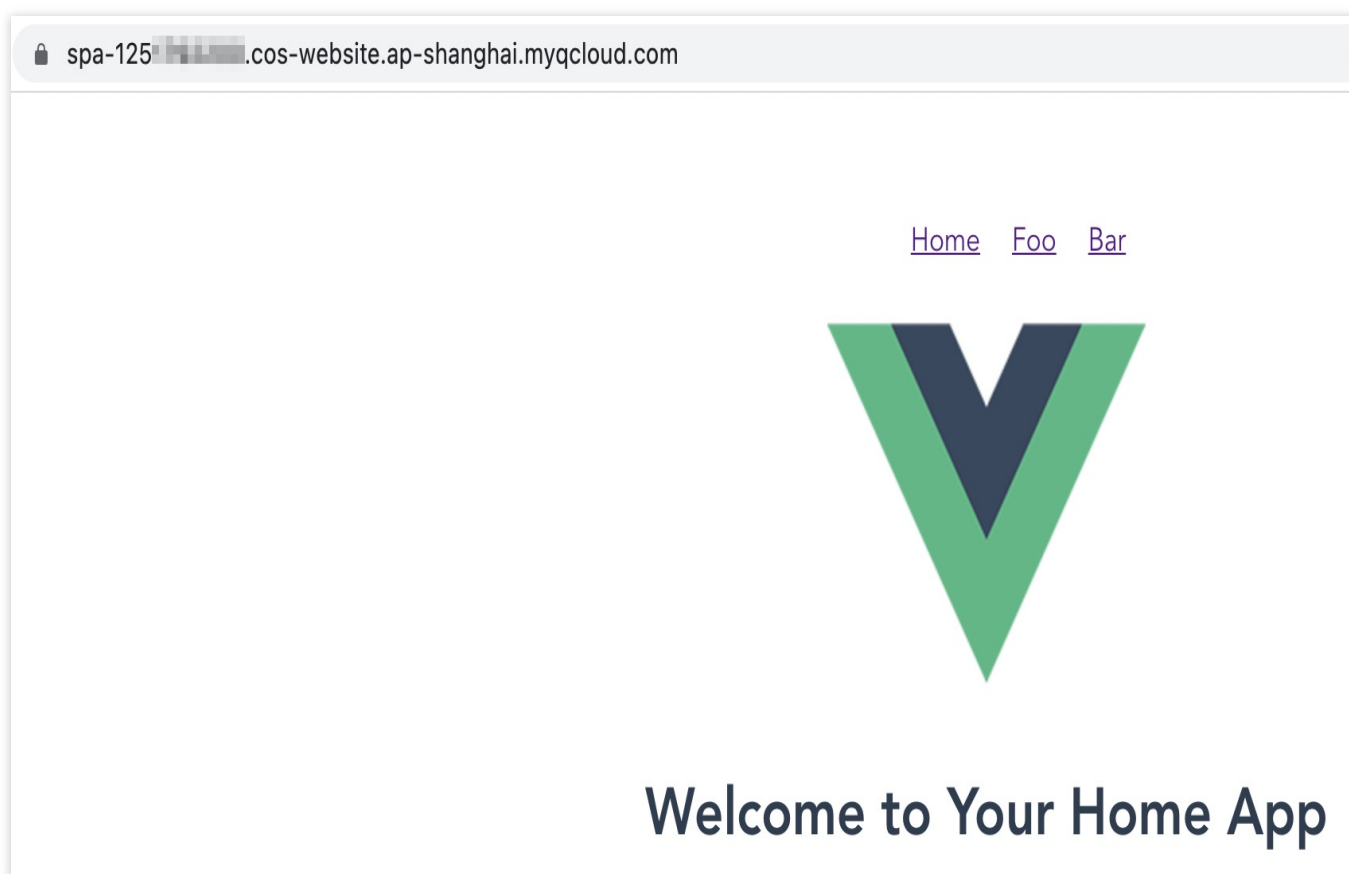
The `build` directory is created in the root directory of the project, and the React program code is ready.

## Configuring the Bucket Static Website

1. Go to the details page of the created bucket and choose **Basic Configurations** > **Static Website**.
2. On the static website management page, configure information as shown in the figure below. For operation details, please see [Setting Up a Static Website](#).

## Deploying the SPA to COS

1. Locate the bucket for which the static website is configured, and go to the corresponding **File List** page.
  2. Upload all files in the compilation directory (default compilation directory: `dist` for Vue and `build` for React) to the root directory of the bucket. For operation details, please see [Uploading Objects](#).
  3. Access the bucket's static website domain (the access node shown in the figure below).
- Then you can view the homepage of the deployed SPA (the Vue application in this example).



4. Try to switch between routes (**Home**, **Foo**, and **Bar**) and refresh the page to check whether the application works properly (no 404 error is reported upon refreshing under new routes).

## FAQs

## What if I don't want to use the default domain name of the static website? Can I use my own domain name?

In addition to the default static website endpoint mentioned above, COS also allows you to set the custom CDN acceleration domain name and custom endpoint. For configuration details, see [Domain Name Management Overview](#). After successful configuration, you can use your desired domain name to access the application.

Note that if you choose to configure a CDN acceleration domain name, refer to [Node Cache Validity Configuration](#) to get the updated data.

## After the application is deployed, rendering is successful after route switching, but the 404 error is reported whenever the page is refreshed. Why is that?

This may be caused by the missing or incorrect configuration of **Error document**. As the figure in [Configuring the Bucket Static Website](#) shows, **Error document** and **Index document** are both set to `index.html`.

Due to the nature of single-page applications, we need to ensure that the application entry (typically `index.html`) can be successfully accessed in any case in order to trigger a set of internal logic for subsequent routing.

## After the route is switched, the page is displayed normally, but the HTTP status code is still 404. How do I make the HTTP status code 200?

The reason is that **Error Document Response Code** is not set during static website configuration. To solve this problem, set **Error Document Response Code** to 200. See the figure in [Configuring the Bucket Static Website](#).

## What should I do to make the application still return the status code 404 for accessing an incorrect path after Error document is set?

It is recommended to implement 404 logic in the frontend code: configure an underlying matching rule at the bottom layer of the routing configuration to configure the system to render a 404 component if the matching of all the preceding rules fails. The content of the 404 component can be designed and implemented according to your own requirements. For more information, please see the last configuration of the routing configuration in the code demo provided in this document.

## Why is the error "403 Access Denied" reported during page access?

The possible cause is that the bucket permission is set to **Private (read-write)**. To solve the problem, change the permission to **Public read & Private write**.

In addition, if you use the CDN acceleration domain name to access a bucket with the **Private (read-write)** permission, be sure to enable **origin-pull authentication** so that you can authorize the CDN service to access COS resources.

# Configuring a Custom CDN Domain Name to Support Gzip Compression

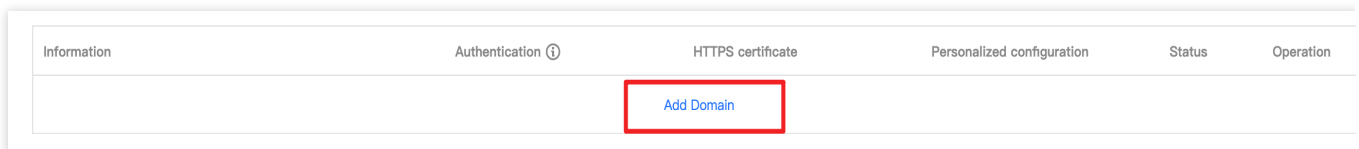
Last updated : 2024-12-16 16:34:35

## Overview

This document only describes how to configure Gzip compression in the Cloud Object Storage (COS) console. To add it in the CDN console, see the CDN [Intelligent Compression Configuration](#) document.

## Directions

1. Log in to the [COS console](#).
2. In the left sidebar, click **Bucket List** to go to the bucket list page.
3. Click the bucket that requires configuring Gzip compression to go to the bucket configuration page.
4. Click **Domains and Transfer > Custom CDN Acceleration Domain** on the left. If there is no available domain name, add one first. For specific configuration, refer to [Enabling Custom CDN Acceleration Domain Name](#).



5. After the domain name is successfully added, the CDN intelligent compression feature is enabled for it by default, and a Gzip compression rule is created.

The details of the Gzip compression rule are as follows:

**Type:** File extension

**Content:** js;html;css;xml;json;shtml;htm

**File Size:** 256 Byte to 2 MB

**Compression Method:** Gzip

6. In the **Personalized configuration** column, **CDN intelligent compression** allows you to view the configuration status of Gzip compression and the configured compression method.

If "Not configured" is displayed, it means the CDN intelligent compression configuration feature is not enabled.

To reduce the size of transferred files and save costs, it is recommended to enable the CDN intelligent compression feature. You can click the



icon and click **CDN Console** according to the reminder to quickly jump to the CDN console for configuration. For operation instructions, see [Intelligent Compression Configuration](#).

If "gzip" or another field is displayed, it indicates the currently configured compression method.

To add, modify, or view CDN intelligent compression rules (such as compression method, type, content, and file scope), you can click the



icon and click **CDN Console** according to the reminder to quickly jump to the CDN console for configuration. For operation instructions, see [Intelligent Compression Configuration](#).

#### **Note**

For more information on intelligent compression configuration rules, such as configuration constraints and the priority of rules to take effect, see [Intelligent Compression Configuration](#) in the CDN Configuration Guide.

# Image Processing

## Hybrid Watermarking

Last updated : 2024-03-25 15:11:19

### Overview

The basic image processing feature of CI enables you to add an [image](#) or [text watermark](#) to images. However, in actual business scenarios, an image may contain both a fixed logo watermark and a dynamically changing text watermark (username). For such scenarios, the following integration methods are provided. You can select an appropriate one based on your actual business scenario.

### Method Comparison

Method	Pros	Cons
1	It is easy to integrate and implement.	The watermark size cannot change dynamically with the image size or tiled.
2	It can better fit the image when the image size changes frequently.	It is difficult to integrate and is charged processing fees twice.

### How to Use

#### Method 1. Using a pipeline operator to add two types of watermarks with only one URL

The basic image processing feature of CI allows you to use [pipeline operators](#) "|" to process an image multiple times with only one request at a one-time charge of processing and traffic fees. This method greatly reduces the latency and additional fees caused by repeated requests.

#### Directions

1. Define the image watermark parameters as instructed in [Image Watermarking](#).

If you are unfamiliar with API parameters, you can add a style in the console to generate parameters as instructed in [Basic Processing](#).

Below are the processing parameters:

```
watermark/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTgxMjU2MzguY29zLmFwLWd1YW5nemhvdS5teXFjb
```

**Note:**

Here,

`aHR0cDovL2V4YW1wbGVzLTEyNTgxMjU2MzguY29zLmFwLWd1YW5nemhvdS5teXFjbG91ZC5jb20vbG9nby5wbmc` is the URL-safe Base64-encoded URL of the image watermark (image stored in the COS bucket).

2. Define the text watermark parameters as instructed in [Text Watermarking](#).

If you are unfamiliar with API parameters, you can add a style in the console to generate parameters as instructed in [Basic Processing](#).

```
watermark/2/text/VU100iAxMjM0NTY3OA/font/SGVsdmV0aWNhLmRmb250/fontsize/36/fill/IzAw
```

**Note:**

Here, `VU100iAxMjM0NTY3OA` is the URL-safe Base64-encoded text of `UIN: 12345678`.

3. Use a pipeline operator to concatenate image and text watermark parameters:

```
watermark/2/text/VU100iAxMjM0NTY3OA/font/SGVsdmV0aWNhLmRmb250/fontsize/36/fill/IzAw
```

4. Add the concatenated parameters to the end of the image download URL:

```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/2/t
```

To shorten the URL, you can add the part of the image watermark (unchanged) as the `watermark1` style in the console as instructed in [Basic Processing](#).

In this way, the URL can be shortened to:

```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png/watermark1?wa
```

If you need to change the text content later, you only need to replace `VU100iAxMjM0NTY3OA` in the URL with the updated Base64 code; for example, `UIN: 88888888` is encoded into `VU100iA4ODg4ODg4OA`, so you only need to change the URL to the following content to replace the text:

```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png/watermark1?wa
```

## Method 2. Printing the text and image watermarks onto a transparent image and using the output image as the final image watermark

1. Upload a 400x400 px transparent PNG image to the bucket as instructed in [File Management](#).

For example: `https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/transparent.png`

2. Generate and concatenate image and text watermark parameters as instructed in **steps 1–3 of method 1**.

3. Add the concatenated parameters at the end of the download URL of the transparent PNG image:

```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/transparent.png?watermark
```

4. Use the transparent image as the image watermark and add the watermark to the original image:



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/1/i
```

You can also use the `scatype` parameter to scale the image watermark proportionally to the image size and use the `batch` parameter to tile the image watermark.

```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/1/i
```

# Audio/Video Practices

## COS Audio/Video Player Overview

Last updated : 2024-03-25 15:11:17

This document describes how to process COS audio/video files in the cloud and play back them on the client. Examples in this document cover protocols and features supported for audio/video processing and offer you more ideas on using the product features based on the rich audio/video processing capabilities of the [media processing service](#) in CI to help you improve the playback performance.

### Supported protocols

Audio/Video Protocol	URL Format	PC Browser	Mobile Browser
MP3	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp3	Supported	Supported
MP4	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4	Supported	Supported
HLS (M3U8)	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8	Supported	Supported
FLV	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.flv	Supported	Supported
DASH	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mpd	Supported	Supported

#### Note:

In some browser environments, HLS, FLV, and DASH video playback depends on [Media Source Extensions](#).

### Supported features

Feature	TCPlayer	DPlayer	VideojsPlayer
MP4 video playback	<a href="#">View details</a>	<a href="#">View details</a>	<a href="#">View details</a>

HLS video playback	<a href="#">View details</a>	<a href="#">View details</a>	<a href="#">View details</a>
FLV video playback	<a href="#">View details</a>	<a href="#">View details</a>	<a href="#">View details</a>
DASH video playback	<a href="#">View details</a>	<a href="#">View details</a>	<a href="#">View details</a>
PM3U8 (private M3U8) video playback	<a href="#">View details</a>	<a href="#">View details</a>	<a href="#">View details</a>
Thumbnail configuration	<a href="#">View details</a>	<a href="#">View details</a>	<a href="#">View details</a>
Standard HLS encryption configuration	<a href="#">View details</a>	<a href="#">View details</a>	<a href="#">View details</a>
Definition switch	<a href="#">View details</a>	<a href="#">View details</a>	-
Dynamic watermark configuration	<a href="#">View details</a>	-	-
Top-left logo configuration	-	<a href="#">View details</a>	-
Progress preview image configuration	<a href="#">View details</a>	-	-
Subtitles configuration	<a href="#">View details</a>	-	-
Multilingual configuration	<a href="#">View details</a>	-	-
Roll image ad configuration	<a href="#">View details</a>	-	-

**Note:**

The player is compatible with mainstream browsers and can automatically select the optimal playback scheme depending on the browser used. For example, for modern browsers such as Chrome, the player uses the HTML5 technology for playback, and for mobile browsers, it uses the HTML5 technology or the browser's built-in capabilities.

## Usage guide

[Playing back COS Video File with TCPlayer](#)

[Playing back Video in COS with DPlayer](#)

[Playing back Video in COS with VideojsPlayer](#)

# Playing back COS Video File with TCPlayer

Last updated : 2024-03-25 15:11:19

## Overview

This document describes how to use the TCPlayer integrated in the TCToolkit SDK together with the rich audio/video capabilities of [Cloud Infinite \(CI\)](#) to play back video files stored in COS in a web browser.

## Integration Guide

### Step 1. Import player style and script files into the page

```
<!--Player style file-->
<link
href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/tcplayer.min.css" rel="stylesheet">
<!--Player script file-->
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.0/tcplayer.v4.5.0.min.js"></script>
```

#### Note:

We recommend you deploy the above static resources on your own when using the player SDK. Click [here](#) to download the player resources.

Deploy the folder generated after decompression. Do not adjust the directories in the folder; otherwise, resource import exceptions may occur.

### Step 2. Set the player container node

Place the player container in the desired place on the page. For example, add the following code to `index.html` (the container ID, width, and height can be customized).

```
<video id="player-container-id" width="414" height="270" preload="auto"
playsinline webkit-playsinline>
</video>
```

#### Note:

The player container must be a `<video>` tag.

The `player-container-id` in the sample is the ID of the player container, which can be customized.

We recommend you set the size of the player container zone through CSS, which is more flexible than the attribute and can achieve effects such as fit to full screen and container adaption.

The `preload` attribute in the sample specifies whether to load the video after the page is loaded, which is usually set to `auto` for faster playback start. Other options include `meta` (to only load the metadata after the page is loaded) and `none` (to not load the video after the page is loaded). Due to system restrictions, videos will not be automatically loaded on mobile devices.

The `playsinline` and `webkit-playsinline` attributes are used to implement inline playback if the standard mobile browser does not hijack the video playback. They are just for reference here and can be used as needed.

If the `x5-playsinline` attribute is set, the X5 UI player will be used in the TBS kernel.

### Step 3. Get the video file object address

1. [Create a bucket](#).
2. [Upload a video file object](#).
3. Get the video file object address in the format of `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<video format>`.

#### Note:

If cross-origin access is involved, you need to set CORS for the bucket as instructed in [Setting CORS](#).

If the bucket permission is private read/write, the object address needs to carry a signature. For more information, see [Request Signature](#).

### Step 4. Initialize the player and pass in the COS video file object URL

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is
the player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); //
COS video object address
```

## Feature Guide

### Playing back video files in different formats

1. Get the object address of the video file in the COS bucket.

#### Note:

We recommend you transcode videos for playback because untranscoded videos may experience compatibility issues during playback. You can get video files in different formats with CI's [audio/video transcoding](#) feature.

2. For different video formats, in order to ensure the compatibility with different browsers, corresponding dependencies need to be imported.

MP4: There is no need to import other dependencies.

HLS: If you want to play back HLS videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `hls.min.js` before importing `tcplayer.min.js`.

```
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.min.0.1
3.2m.js"></script>
```

FLV: If you want to play back FLV videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `flv.min.js` before importing `tcplayer.min.js`.

```
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.min.1.6
.2.js"></script>
```

DASH: You need to import the `dash.all.min.js` file.

```
<script src="https://cos-video-1258344699.cos.ap-
guangzhou.myqcloud.com/lib/dash.all.min.js"></script>
```

3. Initialize the player and pass in the object address.

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is
the player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); //
COS video address
```

Get code samples:

[Sample code for MP4 playback](#)

[Sample code for FLV playback](#)

[Sample code for HLS playback](#)

[Sample code for DASH playback](#)

## Playing back PM3U8 video

PM3U8 refers to private M3U8 video file. COS provides a download authorization API for getting private M3U8 TS resources. For more information, see [Private M3U8 API](#).

```
var player = TCPlayer("player-container-id", {
  poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-
process=pm3u8&expires=3600" // Relative validity period of the download
credential for the private TS resource URL, which is 3,600 seconds.
```

```
});
```

Get code samples:

[Sample code for PM3U8 playback](#)

## Setting thumbnail

1. Get the object address of the thumbnail in the COS bucket.

### Note:

CI's [intelligent thumbnail](#) feature can extract optimal frames to generate thumbnails to make the video content more engaging.

2. Set the thumbnail.

```
var player = TCPlayer("player-container-id", {
  poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png"
});
```

Get code samples:

[Sample code for thumbnail configuration](#)

## Playing back HLS encrypted video

To ensure the security of video content and prevent unauthorized download and distribution of videos, CI provides the feature of encrypting HLS video content, which is more secure than privately readable files. Encrypted videos cannot be distributed to users without access for playback. Even if they are downloaded, they are still encrypted and cannot be redistributed maliciously. This protects your video copyrights from being infringed upon.

The steps are as follows:

1. Generate an encrypted video as instructed in [Playing back HLS Encrypted Video](#).
2. Initialize the player and pass in the video object address.

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is
the player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); //
HLS encrypted video address
```

Get code samples:

[Sample code for HLS encrypted playback](#)

## Switching definition

CI's [adaptive bitrate streaming](#) feature can transcode a video and remux it into adaptive bitstreams for output, helping you quickly distribute video content in different network conditions. The player can dynamically select the most appropriate bitrate to play back the video based on the current bandwidth.

The steps are as follows:

1. Generate the multi-bitrate adaptive HLS or DASH target file with CI's [adaptive bitrate streaming](#) feature.
2. Initialize the player and pass in the video object address.

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is
the player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); //
Multi-bitrate video address
```

Get code samples:

[Sample code for definition switching](#)

## Setting dynamic watermark

The player supports adding a dynamic watermark that changes its position and speed to a video. When using the dynamic watermark feature, the reference of the player object should not be exposed to the global environment; otherwise, the dynamic watermark can be easily removed. CI also allows you to add a dynamic watermark to a video in the cloud. For more information, see [Watermark Template APIs](#).

```
var player = TCPlayer("player-container-id", {
  plugins:{
    DynamicWatermark: {
      speed: 0.2, // Speed
      content: "Tencent Cloud CI", // Text
      opacity: 0.7 // Opacity
    }
  }
});
```

Get code samples:

[Sample code for dynamic watermark configuration](#)

## Setting roll image ad

The steps are as follows:

1. Prepare the ad thumbnail and link.
2. Initialize the player, set the ad thumbnail and link, and set the ad node.

```
var PosterImage = TCPlayer.getComponent('PosterImage');
PosterImage.prototype.handleClick = function () {
```



```
window.open('https://intl.cloud.tencent.com/products/ci'); // Set the ad link
};

var player = TCPlayer('player-container-id', {
  poster: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png', // Ad
  thumbnail
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');

var adTextNode = document.createElement('span');
adTextNode.className = 'ad-text-node';
adTextNode.innerHTML = 'Ad';

var adCloseIconNode = document.createElement('i');
adCloseIconNode.className = 'ad-close-icon-node';
adCloseIconNode.onclick = function (e) {
  e.stopPropagation();
  player.posterImage.hide();
};

player.posterImage.el_.appendChild(adTextNode);
player.posterImage.el_.appendChild(adCloseIconNode);
```

Get code samples:

[Sample code for roll image ad configuration](#)

## Setting video progress thumbnail (image sprite)

The steps are as follows:

1. Generate an image sprite with CI's [video frame capturing](#) feature.
2. Get the object addresses of the image sprite and VTT (image sprite location description file) generated in step 1.
3. Initialize the player and set the video and VTT file addresses.

```
var player = TCPlayer('player-container-id', {
  plugins: {
    VttThumbnail: {
      vttUrl: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.vtt' //
      VTT file
    },
  },
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
```

Get code samples:

[Sample code for image sprite configuration](#)

## Setting video subtitles

The steps are as follows:

1. Generate a subtitle file with CI's speech recognition feature.
2. Get the object address of the SRT file generated in step 1.
3. Initialize the player and set the video and SRT file addresses.

```
var player = TCPlayer('player-container-id', {});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
  // Add the subtitles file
  var subTrack = player.addRemoteTextTrack({
    src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.srt', //
Subtitles file
    kind: 'subtitles',
    srclang: 'zh-cn',
    label: 'Chinese',
    default: 'true',
  }, true);
});
```

Get code samples:

[Sample code for video subtitles configuration](#)

## Setting multilingual video subtitles

The steps are as follows:

1. Generate a subtitles file with CI's speech recognition feature and translate it into multiple languages.
2. Get the object address of the multilingual SRT file generated in step 1.
3. Initialize the player and set the video and multilingual SRT file addresses.

```
var player = TCPlayer('player-container-id', {});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
  // Set Chinese subtitles
  var subTrack = player.addRemoteTextTrack({
    src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/zh.srt', // Subtitles f
    kind: 'subtitles',
    srclang: 'zh-cn',
    label: 'Chinese',
    default: 'true',
  }, true);
  // Set English subtitles
```

```
var subTrack = player.addRemoteTextTrack({
  src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/en.srt', // Subtitles f
  kind: 'subtitles',
  srclang: 'en',
  label: 'English',
  default: 'false',
  }, true);
});
```

Get code samples:

[Sample code for multilingual subtitles configuration](#)

# Playing back Video in COS with DPlayer

Last updated : 2024-03-25 15:11:17

## Overview

This document describes how to use [DPlayer](#) together with the rich audio/video capabilities of [Cloud Infinite \(CI\)](#) to play back video files stored in COS in a web browser.

## Integration Guide

### Step 1. Import player script files and required dependency files into the page

```
<!-- Player script file -->
<script src="https://cdn.jsdelivr.net/npm/dplayer@1.26.0/dist/DPlayer.min.js">
</script>
```

#### Note:

We recommend you deploy the above static resources on your own when using the player.

### Step 2. Set the player container node

Place the player container in the desired place on the page. For example, add the following code to `index.html` (the container ID, width, and height can be customized).

```
<div id="dplayer" style="width: 100%; height: 100%"></div>
```

### Step 3. Get the video file object address

1. [Create a bucket.](#)
2. [Upload a video file object.](#)
3. Get the video file object address in the format of `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<video format>`.

#### Note:

If cross-origin access is involved, you need to set CORS for the bucket as instructed in [Setting CORS](#).

If the bucket permission is private read/write, the object address needs to carry a signature. For more information, see [Request Signature](#).

### Step 4. Initialize the player and pass in the COS video file object URL

```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
    url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4', //
    COS video object address
  },
});
```

## Function Guide

### Playing back video files in different formats

1. Get the object address of the video file in the COS bucket.

#### Note:

We recommend you transcode videos for playback because untranscoded videos may experience compatibility issues during playback. You can get video files in different formats with CI's [audio/video transcoding](#) feature.

2. For different video formats, in order to ensure the compatibility with different browsers, corresponding dependencies need to be imported.

MP4: There is no need to import other dependencies.

HLS: If you want to play back HLS videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `hls.min.js` before importing `tcplayer.min.js`.

```
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.min.0.1
3.2m.js"></script>
```

FLV: If you want to play back FLV videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `flv.min.js` before importing `tcplayer.min.js`.

```
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.min.1.6
.2.js"></script>
```

DASH: You need to import the `dash.all.min.js` file.

```
<script src="https://cos-video-1258344699.cos.ap-
guangzhou.myqcloud.com/lib/dash.all.min.js"></script>
```

3. Initialize the player and pass in the object address.

```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
    url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4', // COS
video object address
  },
});
```

Get code samples:

[Sample code for MP4 playback](#)

[Sample code for FLV playback](#)

[Sample code for HLS playback](#)

[Sample code for DASH playback](#)

## Playing back PM3U8 video

PM3U8 refers to private M3U8 video file. COS provides a download authorization API for getting private M3U8 TS resources. For more information, see [Private M3U8 API](#).

```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  // For more information on pm3u8, visit
https://intl.cloud.tencent.com/document/product/436/47220.
  video: {
    url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-
process=pm3u8&expires=3600' // Relative validity period of the download
credential for the private TS resource URL, which is 3,600 seconds.
  }
});
```

Get code samples:

[Sample code for PM3U8 playback](#)

## Setting thumbnail

1. Get the object address of the thumbnail in the COS bucket.

### Note:

CI's [intelligent thumbnail](#) feature can extract optimal frames to generate thumbnails to make the video content more engaging.

2. Initialize the player and set the thumbnail image.

```
const dp = new DPlayer({
```

```
container: document.getElementById('dplayer'),
video: {
url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4',
pic: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png',
},
});
```

Get code samples:

[Sample code for thumbnail configuration](#)

## Playing back HLS encrypted video

To ensure the security of video content and prevent unauthorized download and distribution of videos, CI provides the feature of encrypting HLS video content, which is more secure than privately readable files. Encrypted videos cannot be distributed to users without access for playback. Even if they are downloaded, they are still encrypted and cannot be redistributed maliciously. This protects your video copyrights from being infringed upon.

Follow the steps below:

1. Generate an encrypted video as instructed in [Playing back HLS Encrypted Video](#) and [COS Audio/Video Practice | Encrypting Your Video](#).
2. Initialize the player and pass in the video object address.

```
const dp = new DPlayer({
container: document.getElementById('dplayer'),
video: {
url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8' //
Encrypted video address
}
});
```

Get code samples:

[Sample code for HLS encrypted playback](#)

## Switching definition

CI's [adaptive bitrate streaming](#) feature can transcode a video and remux it into adaptive bitstreams for output, helping you quickly distribute video content in different network conditions. The player can dynamically select the most appropriate bitrate to play back the video based on the current bandwidth. For more information, see [COS Audio/Video Practice | Playing back Multi-Definition Video with Data Processing Workflow](#).

Follow the steps below:

1. Generate the multi-bitrate adaptive HLS or DASH target file with CI's [adaptive bitrate streaming](#) feature.
2. Initialize the player and pass in the video object address.

```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
    url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8', //
    Multi-bitrate HLS/DASH video
  },
});
```

Get code samples:

[Sample code for definition switching](#)

## Setting the top-left logo

The player allows you to set a logo in the top-left corner.

Follow the steps below:

1. Get the object address of the logo in the COS bucket.
2. Initialize the player and set the logo.

```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
    url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4',
  },
  logo: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.svg'
});
```

Get code samples:

[Sample code for setting the top-left logo](#)



# Playing back Video in COS with VideojsPlayer

Last updated : 2024-03-25 15:11:19

## Overview

This document describes how to use [VideojsPlayer](#) together with the rich audio/video capabilities of [Cloud Infinite \(CI\)](#) to play back video files stored in COS in a web browser.

## Integration Guide

### Step 1. Import player style and script files into the page

```
<!-- Player style file -->
<link href="https://vjs.zencdn.net/7.19.2/video-js.css" rel="stylesheet" />
<!-- Player script file -->
<script src="https://vjs.zencdn.net/7.19.2/video.min.js"></script>
```

#### Note:

We recommend you deploy the above static resources on your own when using the player.

### Step 2. Set the player container node

Place the player container in the desired place on the page. For example, add the following code to `index.html` (the container ID, width, and height can be customized).

```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
  height="100%"
  data-setup="{}"
></video>
```

### Step 3. Get the video file object address

1. [Create a bucket.](#)
2. [Upload a video file object.](#)

3. Get the video file object address in the format of `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<video format>` .

**Note:**

If cross-origin access is involved, you need to set CORS for the bucket as instructed in [Setting CORS](#).

If the bucket permission is private read/write, the object address needs to carry a signature. For more information, see [Request Signature](#).

**Step 4: Set the video address in the player container and pass in the COS video file object URL**

```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
  height="100%"
  data-setup="{}"
>
  <source
    src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
    type="video/mp4"
  />
</video>
```

## Function Guide

### Playing back video files in different formats

1. Get the object address of the video file in the COS bucket.

**Note:**

We recommend you transcode videos for playback because untranscoded videos may experience compatibility issues during playback. You can get video files in different formats with CI's [audio/video transcoding](#) feature.

2. For different video formats, in order to ensure the compatibility with different browsers, corresponding dependencies need to be imported.

MP4: There is no need to import other dependencies.

HLS: If you want to play back HLS videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `hls.min.js` before importing `tcplayer.min.js` .

```
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.min.0.1
3.2m.js"></script>
```

FLV: If you want to play back FLV videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `flv.min.js` before importing `tcplayer.min.js` .

```
<script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.min.1.6
.2.js"></script>
```

DASH: You need to import the `dash.all.min.js` file.

```
<script src="https://cos-video-1258344699.cos.ap-
guangzhou.myqcloud.com/lib/dash.all.min.js"></script>
```

3. Initialize the player and pass in the object address.

```
<!-- MP4 -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
  type="video/mp4"
/>

<!-- HLS -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"
  type="application/x-mpegURL"
/>

<!-- FLV -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.flv"
  type="video/x-flv"
/>

<!-- DASH -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mpd"
  type="application/dash+xml"
/>
```

Get code samples:

[Sample code for MP4 playback](#)

[Sample code for FLV playback](#)

[Sample code for HLS playback](#)

[Sample code for DASH playback](#)

## Playing back PM3U8 video

PM3U8 refers to private M3U8 video file. COS provides a download authorization API for getting private M3U8 TS resources. For more information, see [Private M3U8 API](#).

```
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-
process=pm3u8&expires=3600"
  type="application/x-mpegURL"
/>
```

Get code samples:

[Sample code for PM3U8 playback](#)

## Setting thumbnail

1. Get the object address of the thumbnail in the COS bucket.

### Note:

CI's [intelligent thumbnail](#) feature can extract optimal frames to generate thumbnails to make the video content more engaging.

2. Initialize the player and set the thumbnail image.

```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
  height="100%"
  data-setup="{ }"
  poster="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/poster.png"
>
  <source
    src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
    type="video/mp4"
  />
</video>
```

Get code samples:

[Sample code for thumbnail configuration](#)

## Playing back HLS encrypted video

To ensure the security of video content and prevent unauthorized download and distribution of videos, CI provides the feature of encrypting HLS video content, which is more secure than privately readable files. Encrypted videos cannot be distributed to users without access for playback. Even if they are downloaded, they are still encrypted and cannot be redistributed maliciously. This protects your video copyrights from being infringed upon.

Follow the steps below:

1. Generate an encrypted video as instructed in [Playing back HLS Encrypted Video](#) and [COS Audio/Video Practice | Encrypting Your Video](#).
2. Initialize the player and pass in the video object address.

```
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"
  type="application/x-mpegURL"
/>
```

Get code samples:

[Sample code for HLS encrypted playback](#)

# Workflow

## Using Custom Function to Manage COS Files

Last updated : 2024-03-25 15:16:26

### Overview

COS workflow provides a series of processing capabilities for media files such as audios, videos, and images. You can flexibly and quickly create a media processing workflow as needed. To satisfy your needs for customization and guarantee the flexibility, COS workflow offers the custom function feature for you to implement custom logic in SCF by configuring function nodes in a workflow.

To make this feature easier to use, COS workflow provides common function feature templates and integrates their creation to node configuration steps, facilitating subsequent processing of source and output files. Currently, supported basic operations include modifying object attributes, moving objects, and deleting objects.

### Use Cases

You want to move or transition source files after media processing to reduce the storage costs.

You want to tag output files or modify their headers after media processing to facilitate subsequent business use.

### Solution Strengths

Out-of-the-box service: You can use the service after simple configuration, with no need to develop the function logic or care about the complex deployment process.

Flexible configuration: You can configure nodes of common features as needed to perform different business operations on source and output files.

Easy extension: You can modify the function logic to meet more customization requirements.

### Directions

1. Log in to the [COS console](#).
2. Click **Bucket List** on the left sidebar.
3. Click the target bucket to enter the bucket details page.
4. On the left sidebar, select **Data Processing Workflow > Workflow** and click **Create Workflow**.

5. On the workflow creation page, configure the media processing node required by the business such as **Audio/Video Transcoding**. For more information, see [Workflow](#).
6. On the workflow creation page, add the **Custom Function** node and select a required common feature function. If you haven't created a function of this type, click **Create**.

Create a common feature function as follows:

1. Enter the basic function configuration: Enter the function name prefix, select **Authorize SCF Service**, and click **Next**.
2. Configure attributes: Set the storage class and custom header based on the business needs and click **Next**.
3. Select the object to be processed. You can perform this operation only on the workflow source file.
4. Click **OK**.
5. COS workflow encapsulates processes such as function creation, version release, and alias-based stream switch. Wait for the workflow to be created.
6. After the creation, select the function instance just created and click **OK**.
7. Click **Save**.

## Operation Verification

1. Log in to the [COS console](#).
2. Click **Bucket List** on the left sidebar.
3. Click the target bucket to enter the bucket details page.
4. On the left sidebar, select **Data Processing Workflow > Workflow** to enter the workflow management page.
5. Enable the workflow just created, go to the specified bucket, upload a media file, and wait for the workflow to be executed.
6. After the workflow execution is completed:  
You can see that media processing succeeded, and the output file was generated.  
The storage class and custom header have been set for the source file.

# Direct Data Upload

## Practice of Direct Transfer for Web End

Last updated : 2024-09-29 12:07:08

### Overview

This document describes how to use simple code to upload files to a COS bucket directly from the web without using an SDK.

#### Note:

This document is based on the XML [APIs](#).

### Prerequisites

1. Log in to the [COS console](#) and create a bucket to obtain the `Bucket` (bucket name) and `Region` (region name). For more information, please see [Creating Buckets](#).
2. Go to the bucket detail page, choose the **Security Management** tab, and select **CORS (Cross-Origin Resource Sharing)** from the drop-down list. Then, click **Add a Rule**. A configuration example is shown in the following figure. For more information, please see [Setting Cross-Origin Access](#).



Origin \*

<http://qcloud.com>  
<http://a.qcloud.com>  
<http://b.qcloud.com>

Domain begins with http:// or https://. One domain per line. Up to one wildcard character \* is allowed in a line

Allow-Methods \*  PUT  GET  POST  DELETE  HEAD

Allow-Headers

\*

When you send an OPTIONS request, tell the server which custom HTTP request headers you can use for the next request, such as X-COS-META-

Expose-Headers

Etag

The Expose-header returns the usual cosine Header

Max-age \* 5 s

Options request gets the validity of the result, which must be a positive integer

Return Vary: Origin

Enabling the Vary: Origin Header option may result in increased browser access or CDN back-to-origin. Please set whether to return Vary: Origin Header depending on the situation. If the browser has both CORS and non-CORS requests, enable this option to avoid cross-domain problems.

Save Cancel

3. Log in to the [CAM console](#) and obtain the `SecretId` and `SecretKey` of your project.

## Solution Description

### Implementation Process

1. Select a file on the front end, and the front end will send the suffix to the server.
2. The server generates a random COS file path with time according to the suffix, calculates the corresponding signature, and returns the URL and signature information to the front end.
3. The front end uses a PUT or POST request to directly upload the file to COS.

### Solution Strength

Permissions security: The scope of security permissions can be effectively limited with the server-side signatures, which can only be used to upload a specified file path.

Path security: The random COS file path is determined by the server, which can effectively avoid the problem of existing files being overwritten and security risks.

## Practical Steps

### Configuring the Server to Implement Signatures

#### Note:

Add a layer of authority check on your website itself when the server is officially deployed.

Refer to the document [Request Signature](#) for how to calculate the signature.

Refer to [Nodejs Example](#) for the server-side calculation signature code using Nodejs.

## Web Upload Example

The following code is an example of [PUT Object](#) interface (recommended) and [POST Object](#) interface, the operation guide is as follows:

### Use AJAX PUT to upload

AJAX Upload requires the browser to support basic HTML5 features. The current solution uses [PUT Object](#) documentation. The operation guide is as follows:

1. Prepare the bucket configuration according to the steps of [Prerequisites](#).
2. Create a `test.html` file and copy the code below into the `test.html` file.
3. Deploy the back-end signature service and modify the signature service address in `test.html`.
4. Place `test.html` under the Web server, access the page through a browser, and test the file upload feature.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Ajax Put Upload (Server-side signature calculation)</title>
    <style>
      h1,
      h2 {
        font-weight: normal;
      }

      #msg {
        margin-top: 10px;
      }
    </style>
  </head>
  <body>
    <div id="msg">
      <h1>Ajax Put Upload</h1>
      <h2>Server-side signature calculation</h2>
    </div>
  </body>
</html>
```

```
    }
  </style>
</head>
<body>
  <h1>Ajax Put Upload (Server-side signature calculation)</h1>

  <input id="fileSelector" type="file" />
  <input id="submitBtn" type="submit" />

  <div id="msg"></div>

  <script>
    (function () {
      // url encode format for encoding more characters
      const camSafeUrlEncode = function (str) {
        return encodeURIComponent(str)
          .replace(/!/g, '%21')
          .replace(/'/g, '%27')
          .replace(/\\/g, '%28')
          .replace(/\\)/g, '%29')
          .replace(/\\*/g, '%2A');
      };

      // Calculate the signature.
      const getAuthorization = function (opt, callback) {
        // Replace with your server address to get the PUT upload signature, demo
        const url = http://127.0.0.1:3000/put-sign?ext=${opt.ext};
        const xhr = new XMLHttpRequest();
        xhr.open('GET', url, true);
        xhr.onload = function (e) {
          let credentials;
          try {
            const result = JSON.parse(e.target.responseText);
            credentials = result;
          } catch (e) {
            callback('Error in getting signature');
          }
          if (credentials) {
            // Print to confirm if the credentials are correct
            // console.log(credentials);
            callback(null, {
              securityToken: credentials.sessionToken,
              authorization: credentials.authorization,
              cosKey: credentials.cosKey,
              cosHost: credentials.cosHost,
            });
          } else {

```

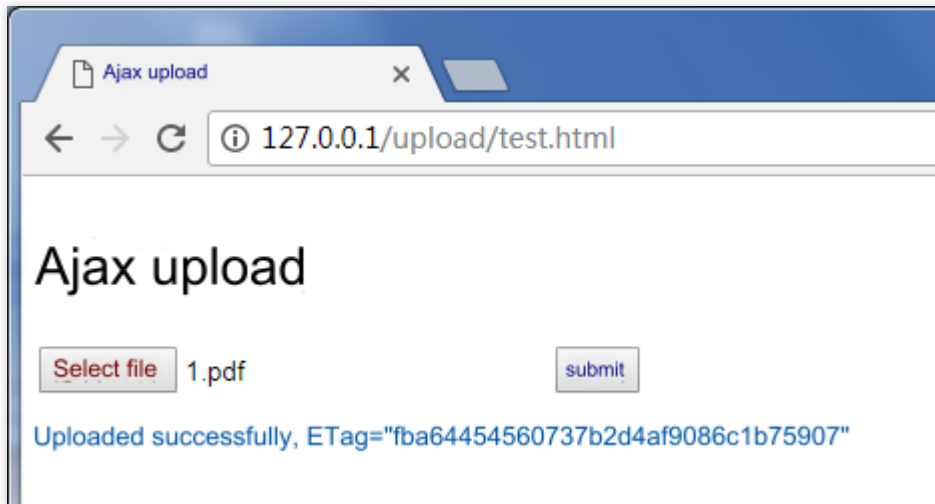
```
        console.error(xhr.responseText);
        callback('Error in getting signature');
    }
};
xhr.onerror = function (e) {
    callback('Error in getting signature');
};
xhr.send();
};

// Uploading Files
const uploadFile = function (file, callback) {
    const fileName = file.name;
    // Get the file extension
    let ext = '';
    const lastDotIndex = fileName.lastIndexOf('.');
    if (lastDotIndex > -1) {
        // Get the file extension here. The server generates the final upload p
        ext = fileName.substring(lastDotIndex + 1);
    }
    getAuthorization({ ext }, function (err, info) {
        if (err) {
            alert(err);
            return;
        }
        const auth = info.authorization;
        const securityToken = info.securityToken;
        const Key = info.cosKey;
        const protocol =
            location.protocol === 'https:' ? 'https:' : 'http:';
        const prefix = protocol + '//' + info.cosHost;
        const url =
            prefix + '/' + camSafeUrlEncode(Key).replace(/%2F/g, '/');
        const xhr = new XMLHttpRequest();
        xhr.open('PUT', url, true);
        xhr.setRequestHeader('Authorization', auth);
        securityToken &&
            xhr.setRequestHeader('x-cos-security-token', securityToken);
        xhr.upload.onprogress = function (e) {
            console.log(
                'Upload progress ' +
                Math.round((e.loaded / e.total) * 10000) / 100 +
                '%'
            );
        };
    });
    xhr.onload = function () {
        if (/^2\d\d$/.test('' + xhr.status)) {
```

```
        const ETag = xhr.getResponseHeader('etag');
        callback(null, { url: url, ETag: ETag });
    } else {
        callback('file' + Key + 'Upload failed, status code: ' + xhr.status
    }
};
xhr.onerror = function () {
    callback(
        'File ' + Key + ' Upload failed. Please check if the CORS cross-dom
    );
};
xhr.send(file);
});
};

// Listen for form submissions
document.getElementById('submitBtn').onclick = function (e) {
    const file = document.getElementById('fileSelector').files[0];
    if (!file) {
        document.getElementById('msg').innerText = 'No file selected for upload
        return;
    }
    file &&
        uploadFile(file, function (err, data) {
            console.log(err || data);
            document.getElementById('msg').innerText = err
                ? err
                : 'Upload successfully, ETag=' + data.ETag;
        });
};
})();
</script>
</body>
</html>
```

The execution effect is as shown below:



## Use AJAX POST to upload

AJAX Upload requires the browser to support basic HTML5 features. The current solution uses [Post Object](#) interface.

Operation guide:

1. Obtain the bucket information by taking the steps in [Prerequisites](#).
2. Create a `test.html` file and copy the code below into the `test.html` file.
3. Deploy the signature service at the backend and modify the signature service address in `test.html`.
4. Place `test.html` on the Web server. Then, browser the page to test the file upload feature.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Ajax Post Upload (Server-side signature calculation)</title>
    <style>
      h1,
      h2 {
        font-weight: normal;
      }

      #msg {
        margin-top: 10px;
      }
    </style>
  </head>
  <body>
    <h1>Post Object Upload (Server-side signature calculation)</h1>

    <input id="fileSelector" type="file" />
    <input id="submitBtn" type="submit" />
  </body>
</html>
```

```
<div id="msg"></div>

<script>
(function () {
  let prefix = '';
  let Key = '';

  // url encode format for encoding more characters
  const camSafeUrlEncode = function (str) {
    return encodeURIComponent(str)
      .replace(/!/g, '%21')
      .replace(/'/g, '%27')
      .replace(/\\(/g, '%28')
      .replace(/\\/g, '%29')
      .replace(/\\*/g, '%2A');
  };

  // Get permission policies
  const getAuthorization = function (opt, callback) {
    // Replace with your server address to get the post upload signature, dem
    const url = http://127.0.0.1:3000/post-policy?ext=${opt.ext};
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url, true);
    xhr.onload = function (e) {
      let credentials;
      try {
        const result = JSON.parse(e.target.responseText);
        credentials = result;
      } catch (e) {
        callback('Error in getting signature');
      }
      if (credentials) {
        // Print to confirm if the credentials are correct
        // console.log(credentials);
        callback(null, {
          securityToken: credentials.sessionToken,
          cosKey: credentials.cosKey,
          cosHost: credentials.cosHost,
          policy: credentials.policy,
          qAk: credentials.qAk,
          qKeyTime: credentials.qKeyTime,
          qSignAlgorithm: credentials.qSignAlgorithm,
          qSignature: credentials.qSignature,
        });
      } else {
        console.error(xhr.responseText);
        callback('Error in getting signature');
      }
    };
  };
});
```

```
    }
  };
  xhr.send();
};

// Uploading Files
const uploadFile = function (file, callback) {
  const fileName = file.name;
  // Get the file extension
  let ext = '';
  const lastDotIndex = fileName.lastIndexOf('.');
  if (lastDotIndex > -1) {
    // Get the file extension here. The server generates the final upload p
    ext = fileName.substring(lastDotIndex + 1);
  }
  getAuthorization({ ext }, function (err, credentials) {
    if (err) {
      alert(err);
      return;
    }
    const protocol =
      location.protocol === 'https:' ? 'https:' : 'http:';
    prefix = protocol + '//' + credentials.cosHost;
    Key = credentials.cosKey;
    const fd = new FormData();

    // Put an empty.html in the current directory so that the API can jump
    fd.append('key', Key);

    // Use policy signature protection formats
    credentials.securityToken &&
      fd.append('x-cos-security-token', credentials.securityToken);
    fd.append('q-sign-algorithm', credentials.qSignAlgorithm);
    fd.append('q-ak', credentials.qAk);
    fd.append('q-key-time', credentials.qKeyTime);
    fd.append('q-signature', credentials.qSignature);
    fd.append('policy', credentials.policy);

    // File content, place the file field at the end of the form to avoid l
    fd.append('file', file);

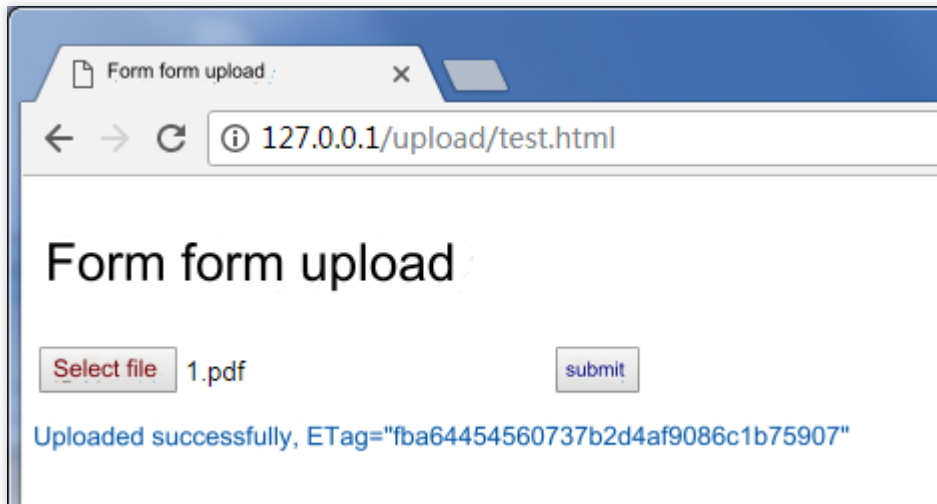
    // xhr
    const url = prefix;
    const xhr = new XMLHttpRequest();
    xhr.open('POST', url, true);
    xhr.upload.onprogress = function (e) {
      console.log(
```



```
        'Upload progress ' +
        Math.round((e.loaded / e.total) * 10000) / 100 +
        '%';
    });
};
xhr.onload = function () {
    if (Math.floor(xhr.status / 100) === 2) {
        const ETag = xhr.getResponseHeader('etag');
        callback(null, {
            url:
                prefix + '/' + camSafeUrlEncode(Key).replace(/%2F/g, '/'),
            ETag: ETag,
        });
    } else {
        callback('file' + Key + 'Upload failed, status code: ' + xhr.status
        );
    }
};
xhr.onerror = function () {
    callback(
        'File ' + Key + ' Upload failed. Please check if the CORS cross-domain
        ');
};
xhr.send(fd);
});
};

// Listen for form submissions
document.getElementById('submitBtn').onclick = function (e) {
    const file = document.getElementById('fileSelector').files[0];
    if (!file) {
        document.getElementById('msg').innerText = 'No file selected for upload
        return;
    }
    file &&
        uploadFile(file, function (err, data) {
            console.log(err || data);
            document.getElementById('msg').innerText = err
                ? err
                : 'Upload successfully, ETag=' + data.ETag + 'url=' + data.url;
        });
};
})();
</script>
</body>
</html>
```

The execution effect is as shown below:



## Use Form to upload

Form Upload supports uploads from older browsers (for example, IE8). The current solution uses [Post Object](#) interface. Operation guide:

1. Obtain the bucket information by taking the steps in [Prerequisites](#).
2. Create a `test.html` file and copy the code below into the `test.html` file.
3. Deploy the signature service at the backend and modify the signature service address in `test.html`.
4. In the directory where `test.html` is stored, create an empty `empty.html` file to be redirected back when the upload is successful.
5. Place `test.html` and `empty.html` on the Web server. Then, browse the page to test the file upload feature.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Simple Form Upload (Compatible with IE8) (Server-side Signature Calculat
    <style>
      h1,
      h2 {
        font-weight: normal;
      }
      #msg {
        margin-top: 10px;
      }
    </style>
  </head>
  <body>
    <h1>Simple Form Upload (Compatible with IE8) (Server-side Signature Calculation
    <div>Minimum compatibility with IE6 for uploading, and onprogress is not suppor
```

```
<form
  id="form"
  target="submitTarget"
  action=""
  method="post"
  enctype="multipart/form-data"
  accept="*/*"
>
<input id="name" name="name" type="hidden" value="" />
<input name="success_action_status" type="hidden" value="200" />
<input
  id="success_action_redirect"
  name="success_action_redirect"
  type="hidden"
  value=""
/>
<input id="key" name="key" type="hidden" value="" />
<input id="policy" name="policy" type="hidden" value="" />
<input
  id="q-sign-algorithm"
  name="q-sign-algorithm"
  type="hidden"
  value=""
/>
<input id="q-ak" name="q-ak" type="hidden" value="" />
<input id="q-key-time" name="q-key-time" type="hidden" value="" />
<input id="q-signature" name="q-signature" type="hidden" value="" />
<input name="Content-Type" type="hidden" value="" />
<input
  id="x-cos-security-token"
  name="x-cos-security-token"
  type="hidden"
  value=""
/>

<!-- Place the file field at the end of the form to avoid long file content a
<input id="fileSelector" name="file" type="file" />
<input id="submitBtn" type="button" value="Submit" />
</form>
<iframe
  id="submitTarget"
  name="submitTarget"
  style="display: none"
  frameborder="0"
></iframe>
```

```
<div id="msg"></div>

<script>
(function () {
  const form = document.getElementById('form');
  let prefix = '';

  // url encode format for encoding more characters
  const camSafeUrlEncode = function (str) {
    return encodeURIComponent(str)
      .replace(/!/g, '%21')
      .replace(/'/g, '%27')
      .replace(/\\/g, '%28')
      .replace(/\\\/g, '%29')
      .replace(/\\*/g, '%2A');
  };

  // Calculate the signature.
  const getAuthorization = function (opt, callback) {
    // Replace with your server address to get the post upload signature, dem
    const url = http://127.0.0.1:3000/post-policy?ext=${opt.ext || ''};
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url, true);
    xhr.onload = function (e) {
      let credentials;
      try {
        const result = JSON.parse(e.target.responseText);
        credentials = result;
      } catch (e) {
        callback('Error in getting signature');
      }
      if (credentials) {
        // Print to confirm if the credentials are correct
        // console.log(credentials);
        callback(null, {
          securityToken: credentials.sessionToken,
          cosKey: credentials.cosKey,
          cosHost: credentials.cosHost,
          policy: credentials.policy,
          qAk: credentials.qAk,
          qKeyTime: credentials.qKeyTime,
          qSignAlgorithm: credentials.qSignAlgorithm,
          qSignature: credentials.qSignature,
        });
      } else {
        console.error(xhr.responseText);
        callback('Error in getting signature');
      }
    };
  };
});
```

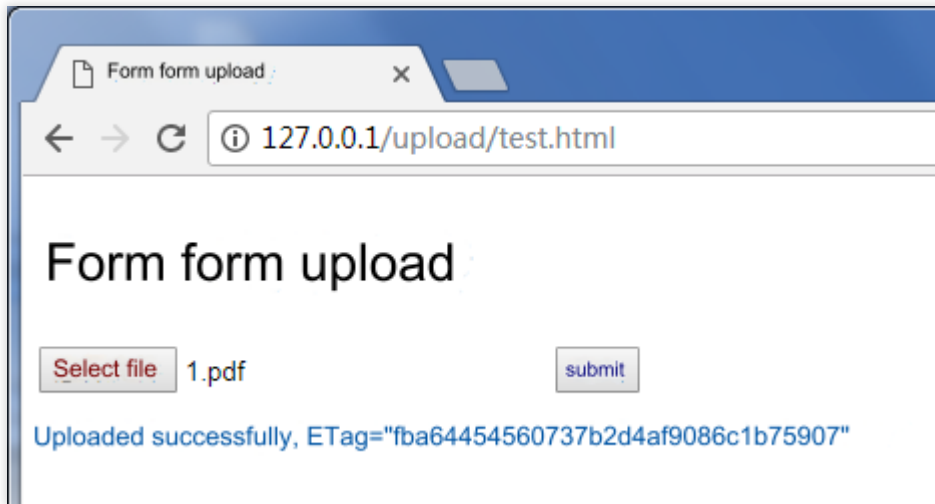
```
    }
  };
  xhr.send();
};

// Listen upload completion
let Key;
const submitTarget = document.getElementById('submitTarget');
const showMessage = function (err, data) {
  console.log(err || data);
  document.getElementById('msg').innerText = err
    ? err
    : 'Upload successfully, ETag=' + data.ETag;
};
submitTarget.onload = function () {
  let search;
  try {
    search = submitTarget.contentWindow.location.search.substr(1);
  } catch (e) {
    showMessage('file' + Key + 'Upload failed');
  }
  if (search) {
    const items = search.split('&');
    let i = 0;
    let arr = [];
    const data = {};
    for (i = 0; i < items.length; i++) {
      arr = items[i].split('=');
      data[arr[0]] = decodeURIComponent(arr[1] || '');
    }
    showMessage(null, {
      url: prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/'),
      ETag: data.etag,
    });
  } else {
  }
};

// Start uploading
document.getElementById('submitBtn').onclick = function (e) {
  const filePath = document.getElementById('fileSelector').value;
  if (!filePath) {
    document.getElementById('msg').innerText = 'No file selected for upload';
    return;
  }
  // Get the file extension
  let ext = '';
```

```
const lastDotIndex = filePath.lastIndexOf('.');
if (lastDotIndex > -1) {
    // Get the file extension here. The server generates the final upload p
    ext = filePath.substring(lastDotIndex + 1);
}
getAuthorization({ ext }, function (err, AuthData) {
    if (err) {
        alert(err);
        return;
    }
    const protocol =
        location.protocol === 'https:' ? 'https:' : 'http:';
    prefix = protocol + '//' + AuthData.cosHost;
    form.action = prefix;
    Key = AuthData.cosKey;
    // Put an empty.html in the current directory so that the API can jump
    document.getElementById('success_action_redirect').value =
        location.href.substr(0, location.href.lastIndexOf('/') + 1) +
        'empty.html';
    document.getElementById('key').value = AuthData.cosKey;
    document.getElementById('policy').value = AuthData.policy;
    document.getElementById('q-sign-algorithm').value =
        AuthData.qSignAlgorithm;
    document.getElementById('q-ak').value = AuthData.qAk;
    document.getElementById('q-key-time').value = AuthData.qKeyTime;
    document.getElementById('q-signature').value = AuthData.qSignature;
    document.getElementById('x-cos-security-token').value =
        AuthData.securityToken || '';
    form.submit();
});
};
})();
</script>
</body>
</html>
```

The execution effect is as shown below:



## Restricting the File Type and the File Size During Upload

### File Types Limited by Front End

Refer to the above AJAX PUT upload, add a layer of judgment when selecting files. (Only restricted file extensions are supported)

```
// Omit other codes
document.getElementById('submitBtn').onclick = function (e) {
  const file = document.getElementById('fileSelector').files[0];
  if (!file) {
    document.getElementById('msg').innerText = 'No file selected for upload';
    return;
  }
  // Get the file extension
  const fileName = file.name;
  const lastDotIndex = fileName.lastIndexOf('.');
  const ext = lastDotIndex > -1 ? fileName.substring(lastDotIndex + 1) : '';

  // Please replace with the formats you want to restrict, for example, only jpg and
  const allowExt = ['jpg', 'png'];
  if (!allowExt.includes(ext)) {
    alert('Only jpg and png files are supported for upload');
    return;
  }

  file &&
  uploadFile(file, function (err, data) {
    console.log(err || data);
    document.getElementById('msg').innerText = err
      ? err
      : 'Upload successfully, ETag=' + data.ETag + 'url=' + data.url;
  });
}
```

```
});  
};
```

### File Size Limited by Front End

Refer to the above AJAX PUT upload, add a layer of judgment when selecting files.

```
// Omit other codes  
document.getElementById('submitBtn').onclick = function (e) {  
  const file = document.getElementById('fileSelector').files[0];  
  if (!file) {  
    document.getElementById('msg').innerText = 'No file selected for upload';  
    return;  
  }  
  const fileSize = file.size;  
  
  // Please replace with the object size you want to restrict, up to a maximum of 5  
  if (fileSize > 5 * 1024 * 1024) {  
    alert('The selected file exceeds 5MB, please selected another one');  
    return;  
  }  
  
  file &&  
  uploadFile(file, function (err, data) {  
    console.log(err || data);  
    document.getElementById('msg').innerText = err  
      ? err  
      : 'Upload successfully, ETag=' + data.ETag + 'url=' + data.url;  
  });  
};
```

### Server-Side Signature Restriction

Refer to the server-side signature code [Nodejs Example](#) for put-sign-limit and post-policy-limit.

## Reference

If you need to call more APIs, please see the following JavaScript SDK document:

[JavaScript SDK](#)



# Practice of Direct Upload Through WeChat Mini Program

Last updated : 2024-03-25 15:16:26

## Introduction

This document describes how to directly transfer files to a Cloud Object Storage (COS) bucket in a Mini Program with simple code without relying on the SDK.

### Note:

The content of this document is based on the XML version of the API.

## Preconditions

1. Log in to the [COS console](#) and create a bucket with `BucketName` (bucket name) and `Region` (region name) specified. For more information, see [Creating Buckets](#).
2. Log in to the [CAM console](#) and obtain your project's SecretId and SecretKey on the API key management page.
3. Configure the applet domain name whitelist

To request COS in the Mini Program, you need to log in to the WeChat public platform, and configure the domain name whitelist in "Development" -> "Development Settings". The SDK uses two interfaces: `wx.uploadFile` and `wx.request`.

`cos.postObject` uses `wx.uploadFile` to send the request.

Other methods use `wx.request` to send requests.

Both need to configure the COS domain name in the corresponding white list. There are two formats of domain names configured in the whitelist:

If only one bucket is used, you can configure the Bucket domain name as a whitelist domain name, for example

```
examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com .
```

If multiple storage buckets are used, you can choose the suffix method to request COS, and place the bucket in the pathname to request. This method needs to configure the regional domain name as a whitelist, such as `cos.ap-guangzhou.myqcloud.com` . For details, please refer to the comments in the code below.

## plan description

### Implementation process

1. Select a file on the front end, and the front end will send the suffix to the server.
2. The server generates a random COS file path with time according to the suffix, calculates the corresponding signature, and returns the URL and signature information to the front end.
3. The front end uses a PUT or POST request to directly upload the file to COS.

## Solution Advantages

**Permissions security:** Using server-side signatures can effectively limit the scope of security permissions, which can only be used to upload a specified file path.

**Path security:** The random COS file path is determined by the server, which can effectively avoid the problem of existing files being overwritten and security risks.

## Suffix request

The general request format of COS API is similar to `POST http://examplebucket-1250000000.cos.ap-beijing.myqcloud.com/` , and the requested domain name is the bucket domain name. In this way, if multiple storage buckets are used in the applet, you need to configure the bucket domain name as a whitelist domain name.

The solution is as follows:

COS provides a postfix request format `POST http://cos.ap-beijing.myqcloud.com/examplebucket-1250000000/` , the requested domain name is the regional domain name, and the bucket name is placed in the requested path. To use multiple storage buckets in the same region in the applet, you only need to configure a domain name `cos.ap-beijing.myqcloud.com` as the whitelist domain name.

Note that the suffix request format needs to be noted that the path used when signing should be prefixed with the bucket name, for example `/examplebucket-1250000000/` .

## Practical steps

### Configure the server to implement signature

#### Note:

Please add a layer of authority check on your website itself when the server is officially deployed.

How to calculate the signature can refer to the document [Request Signature](#).

Please refer to [Nodejs Example](#) for the server-side calculation signature code using Nodejs.

### Small program upload example

The following code is an example of [PUT Object](#) interface (recommended) and [POST Object](#) interface, the operation guide is as follows:

#### Upload using POST

```
var uploadFile = function () {
  // url encode format for encoding more characters
  var camSafeUrlEncode = function (str) {
    return encodeURIComponent(str)
      .replace(/!/g, '%21')
      .replace(/'/g, '%27')
      .replace(/\\/g, '%28')
      .replace(/\\/g, '%29')
      .replace(/\\*/g, '%2A');
  };

  // get the signature
  var getAuthorization = function (options, callback) {
    wx.request({
      method: 'GET',
      // Replace it with your own server address to get the post upload signature
      url: 'http://127.0.0.1:3000/post-policy?ext=' + options.ext,
      dataType: 'json',
      success: function (result) {
        var data = result.data;
        if (data) {
          callback(data);
        } else {
          wx.showModal({
            title: 'Failed to obtain temporary key',
            content: JSON.stringify(data),
            showCancel: false,
          });
        }
      },
      error: function (err) {
        wx.showModal({
          title: 'Failed to obtain temporary key',
          content: JSON.stringify(err),
          showCancel: false,
        });
      },
    });
  };

  var postFile = function ({ prefix, filePath, key, formData }) {
    var requestTask = wx.uploadFile({
      url: prefix,
      name: 'file',
      filePath: filePath,
      formData: formData,
    });
  };
};
```

```
success: function (res) {
  var url = prefix + '/' + camSafeUrlEncode(key).replace(/%2F/g, '/');
  if (res.statusCode === 200) {
    wx.showModal({ title: 'Uploaded successfully', content: url, showCancel:
  } else {
    wx.showModal({
      title: 'Upload failed',
      content: JSON.stringify(res),
      showCancel: false,
    });
  }
  console.log(res.header['x-cos-request-id']);
  console.log(res.statusCode);
  console.log(url);
},
fail: function (res) {
  wx.showModal({
    title: 'Upload failed',
    content: JSON.stringify(res),
    showCancel: false,
  });
},
});
requestTask.onProgressUpdate(function (res) {
  console.log('Progress:', res);
});
};

// upload files
var uploadFile = function (filePath) {
  var extIndex = filePath.lastIndexOf('.');
  var fileExt = extIndex >= -1 ? filePath.substr(extIndex + 1) : '';
  getAuthorization({ ext: fileExt }, function (AuthData) {
    // Parameters used in the request
    var prefix = 'https://' + AuthData.cosHost;
    var key = AuthData.cosKey; // It is safer to let the server decide the file
    var formData = {
      key: key,
      success_action_status: 200,
      'Content-Type': '',
      'q-sign-algorithm': AuthData.qSignAlgorithm,
      'q-ak': AuthData.qAk,
      'q-key-time': AuthData.qKeyTime,
      'q-signature': AuthData.qSignature,
      policy: AuthData.policy,
    };
    if (AuthData.securityToken)
```

```
        formData['x-cos-security-token'] = AuthData.securityToken;
        postFile({ prefix, filePath, key, formData });
    });
};

// Select a document
wx.chooseMedia({
    count: 1, // default 9
    sizeType: ['original'], // You can specify whether it is the original image or
    sourceType: ['album', 'camera'], // You can specify whether the source is an a
    success: function (res) {
        uploadFile(res.tempFiles[0].tempFilePath);
    },
});
};
```

## Upload using PUT

```
var uploadFile = function () {
    // url encode format for encoding more characters
    var camSafeUrlEncode = function (str) {
        return encodeURIComponent(str)
            .replace(/!/g, '%21')
            .replace(/'/g, '%27')
            .replace(/\\/g, '%28')
            .replace(/\\/g, '%29')
            .replace(/\\*/g, '%2A');
    };

    // get the signature
    var getAuthorization = function (options, callback) {
        wx.request({
            method: 'GET',
            // Replace it with your own server address to get the put upload signature
            url: 'http://127.0.0.1:3000/put-sign?ext=' + options.ext,
            dataType: 'json',
            success: function (result) {
                var data = result.data;
                if (data) {
                    callback(data);
                } else {
                    wx.showModal({
                        title: 'Failed to obtain temporary key',
                        content: JSON.stringify(data),
                        showCancel: false,
                    });
                }
            }
        });
    };
};
```

```
    }
  },
  error: function (err) {
    wx.showModal({
      title: 'Failed to obtain temporary key',
      content: JSON.stringify(err),
      showCancel: false,
    });
  },
});
};

var putFile = function ({ prefix, filePath, key, AuthData }) {
  // put upload needs to read the real content of the file to upload
  const wxfs = wx.getFileSystemManager();
  wxfs.readFile({
    filePath: filePath,
    success: function (fileRes) {
      var requestTask = wx.request({
        url: prefix + '/' + key,
        method: 'PUT',
        header: {
          Authorization: AuthData.authorization,
          'x-cos-security-token': AuthData.securityToken,
        },
        data: fileRes.data,
        success: function success(res) {
          var url = prefix + '/' + camSafeUrlEncode(key).replace(/%2F/g, '/');
          if (res.statusCode === 200) {
            wx.showModal({
              title: 'Uploaded successfully',
              content: url,
              showCancel: false,
            });
          } else {
            wx.showModal({
              title: 'Upload failed',
              content: JSON.stringify(res),
              showCancel: false,
            });
          }
          console.log(res.statusCode);
          console.log(url);
        },
        fail: function fail(res) {
          wx.showModal({
            title: 'Upload failed',
```

```
        content: JSON.stringify(res),
        showCancel: false,
    });
},
});
requestTask.onProgressUpdate(function (res) {
    console.log('Progress:', res);
});
},
});
};

// upload files
var uploadFile = function (filePath) {
    var extIndex = filePath.lastIndexOf('.');
    var fileExt = extIndex >= -1 ? filePath.substr(extIndex + 1) : '';
    getAuthorization({ ext: fileExt }, function (AuthData) {
        const prefix = 'https://' + AuthData.cosHost;
        const key = AuthData.cosKey;
        putFile({ prefix, filePath, key, AuthData });
    });
};

// Select a document
wx.chooseMedia({
    count: 1, // default 9
    sizeType: ['original'], // You can specify whether it is the original image or
    sourceType: ['album', 'camera'], // You can specify whether the source is an a
    success: function (res) {
        uploadFile(res.tempFiles[0].tempFilePath);
    },
});
};
```

## Related documents

To use the Mini Program SDK, please refer to the Mini Program SDK [Quick Start](#) document.

# Practice of Direct Upload for Mobile Apps

Last updated : 2024-03-25 15:16:26

## Overview

This document describes how to achieve direct file upload with Tencent Cloud COS for your mobile applications. COS makes it easy for you to store files and data as you only need to generate and manage the access key on your server.

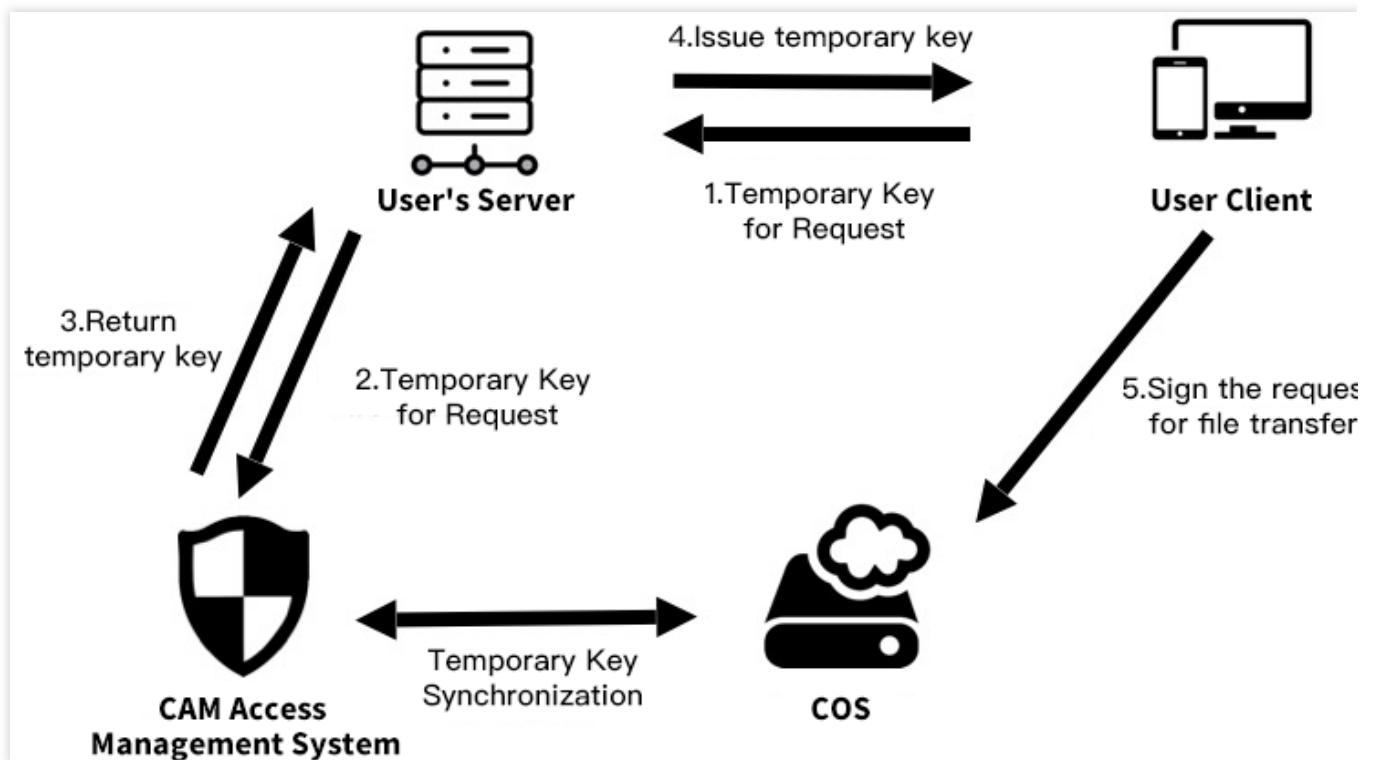
## Architecture

For client applications, storing permanent keys in the code increases the risk that your credentials could leak and makes it difficult to control access permissions. We recommend that your applications use temporary keys with a specified validity period to access your COS resources to prevent credential leakage.

The COS Mobile SDK (Android/iOS) allows you to apply temporary keys on COS access request authorization. You only need to set up the temporary key service on the backend to authorize the requests.

### Workflow

The diagram below illustrates how to enable CAM for the COS:





Notes:

User's client: user's mobile App.

COS: [Cloud Object Storage](#), which stores the data uploaded from the App.

CAM: [Cloud Access Management](#), which is used to generate temporary keys for COS.

User's server: user's backend server, which is used to obtain the temporary keys and return them to the App.

## Prerequisites

1. Create a bucket.

Create a bucket using the [COS Console](#). Depending on your needs, set your bucket permission to private read/write, or public read and private write. For detailed directions, see [Creating Buckets](#) and [Setting Access Permission](#).

2. Obtain the permanent key.

Temporary keys are generated with the permanent key. Get the SecretId and SecretKey on the [API Key Management](#) page. Get the APPID in the [Account Center](#).

## Directions

### Setting up temporary key service

For security purposes, we recommend you use temporary keys to calculate a signature. To create and use temporary keys, you need to set up the temporary key service on your server and an API for your application. For more information, see [Temporary Key Generation and Usage Guide](#).

#### Note:

In official deployment, add a layer of permission check of your website on the server side.

### Defining permissions

We recommended that you define a scope for the permission granted to temporary keys through Policy based on the principle of "least privilege". A key that has all permissions to read and write the data increases the risk that your other data could leak. For more information, see [Temporary Key Generation and Usage Guide](#).

### Integrating SDK with authorization service

#### Android

After the temporary key service is set up, you integrate the SDK with the authorization service. Instead of you managing the temporary keys, the SDK controls the number of concurrent requests to process, cache the valid keys on your local devices, and requests new keys to replace the expired ones.

## Authorization with standard response body

If you use the JSON data obtained from the STS SDK as a response body for the temporary key service (similar to `cosign`), you can create an authorization class in the COS SDK using the following code:

```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;

Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;

/**
 * Get the authorization service URL
 */
URL url = null; // URL of the backend authorization service
try{
    url = new URL("your_auth_server_url");
} catch (MalformedURLException e) {
    e.printStackTrace();
}

/**
 * Initialize the {@link QCloudCredentialProvider} object to provide a
temporary key to the SDK.
 */
QCloudCredentialProvider credentialProvider = new SessionCredentialProvider(new
HttpRequest.Builder<String>()
    .url(url)
    .method("GET")
    .build());

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig,
credentialProvider);
```

### Note:

Because this method requires that the signature start time matches the local time on your mobile phone. A large difference (more than 10 minutes) between the time on your phone and the correct local time may cause a signature error. In this case, we recommend you use the custom response body for authorization as described below:

## Authorization with custom response body

For higher flexibility, you can inherit the `BasicLifecycleCredentialProvider` class and implement its `fetchNewCredentials()` for custom configurations. For example, you can customize the HTTP response body for the temporary key service to return the server time to the device as the signature start time so as to avoid signature error caused by big device time difference. You can also choose to use other protocols for the communication between the end device and the service side.

Define a `MyCredentialProvider` class first:

```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;

public class MyCredentialProvider extends BasicLifecycleCredentialProvider {

    @Override
    protected QCloudLifecycleCredentials fetchNewCredentials() throws QCloudClientException {

        // First, get the response containing a signature from your temporary key service.
        ....

        // Then, parse the response to get the key.
        String tmpSecretId = ...;
        String tmpSecretKey = ...;
        String sessionToken = ...;
        long expiredTime = ...;

        // Return server time as the start time of the signature.
        long beginTime = ...;

        // todo something you want

        // Finally return the temporary key info.
        return new SessionQCloudCredentials(tmpSecretId, tmpSecretKey, sessionToken, expiredTime, beginTime);
    }
}
```

Authorize the request with your custom `MyCredentialProvider` instance:

```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
```

```
import java.net.*;

Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;

/**
 * Initialize the {@link QCloudCredentialProvider} to provide a temporary key
 * to the SDK.
 */
QCloudCredentialProvider credentialProvider = new MyCredentialProvider();

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig,
credentialProvider);
```

For more information on how files are uploaded and downloaded between an Android device and COS, see [Getting Started with Android SDK](#).

## iOS

We provide `QCloudCredentialFenceQueue` to make it easier for you to obtain and manage temporary signatures. With `QCloudCredentialFenceQueue`, a request for signature will be processed only after the signature process is completed.

To use `QCloudCredentialFenceQueue`, you need to first create an instance.

```
//AppDelegate.m
//AppDelegate must follow QCloudCredentialFenceQueueDelegate protocol
//
- (BOOL)application:(UIApplication * )application
didFinishLaunchingWithOptions:(NSDictionary * )launchOptions {
    // init step
    self.credentialFenceQueue = [QCloudCredentialFenceQueue new];
    self.credentialFenceQueue.delegate = self;
    return YES;
}
```

Next, to call the `QCloudCredentialFenceQueue` class, you must follow `QCloudCredentialFenceQueueDelegate` and implement the method defined in the protocol:

```
- (void) fenceQueue:(QCloudCredentialFenceQueue * )queue
requestCreatorWithContinue:(QCloudCredentialFenceQueueContinue) continueBlock
```

When you obtain the signature using `QCloudCredentialFenceQueue`, all the requests in the SDK that need signatures will not be performed until the parameters required for the signature are obtained via the method defined by the protocol and a valid signature is generated. See the example below:

```

- (void) fenceQueue: (QCloudCredentialFenceQueue *) queue
requestCreatorWithContinue: (QCloudCredentialFenceQueueContinue) continueBlock {
    QCloudHTTPRequest* request = [QCloudHTTPRequest new];
    request.requestData.serverURL = @"your sign service url";//Request URL

    [request setConfigureBlock:^(QCloudRequestSerializer *requestSerializer,
QCloudResponseSerializer *responseSerializer) {
        requestSerializer.serializerBlocks =
@[QCloudURLFuseWithURLEncodeParamters];
        responseSerializer.serializerBlocks =
@[QCloudAcceptRespnseCodeBlock([NSSet setWithObjects:@(200), nil],nil),//An
error is returned when the return code is other than 200.

QCloudResponseJSONSerilizerBlock];//Return element parsed in JSON format
    }];

    [request setFinishBlock:^(id response, NSError *error) {
        if (error) {
            error = [NSError errorWithDomain:@"com.tac.test" code:-1111
userInfo:@{NSLocalizedStringKey:@"No temporary key is obtained."}];
            continueBlock(nil, error);
        } else {
            QCloudCredential* credential = [[QCloudCredential alloc] init];
            credential.secretID = response[@"data"][@"credentials"]
[@"tmpSecretId"];
            credential.secretKey = response[@"data"][@"credentials"]
[@"tmpSecretKey"];
            credential.startDate = [NSDate
dateWithTimeIntervalSince1970:@"Returned server time"]
            credential.experationDate = [NSDate dateWithTimeIntervalSinceNow:
[response[@"data"][@"expiredTime"] intValue]];
            credential.token = response[@"data"][@"credentials"]
[@"sessionToken"];
            QCloudAuthenticationV5Creator* creator =
[[QCloudAuthenticationV5Creator alloc] initWithCredential:credential];
            continueBlock(creator, nil);

        }
    }];
    [[QCloudHTTPSessionManager sharedInstance] performRequest:request];
}

```

For more information on how files are uploaded and downloaded between an iOS device and COS, see [Getting Started With iOS SDK](#).

## Demo code

## Android

Download the Demo by clicking [here](#), or scanning QR code with your Android mobile browser:



## iOS

For complete sample project for iOS, see [COS iOS Demo](#).

Modify the file `QCloudCOSXMLDemo/QCloudCOSXMLDemo/key.json` by adding your APPID, secretID, and secretKey, and run this command:

```
pod install
```

### Note:

Get your APPID, secretID, and secretKey from the [API Key Management](#) page.

After executing the command, open `QCloudCOSXMLDemo.xcworkspace` to view the Demo.

# uni-app Direct Upload Practice

Last updated : 2024-03-25 15:16:26

## Overview

This document describes how to use simple code to upload files to a COS bucket directly via uni-app without using an SDK.

### Note:

This document is based on the XML API [PostObject](#).

## Solution

### Directions

1. Select a file in the frontend, and the frontend sends the file extension to the server.
2. The server generates a random COS file path with time according to the file extension, calculates the corresponding PostObject policy signature, and returns the URL and signing information to the frontend.
3. The frontend calls the [PostObject API](#) to upload the file to COS.

### Strengths

Permission security: The PostObject policy signature effectively limits the security permission scope and can only be used to upload to a specified file path.

Path security: The server determines the random COS file path, which effectively avoids the problem of overwriting existing files and security risks.

Multi-terminal compatibility: The file selection and upload APIs provided by uni-app can be used to make the same code compatible with multiple terminals (web, mini programs, or apps).

## Prerequisites

1. Log in to the [COS console](#) and create a bucket to obtain the `Bucket` (bucket name) and `Region` (region name). For more information, please see [Creating Buckets](#).
2. Log in to the [CAM console](#) and obtain the `SecretId` and `SecretKey` of your project.
3. Go to the details page of the newly created bucket, choose **Security Management > CORS (Cross-Origin Resource Sharing)**, and click **Add a Rule**. A configuration example is shown in the following figure. For more information, see [Setting Cross-Origin Access](#).

## Directions

### Note:

Before formal deployment, it is recommended that you add a permission verification step for your website itself on the server side.

### Frontend upload

1. Implement a server API to generate random file paths, calculate signatures, and return them to the frontend. For implementation details, see the [post-policy example](#).

2. Use HBuilderX's default template to create a uni-app app.

The created app is a Vue project.

3. Copy the following code to replace the content of the `pages/index/index.vue` file and modify the `post-policy` API call link to make it point to your server address (the server API implemented in step 1).

```
<template>
<view class="content">
  <button type="default" @click="selectUpload">Select file upload</button>
  <image v-if="fileUrl" class="image" :src="fileUrl"></image>
</view>
</template>

<script>
export default {
  data() {
    return {
      title: 'Hello',
      fileUrl: ''
    };
  },
  onLoad() {

  },
  methods: {
    selectUpload() {

      var vm = this;

      // URL-encode more characters.
      var camSafeUrlEncode = function (str) {
        return encodeURIComponent(str)
          .replace(/!/g, '%21')
          .replace(/'/g, '%27')
          .replace(/\\/g, '%28')
          .replace(/\\/g, '%29')
      };
    }
  }
};
```



```
        .replace(/\\/g, '%2A');
    };

    // Get the upload path and credential
    var getUploadInfo = function (extName, callback) {
        // Pass in the file extension to enable the backend to generate a random
        // Refer to the server example at https://github.com/tencentyun/cos-demo
        uni.request({
            url: 'http://127.0.0.1:3000/post-policy?ext=' + extName,
            success: (res) => {
                callback && callback(null, res.data.data);
            },
            error(err) {
                callback && callback(err);
            },
        });
    };

    // Initiate an upload request, and the upload uses the `PostObject` API and
    // API documentation: https://intl.cloud.tencent.com/document/product/436/
    var uploadFile = function (opt, callback) {
        var formData = {
            key: opt.cosKey,
            policy: opt.policy, // Base64 policy string
            success_action_status: 200,
            'q-sign-algorithm': opt.qSignAlgorithm,
            'q-ak': opt.qAk,
            'q-key-time': opt.qKeyTime,
            'q-signature': opt.qSignature,
        };
        // If the server uses a temporary key for calculation, you need to pass it
        if (opt.securityToken) formData['x-cos-security-token'] = opt.securityToken;
        uni.uploadFile({
            url: 'https://' + opt.cosHost, // This is only an example, not the real
            filePath: opt.filePath,
            name: 'file',
            formData: formData,
            success: (res) => {
                if (![200, 204].includes(res.statusCode)) return callback && callback(
                    'Upload failed: ' + res.statusCode);
                var fileUrl = 'https://' + opt.cosHost + '/' + encodeURIComponent(opt.filePath);
                callback && callback(null, fileUrl);
            },
            error(err) {
                callback && callback(err);
            },
        });
    };
};
```

```
// Select the file
uni.chooseImage({
  success: (chooseImageRes) => {
    var file = chooseImageRes.tempFiles[0];
    if (!file) return;
    // Get the path of the local file to upload
    var filePath = chooseImageRes.tempFilePaths[0];
    // Get the extension of the file to upload and then the backend can
    var fileName = file.name;
    var lastIndex = fileName.lastIndexOf('.');
    var extName = lastIndex > -1 ? fileName.slice(lastIndex + 1) : '';
    // Get the domain name, path, and credential for pre-upload
    getUploadInfo(extName, function (err, info) {
      // Upload the file
      info.filePath = filePath;
      uploadFile(info, function (err, fileUrl) {
        vm.fileUrl = fileUrl;
      });
    });
  }
});
},
}
}
</script>

<style>
.content {
  padding: 20px 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

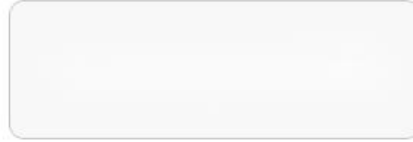
.image {
  margin-top: 20px;
  margin-left: auto;
  margin-right: auto;
}
</style>
```

4. In HBuilderX, choose **Run > Run to browser > Chrome**. Then you can select files to upload in the browser.

The execution result is as shown below:

Direct upload effect:

## uni-app



## Demo Code Address

Demo download address: [upload-demo](#)

# Content Moderation

## Blocking CDN Cache Based on Moderation Result

Last updated : 2024-03-25 15:16:26

### Overview

The content moderation feature can automatically block non-compliant files. It only applies to data stored on COS origins rather than data cached on CDN nodes.

This document describes how to promptly block non-compliant data cached in CDN through SCF and API Gateway.

### Directions

1. Log in to the [SCF console](#), select **Functions**, and click **Create** to create a function.

2. Select **Create from scratch** and set the following basic configuration items:

Function type: Select **Event-triggered function**.

Function name: Enter a custom function name.

Region: Select the region of the bucket with the moderation feature enabled.

Runtime environment: Select **Python 2.7**.

3. Configure the function code as follows:

Submitting method: Select **Local ZIP file** or **Upload a ZIP pack via COS**. You can download the ZIP package [here](#).

Execution: Enter **index.main\_handler**.

4. Click **Advanced configuration** to configure **Environment configuration**. You can modify all configuration items except the environment variables as needed.

Resource type: CPU

MEM: 512 MB

Initialization timeout period: 65

Execution timeout period: 30

Environment variable:

CI\_AUDITING\_CALLBACK: Enter the callback address configured in the callback settings of content moderation here. The callback will be performed only if the callback address is set.

CDN\_URL: Set the required CDN address to purge the CDN cache.

REGION: Set the required region where the bucket resides.

**BUCKET\_ID:** Set the required bucket ID (i.e., bucket name), which is used to query the image style. Entering an incorrect value will result in an error during style query.

**IMAGE\_STYLE\_SEPARATORS:** Set the image style separator if you want to purge image style. You can enter multiple separators consecutively with no need to separate them.

**CDN\_REFRESH\_TYPE:** Set the image object cache purge method, which is purge by URL by default (that is, only the style will be purged). If you set this variable to **path**, the cache accessed by image processing parameters will be purged. Note that purge by path will remove files with longer filenames containing this filename.

Be sure to configure the above environment variables correctly so that cache purge can work as expected.

5. Select **Execution Role** for **Permission configuration** and click **Create execution role** to enter the **Create custom role** page.

6. Select **Serverless Cloud Function (SCF)** as the role entity and click **Next**.

7. Select the `QcloudCDNFullAccess` and `QcloudCIReadOnlyAccess` role policies and click **Next**.

8. Name the role and click **Complete**.

9. After creating the custom role, go back to the function creation page, refresh the role drop-down list, and select the newly created role.

10. Click **Complete**.

11. Go to the [API Gateway console](#) to activate the API Gateway service.

12. Create an API gateway as instructed in [Creating APIs Connecting to the SCF Backend](#).

13. Select **Publish** for **Release Environment** and click **Publish service**.

14. On the content moderation page in the CI console, set callback parameters for image or other target types, and set the address of the API gateway created in the previous step as the callback URL.

15. After the callback is configured, resource cache in CDN will be automatically purged based on the moderation callback result.

# Data Security

## Introduction to COS Data Security Solution

Last updated : 2024-03-25 15:16:26

### Pre-Event Protection

#### 1. Isolating permissions

An in-cloud enterprise should pay attention to account security and resource authorization to protect the security system. To manage in-cloud resources properly, the following risks should be avoided:

Using the Tencent Cloud root account for routine operations

Over-authorizing employees' sub-accounts

No access control over risky operations and high-permission sub-accounts

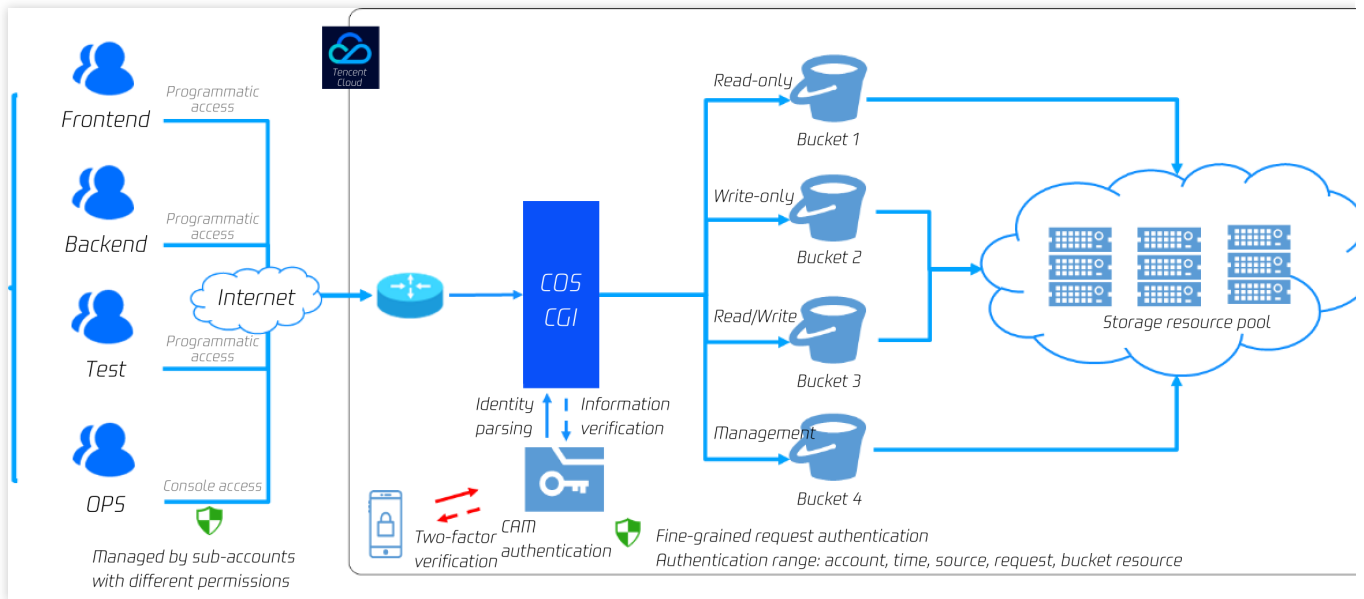
Failure to regularly audit user permissions and login information

No regulation for permission management

With Tencent Cloud Access Management (CAM), users can set levels for accounts and permissions for clearer and securer permission management. As for account levels, the root account can grant permissions, such as programmatic access and console access, to all valid CAM users (e.g., sub-accounts and collaborators). As for permission levels, you can grant service-level, API-level, resource-level, and other levels of permissions to CAM users. In this way, you can specify the operations a user can perform and the methods and resources the user can use under a specific condition.

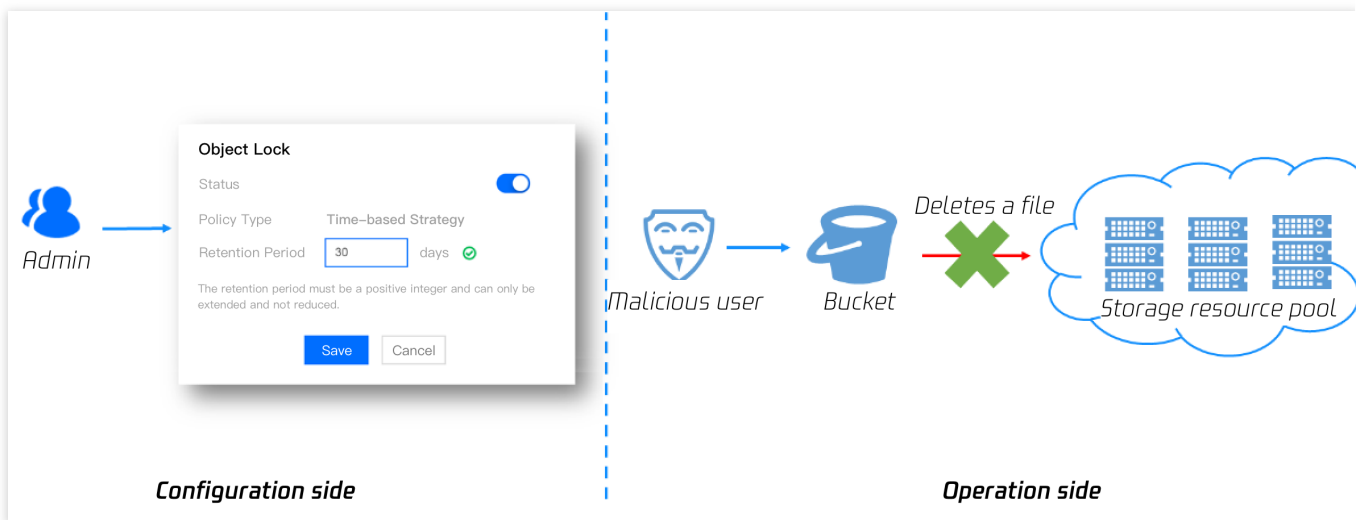
For example, you can create a sub-account and grant it permissions to manage the root account's resources without sharing the identity credential of the root account. Also, different access permissions to different resources can be granted to different users. For example, some sub-accounts can be granted the read access permission to a COS bucket, while some other sub-accounts or root accounts can own the write access to a COS object. The aforementioned resources, access permissions, and users can all be managed in batches to facilitate refined permission management.

You can also limit risky operations (for example, data deletion) to the console and enable MFA for two-factor verification at the same time. After MFA is enabled, risky operations will trigger an SMS verification code.



## 2. Locking objects

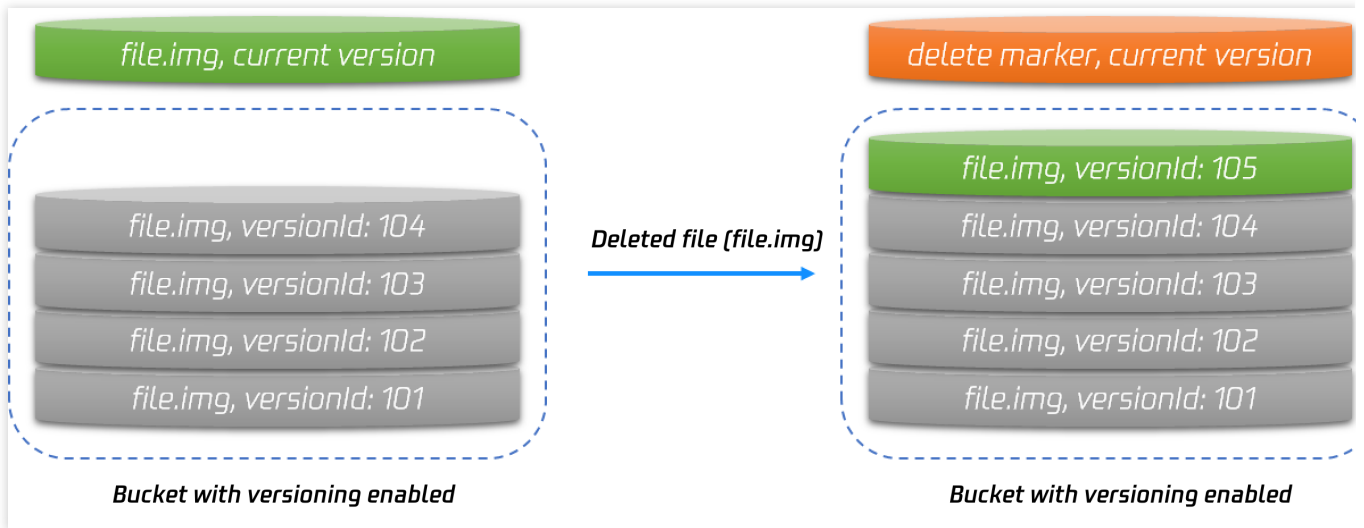
Users can enable the object locking feature for sensitive and essential data (e.g., financial transaction data and medical image data) to prevent them from being deleted or modified. After the object locking feature is enabled, all data in the bucket can only be read and cannot be overwritten or deleted during the effective period. This setting applies to all CAM users (including the root account) and anonymous users.



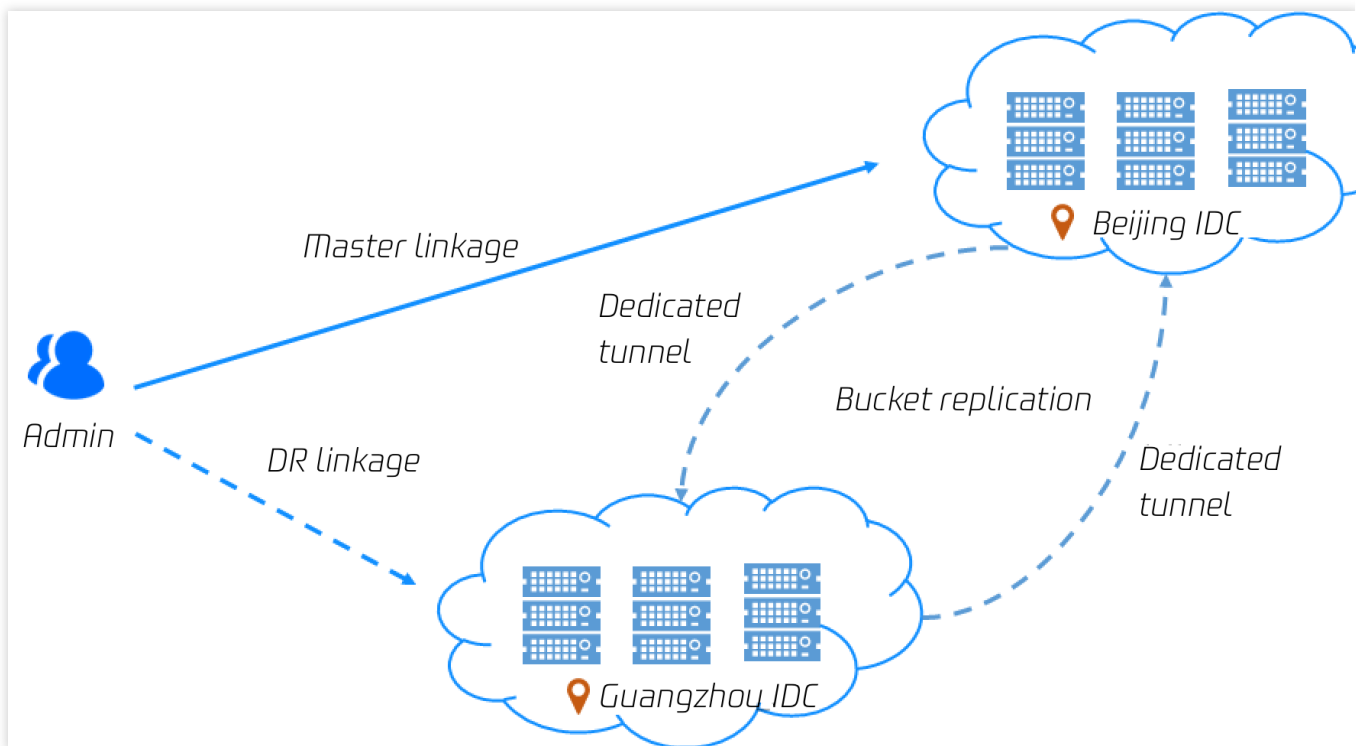
## 3. Disaster recovery

Tencent Cloud COS provides data management capabilities, including data encryption (secures read/write operations of sensitive files), versioning and bucket replication (facilitate remote disaster recovery for data persistence and data recovery for data deleted accidentally or maliciously), and lifecycle (transitions and deletes data for storage cost reduction).

The versioning feature ensures that files will not be overwritten or deleted. After versioning is enabled, if you upload a file whose name already exists, a new version of the file will be generated. If you delete a file, a delete marker will be inserted. You can access the data of any version with a version ID to implement data rollback, freeing yourself from the hassles associated with mis-deleting or overwriting data.

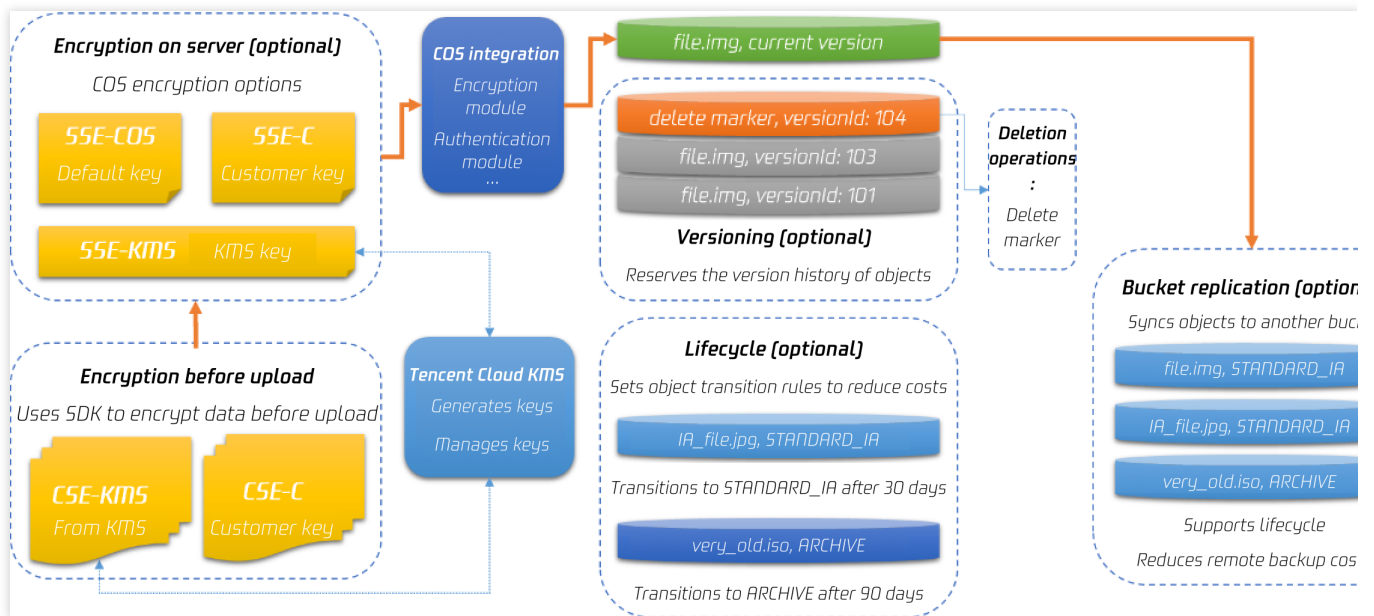


COS bucket replication enables users to copy all incremental files to IDCs in other cities over a dedicated tunnel to implement remote disaster discovery. Data deleted in the master bucket can be restored by batch-replicating data from the backup bucket.

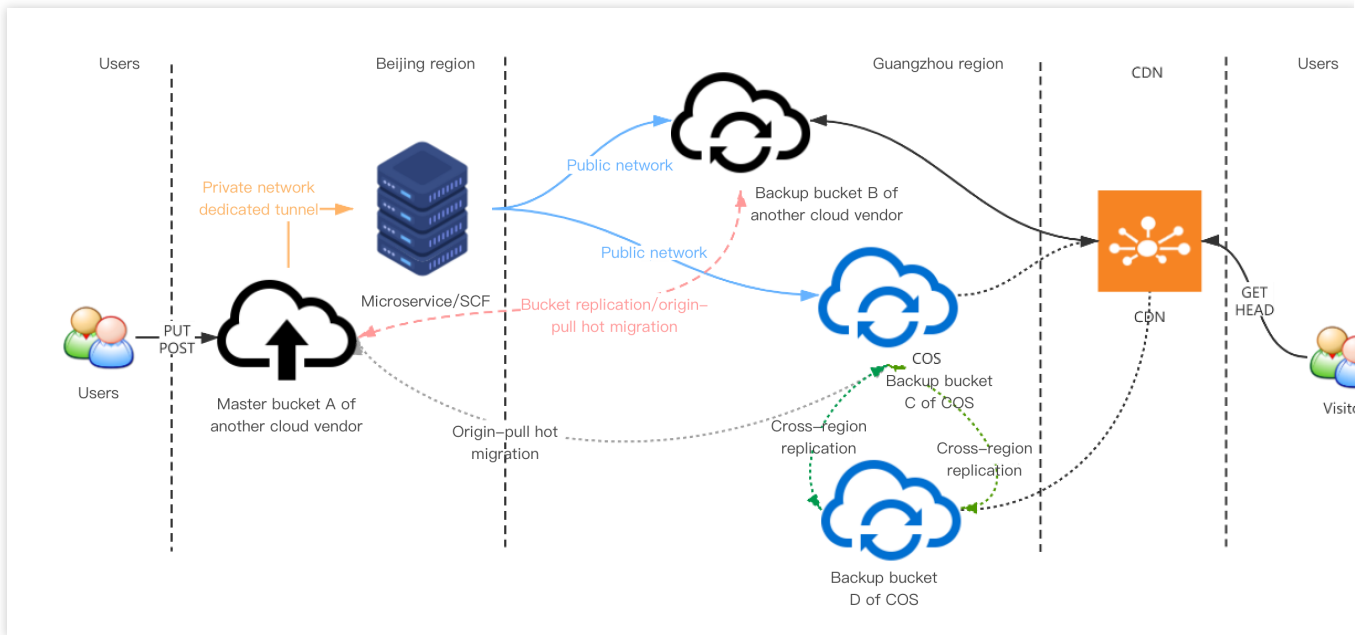




Since versioning and bucket replication might lead to an increase in files, users can leverage the lifecycle feature to transition some backup data to a more affordable storage class (such as STANDARD\_IA and ARCHIVE). Leveraging its data encryption, versioning, bucket replication, and lifecycle features, Tencent Cloud COS offers a complete code backup solution:

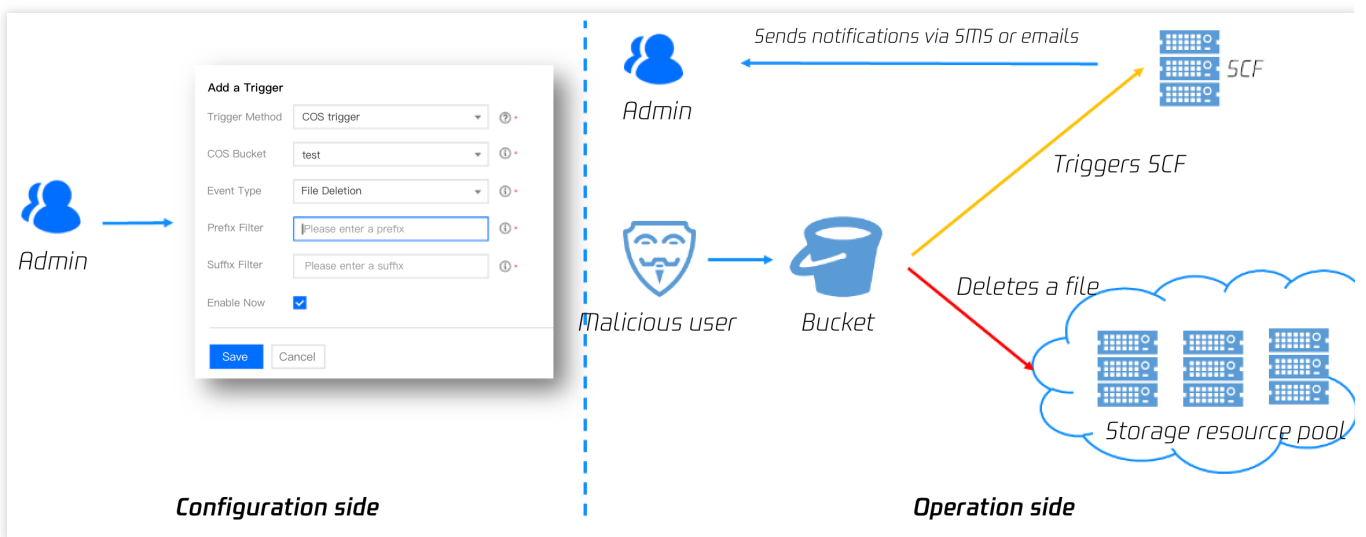


If you are using storage services of other cloud vendors and have strict requirements on data persistence, you can choose the SCF-based COS disaster recovery solution. For example, if your data is stored in other cloud vendors (e.g., AWS or OSS), you can implement remote disaster recovery by using SCF to trigger data sync or by using the bucket replication feature. You can also use SCF to trigger data migration to back up essential data to COS, and use the bucket replication feature for remote disaster recovery. Moreover, Tencent CAM allows you to manage the access permissions of COS data, ensuring that you can restore data from COS even in extreme cases.




## Mid-Event Monitoring

COS supports event notifications by working with SCF. You can use SCF to configure risky operations (such as `DeleteObject`). In this way, notifications will be sent to your email or phone when a risky operation is taking place, allowing you to promptly detect and respond to risky operations.




## Post-Event Tracing

COS provides multiple methods with a low threshold for log monitoring and audit. The logging feature allows users to trace the access logs of buckets. In this way, risky operations such as `DeleteObject`, `PutObjectCopy`, and `PutObjectACL` can be traced. In addition, users can leverage the CloudAudit logs to trace the bucket configurations, such as `DeleteBucket`, `PutBucketACL`, and `PutBucketPolicy`. Moreover, a modification on the permission configuration can also be traced.




**CAM authentication logs**




**Bucket logs**

**Log content**


<i>Resource owner</i>	qcs::cam:uin/39862:uin/39862
<i>Accessed resource</i>	qcs::cos::bucket/object
<i>Access time</i>	06/Feb/2017:12:02:38+0000
<i>Remote IP</i>	120.69.58.5
<i>Requester identity</i>	qcs::cam:uin/39862:uin/26565
<i>Request ID</i>	nTg32GŃjyTLfnDVyMDRLXzUyOT
<i>Operation</i>	REST.PUT.OBJECT
<i>Requested parameter</i>	x-foo=bar
<i>Request response</i>	200
<i>Request error code</i>	noAccess
<i>Bytes sent</i>	265698
<i>Object size (in bytes)</i>	4096
<i>Duration</i>	265
<i>Source</i>	<a href="http://www.qq.com/inde.html">http://www.qq.com/inde.html</a>
<i>User proxy</i>	Chrome/49.50




**App Access logs**



**Log storage**



**Behavior monitoring**



**Alert trigger**

# Anti-Fraud Guide

Last updated : 2024-07-17 16:19:30

## Foreword

In recent years, more and more users have uploaded images, videos, and other resources to Cloud Object Storage (COS) when building websites or image hosting platforms. This not only enhances the access stability but also reduces the storage pressure on servers. However, the accompanying problems such as traffic stealing and image hotlink also cause trouble to many developers. If the storage is accessed maliciously, it may incur high traffic fees and lead to unnecessary disputes. These problems can be prevented actually through various protective measures. This document mainly introduces some common protective measures to help developers properly configure buckets, establish security mechanisms, and reduce the risk of significant financial losses due to such problems.

## Protection Schemes

Hotlink can be prevented in many ways, which are distinguished here by configuration difficulty and threshold. Developers can make a choice according to actual situations and requirements.

### Basic Protection Schemes

#### Modifying Bucket Access Permission

The access permission is one of the most critical and sensitive configurations for buckets. According to incomplete statistics, hotlink occurs mostly because users set the bucket permission to **Public Read**. Since the public read permission allows anonymous access without a signature, it provides opportunities for the dark industry and attackers. Changing the bucket access permission to private read/write can significantly reduce the hotlink risk, because without the key, the signature cannot be computed and the access will be denied.

In case of no special business requirements, it is recommended to configure the **Private Read/Write** permission as much as possible. The COS console currently provides 2 options to set the bucket permission:

Bucket Creation Popup

### Create Bucket ✕

1 Information >
2 Advanced optional configuration >
3 Confirm

Region: China Nanjing

The storage bucket communicates with other Tencent cloud service Intranet in the same region;**The region cannot be modified after creation**, please choose carefully.

Name\* ⓘ The bucket name cannot be r

You can also enter 21 characters,Lowercase letters, digits, and hyphens are supported.**The name cannot be modified after it is created.**

Access Permission:  Private Read/Write  Public Read/Private Write High risk  Public Read/Write High risk

After authentication, the object can be accessed. You can browse through [Setting Access Permissions](#)Authorize users

Endpoint: <Name> yqcloud.com

Request endpoint

Cancel
Next

Bucket Details Page - Permission Management

← Back to Bucket List
Search menu name

Overview

File List

Basic Configurations

Security Management

Permission Management

- Bucket ACL(Access Control List)
- Permission Policy Settings
- Associated CAM Policies

#### Bucket ACL(Access Control List) ⓘ

Public Permission:  Private Read/Write  Public Read/Private Write High risk  Public Read/Write High risk

After authentication, the object can be accessed. You can browse through [Setting Access Permissions](#)Authorize users

User ACL

User Type	Account ID ⓘ	Permission
Root account	<span style="border: 1px solid #ccc; padding: 2px;"> </span>	Full control
<a href="#">Add User</a>		

#### Permission Policy Settings

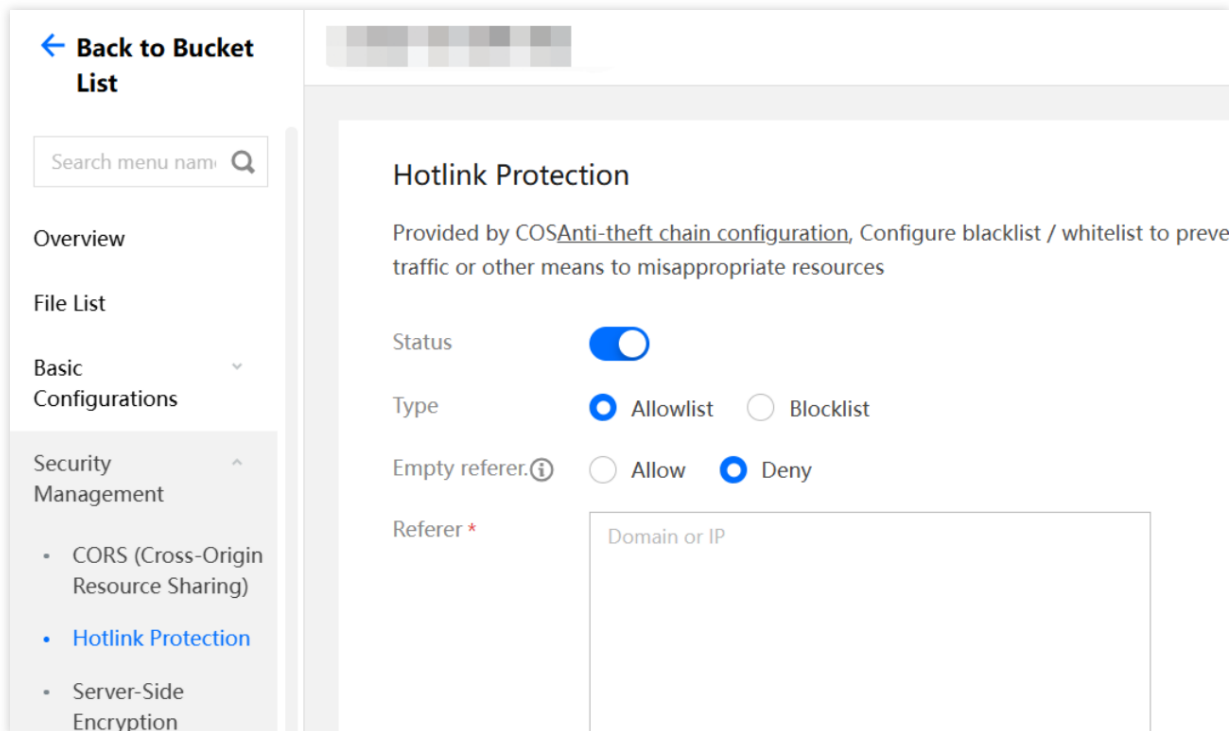
Visual Editor
{} JSON

Strategy	Effect	User Type	Account ID	Resource	Authorized Action	Condition
----------	--------	-----------	------------	----------	-------------------	-----------

## Enabling Bucket Hotlink Protection

Hotlink protection is also one of the most common protection methods. Its principle is to make judgment and verification via the HTTP Referer header. Users can configure an allowlist or a blocklist to permit or prohibit certain access sources.

From **COS Console - Bucket Details Page - Security Management**, you can find the hotlink protection settings:



Here, it is recommended to set the empty referer configuration to **Deny**. The allowlist and blocklist can be selected based on actual business situations. For fixed accesses under certain domain names, you can set an allowlist. If malicious accesses are detected and the accessing domain name or IP address is known, you can manually set a blocklist for blocking. For more hotlink protection tips, refer to [Hotlink Protection Practice](#).

## Configuring Cloud Monitor Alarms

Configuring log management can help locate and analyze the hotlink sources. Abundant fields are provided for query, but the drawback is that developers must actively monitor the logs. If you want to detect hotlink problems more promptly, you can configure Cloud Monitor Alarms. Currently, the COS console supports configuring alarms at the same time when a bucket is created:

Access Permission  Private Read/Write  Public Read/Private Write High risk  Public Read/Write High risk

Data in your bucket can be read without authentication. This configuration is not recommended because of high security risks. Write operations require authentication

⚠ Charging warning: Enabling the public read permission may generate unexpected charges for Internet offline traffic. For details, see [Traffic cost](#)

ⓘ If you must use public read permissions, it is recommended that you take steps to avoid unintended costs, see [Anti-theft brush guide](#)

I have read and agree to the [《COS Charging Description for Object Storage Buckets》](#) , [《Sudden Spike in Requests/Traffic》](#)

Default Alarm

An alarm notification will be issued when the offline traffic within 1 minute is detected to be greater than 5000MB.

If the default alarm policies do not meet your requirements, you can manually create a custom alarm policy in [TCOP](#):

The screenshot shows the 'Create Alarm Policy' interface. The left sidebar is the 'Observability Platform' menu. The main content area has two steps: '1 Configure Alarm Policy' and '2 Configure Alarm Notification'. Under step 1, the 'Basic Info' section includes 'Policy Name' (up to 60 characters) and 'Remarks' (up to 100 characters). The 'Configure Alarm Rule' section includes 'Monitoring Type' (Cloud Product Monitoring, APM, RUM, Cloud Probe Monitor), 'Policy Type' (COS), 'Project' (DEFAULT PROJECT), and 'Tag' (Tag Key, Tag Value). The 'Alarm Object' is set to 'Instance ID'. The 'Trigger Condition' section has 'Apply preset trigger conditions' checked. The 'Metric Alarm' section is partially visible at the bottom.

Go to **Alarm Configuration - Alarm Policy**, click **Create Policy**, find COS under Cloud Product Monitoring, and then find the bucket to be configured in the Instance IDs of the alarm object. You can specify the trigger conditions yourself. All metrics displayed on the data monitoring page of the COS bucket can be configured here.

## Enabling Log Management

In the above description about the hotlink protection blacklist, we mentioned "if malicious domain names or IP addresses are detected". Here we need to enable bucket log management. The access logs will record various fields of each request to help us quickly identify the access source.

From **COS Console - Bucket Details Page - Log Management**, you can find log storage. Once log management is enabled, you can find the access logs under the specified path prefix of the bucket.



**Logging**

You can record the request log related to the bucket operation and save it in the specified bucket in the form of a log manage and use the bucket [Instructions](#) .

Status

Destination Bucket

Path Prefix

Path to save logs: ceshi000-1316781462/cos-access-log/{YYYY}/{MM}/{DD}/{time}\_{random}\_{index}

Service Authorization You've authorized CLS to deliver access logs to your bucket.

For the fields in the log files, you can refer to [Logging Overview](#) documentation. Here are some common analyzable fields:

**userSecretKeyId**: indicates the access key (KeyId) used by the request. If you confirm that the request wasn't initiated by the business itself, the key might have been leaked. You can go to the [Tencent Cloud CAM console](#) to disable the insecure key.

**referer**: indicates the condition used for judgment in hotlink protection settings. If an unfamiliar referer is found, it may be hotlink by other websites. You can configure **Hotlink Protection - Blocklist** to restrict accesses by the referer. For enabling bucket hotlink protection, refer to 1.2.

**remoteIp**: indicates the IP address of the access source. If an untrusted IP address is found, you can go to the **Bucket Details Page - Permission Management - Policy Permission Settings**, click **Add Policy**, and configure the policy to prohibit bucket access by the IP address. For example:

### Add Policy ✕

1 Template > 2 **Configure Policy**

**ⓘ** When dealing with authorizations, it is recommended that you strictly comply to [principles of least privilege](#). You can authorize the user to perform restricted operations (such as only authorize read operations) and access only the resources with specified prefix, to avoid data security risks due to excessive permissions and operations that you don't mean to authorize.

Strategy ID

Effect \*  Allow  Deny

User \*   🗑️  
[Add User](#)

Resource \*  The whole bucket  Specified path

Operation \* [Add Action](#)

Condition ⓘ    ⓘ 🗑️  
[Add Rule](#)

## Advanced Protection Schemes

### Using Custom CDN Acceleration Domain Name

Tencent Cloud CDN also provides many configuration items for protection. They can also effectively prevent hotlink. If the business uses a custom CDN domain name, you can go to the [Tencent Cloud CDN console- Domain Details - Access Control Page](#) for configuration. Some common configurations are as follows:

Hotlink Protection Configuration:

### Hotlink Protection Configuration

Hotlink protection configuration uses http referer to filter the content being requested. [What's hotlink protection](#)

On/Off

Hotlink protection rules not set

### CDN Authentication Configuration:

#### Authentication Configuration

Custom token authentication allows you to authenticate access based on the specified file extension. Access paths containing Chinese characters are not supported currently. [What's authentication configuration](#)

The authentication parameters are ignored in node caching, which does not affect the cache hit rate of the domain name.

[Authentication Calculator](#)

On/Off

### IP Blocklist/Allowlist Configuration:

#### IP Blocklist/Allowlist Configuration

IP blocklist/allowlist filters requests by request IPs. [What's IP blocklist and allowlist](#)

Rule priority: The priority of the rules below the list is higher than that of the rules above the list.

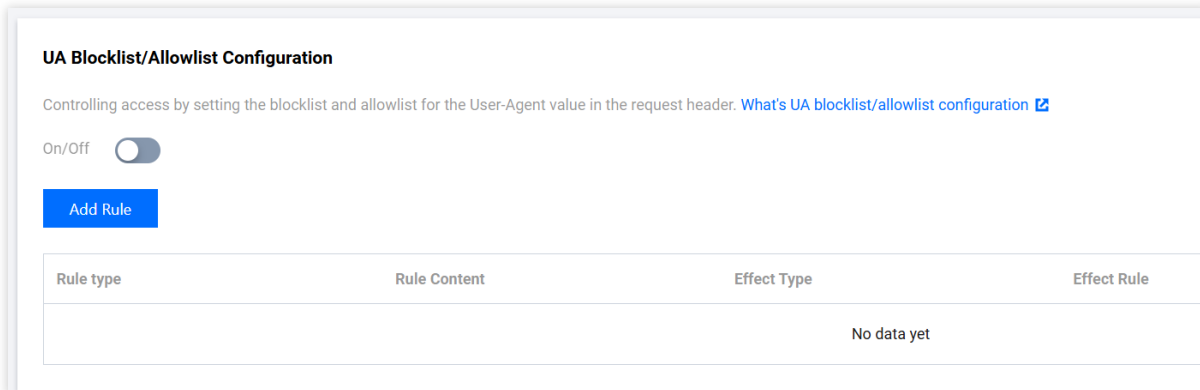
On/Off

Create Rule

Adjust priority

Rule type	Rule Content

### UA Blocklist/Allowlist Configuration:



There are also more complex configuration items, such as IP access frequency control configuration and downlink rate limit configuration. You can refer to [CDN Console Documentation](#) for usage.

## Implementing Auto-Blocking with SCF, Cloud Monitor, and COS APIs

In basic protection, we mentioned that Cloud Monitor alarms can be configured to detect abnormal traffic in a timely manner. However, sometimes information may be missed or it is inconvenient to handle problems immediately when you're out. In such cases, a simple automation logic can be implemented by calling APIs with automated script code. Get abnormal public network downlink traffic metrics (InternetTraffic) from Cloud Monitor -> Automatically call COS PutBucketAcl to change the bucket permission to private read/write. The following 2 APIs are mainly involved, and we provide online call samples for debugging reference:

Cloud Monitor's GetMonitorData - [Sample Call](#)

COS's PutBucketAcl - [Sample Call](#)

## Related Suggestions

The above are some common protective measures for hotlink provided in this document. In addition, we must pay attention to some small details in daily use of the COS. Here are some additional usage recommendations that can help reduce the hotlink risk to some extent:

Avoid using plaintext API access keys in the open-sourced code, to prevent key leakage from causing security risks. It is recommended to follow the **Principle of Least Privilege**.

When configuring cross-domain rules, avoid allowing all sources (i.e., Origin: \*) for access. Try to set specific sources instead.

When uploading a large number of files, avoid using too simple sequential prefixes (such as sequence numbers and timestamps), for this could cause attackers to traverse files in the bucket more easily.

# Hotlink Protection Practice

Last updated : 2024-03-25 15:16:26

## Overview

COS allows you to configure hotlink protection for your bucket. You can set a blocklist and allowlist for access sources to prevent resource hotlinking. This document describes how to configure hotlink protection for a bucket.

## How Does Hotlink Protection Work

Hotlink protection works by checking the Referer address in the request header:

Referer is a part of the header. When a browser sends a request to a web server, it usually carries a Referer to tell the server which page the request comes from, so that the server can decide to deny or allow the access to resources.

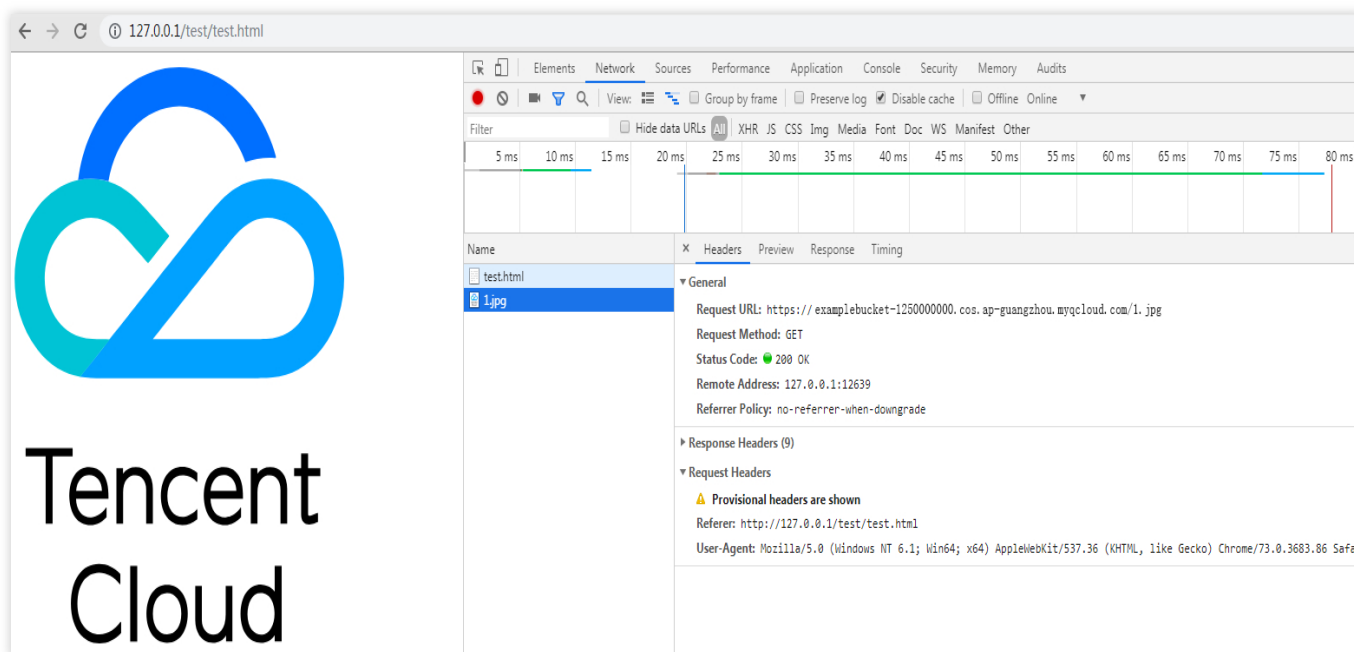
If you open the file link `https://examplebucket-1250000000.cos.ap-`

`guangzhou.myqcloud.com/1.jpg` directly in a browser, the request header will not have a Referer.

For example, in the figure below, the image `1.jpg` is embedded in `https://127.0.0.1/test/test.html`,

and a Referer pointing to the access origin will be carried when you access

`https://127.0.0.1/test/test.html` :



# Hotlink Protection Case Study

User A uploaded the image resource `1.jpg` to COS, and the accessible link to the image is

```
https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg
```

User A embedded the image in their webpage `https://example.com/index.html` and the image is accessible.

User B saw the image on user A's webpage and decided to embed it in their own webpage

```
https://b.com/test/test.html
```

, and user B's webpage can also display the image properly.

In the above case, user A's image resource `1.jpg` was hotlinked by user B. User A doesn't know that their resource in COS is being used by user B's webpage and suffers from losses caused by extra traffic fees.

## Solution

In the above [Hotlink Protection Case Study](#), user A can prevent user B from hotlinking their image by setting hotlink protection in the following way:

1. Set a hotlink protection rule for the bucket "examplebucket-1250000000". There are two options for preventing user B from hotlinking:

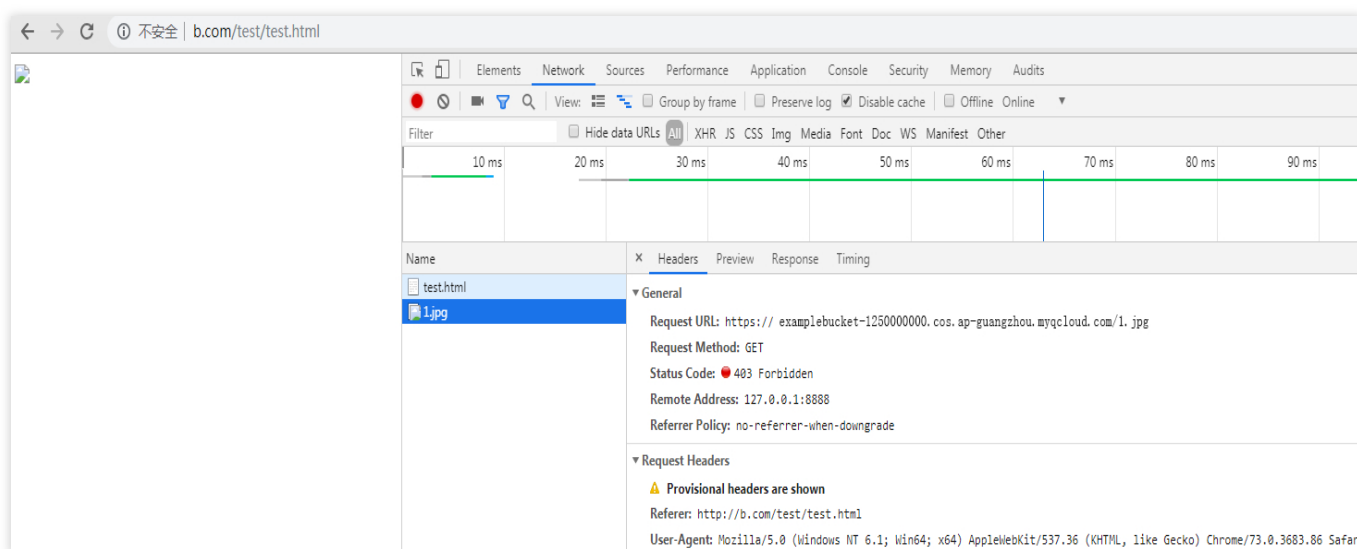
Option 1: configure a blacklist by entering the domain name `*.b.com`, and save it.

Option 2: configure a whitelist, enter `*.example.com` for the domain name, and save.

2. After hotlink protection is enabled:

The image can be displayed properly when `https://example.com/index.html` is accessed.

The image cannot be displayed when `https://b.com/test/test.html` is accessed, as shown below:

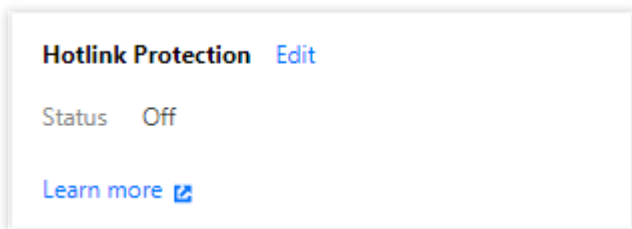


## Directions

1. Log in to the [COS Console](#) and click **Bucket List** on the left sidebar to enter the bucket list page.
2. Select the bucket for which to configure hotlink protection and enter it.

Bucket Name ↕	Access ▼	Region ▼	Creation Time ↕	Operation
examplebucket-1250000000	Specified user	Chengdu (China) (ap-chengdu)	2019-03-20 15:29:59	<a href="#">Monitor</a> <a href="#">Configure</a> <a href="#">M...</a>

3. Click **Security Management > Hotlink Protection** on the left.
4. In the **Hotlink Protection** area, click **Edit**.



5. Enable hotlink protection and configure the list type and domain name. Here, select Option 2 as detailed below:

**Type:** blocklist or allowlist

**Blocklist:** It prohibits domain names in the list to access the default access address of the bucket. If a domain name **in the list** accesses the default access address of the bucket, a 403 error will be returned.

**Allowlist:** It prohibits domain names not in the list to access the default access address of the bucket. If a domain name **not in the list** accesses the default access address of the bucket, a 403 error will be returned.

**Referer :** Up to 10 domain names can be set and they will be matched by a prefix. Domain names, IPs, and asterisk \* are supported formats (one address per line). Below are configuration rule description and examples:

Domain names and IPs with a specific port are supported, such as `example.com:8080` and `10.10.10.10:8080`.

If `example.com` is configured, addresses prefixed with `example.com` can be hit, such as `example.com/123`.

If `example.com` is configured, addresses prefixed with `https://example.com` and `http://example.com` can be hit.

If `example.com` is configured, the domain name with a specific port can also be hit, such as `example.com:8080`.

If `example.com:8080` is configured, the domain name `example.com` cannot be hit.

If `*.example.com` is configured, its second-level and third-level domain names can be restricted, such as `example.com`, `b.example.com`, and `a.b.example.com`.

**Note:**

After hotlink protection is **enabled**, the corresponding domain names must be entered.

6. After completing the configuration, click **Save**.

### Hotlink Protection

Status

Type  Whitelist  Blacklist

Allow empty referer ⓘ  Allow  Deny

Referer  ✓

Please enter domain name or IP address, support multi-line, up to 10 lines, support wildcard \*, such as: \*.test.c

[Learn more](#)

## FAQs

For questions about hotlink protection, see the [Data Security](#) section in COS FAQs.



# Data Verification

## MD5 Verification

Last updated : 2024-03-25 15:16:26

### Overview

Errors may occur when data is being transmitted between the client and the server. COS can guarantee the integrity of the uploaded data through MD5 verification. Only when the MD5 checksum received by the COS server is the same as that you set can the data be successfully uploaded.

Each object in COS has a corresponding ETag, which is the information identifier of the object content when the object is created. However, the ETag is not necessarily equivalent to the MD5 checksum of the object content.

Therefore, the ETag cannot be used to verify whether the downloaded object is the same as the original object. In this case, you can use custom object metadata (x-cos-meta-\*) to verify the object consistency.

### Data Verification Methods

#### Verify an uploaded object

If you need to verify whether the object uploaded to COS is the same as the local object, you can set the [Content-MD5](#) field in the HTTP upload request to the Base64-encoded MD5 checksum of the object content. After that, the COS server will verify the uploaded object. Only when the MD5 checksum received by the COS server is the same as the Content-MD5 value you set can the object be successfully uploaded.

#### Verify a downloaded object

If you need to verify whether the downloaded object is the same as the original object, you can use a verification algorithm to calculate the checksum of the object when it is uploaded, set the checksum of the object through custom metadata, recalculate the checksum of the object after downloading the object, and then verify it against the custom metadata. In this mode, you can choose the verification algorithm as you wish, but for the same object, the algorithm used during upload should be the same as that used during download.

### API Samples

#### Simple Upload Request

Below is a sample request for object upload. When uploading the object, set the Content-MD5 to the Base64-encoded MD5 checksum of the object content and set the custom metadata "x-cos-meta-md5" to the checksum of the object.

Only when the MD5 checksum received by the COS server is the same as the Content-MD5 value you set can the object be successfully uploaded.

**Note:**

In the sample, the checksum of the object is obtained through the MD5 checksum algorithm, and you can choose other algorithms as you wish.

```
PUT /exampleobject HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 21 Jun 2019 09:24:28 GMT
Content-Type: image/jpeg
Content-Length: 13
Content-MD5: ti4QvKtVqIJA VzxD bP/c+Q==
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO***&q-si
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
Connection: close

[Object Content]
```

**Multipart Upload Request**

Below is a sample request to initialize a multipart upload. When uploading object parts, you can set the custom metadata of the object by initializing the multipart upload. Here, set the custom metadata "x-cos-meta-md5" as the checksum of the object.

```
POST /exampleobject?uploads HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 21 Jun 2019 09:45:12 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO***&q-si
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
```

**Note:**

For files uploaded using multipart upload, COS will verify the MD5 checksum of each part instead of the MD5 checksum of the merged file.

**Object download response**

Below is a sample response obtained after you send an object download request. You can get the custom metadata "x-cos-meta-md5" of the object from the response and then check it against the recalculated checksum of the object to verify whether the downloaded object is the same as the original object.

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 13
Connection: close
Accept-Ranges: bytes
```

```
Cache-Control: max-age=86400
Content-Disposition: attachment; filename=example.jpg
Date: Thu, 04 Jul 2019 11:33:00 GMT
ETag: "b62e10bcab55a88240bd9c436cffdcf9"
Last-Modified: Thu, 04 Jul 2019 11:32:55 GMT
Server: tencent-cos
x-cos-request-id: NWQxZGUzZWVfNjI4NWQ2NF9lMWYyXzk1NjFj****
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
```

[Object Content]

## SDK Samples

The following example uses the Python SDK to verify object integrity. The complete sample code is as follows.

### Note:

The code is based on Python 2.7. For more information on how to use the Python SDK, see [Object Operations](#).

### 1. Initialization configuration

Configure user attributes, including SecretId, SecretKey, and region, and create a client object.

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import os
import logging
import hashlib

logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# Configure user attributes, including SecretId, SecretKey, and region
# APPID has been removed from the configuration. Please specify it using the `Bucket
secret_id = os.environ['COS_SECRET_ID']      # User `SecretId`. We recommend you us
secret_key = os.environ['COS_SECRET_KEY']    # User `SecretKey`. We recommend you us
region = 'ap-beijing'                       # Replace with your own region (which is Beijing in this
token = None                                 # Temporary key token. For more information on how to ge
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

### 2. Verify an object uploaded using simple upload

### (1) Calculate the checksum of the object

Get the checksum of the object through the MD5 checksum algorithm (you can choose other algorithms as you wish).

```
object_body = 'hello cos'
# Get the MD5 checksum of the object
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

### (2) upload objects using simple upload

EnableMD5=True in the code indicates the enablement of MD5 verification for object upload. The SDK for Python will calculate the Content-MD5. Enabling this will increase the time it takes to upload the object. Only when the MD5 checksum of the object received by the COS server is the same as the Content-MD5 can the object be successfully uploaded.

x-cos-meta-md5 is a custom parameter (in the name format of x-cos-meta-\*), which represents the MD5 checksum of the object.

```
# Upload the object using simple upload and enable MD5 verification
response = client.put_object(
    Bucket='examplebucket-1250000000',      # Replace with your own bucket name. He
    Body='hello cos',                      # Content of the uploaded object
    Key='example-object-1',                 # Replace with the key value of your uploaded
    EnableMD5=True,                        # Enable MD5 verification for upload
    Metadata={                              # Set the custom parameter and save the MD5 c
        'x-cos-meta-md5' : md5_str
    }
)
print 'ETag: ' + response['ETag']         # Etag value of the object
```

### (3) Download the object

Download the object and get the custom parameter.

```
# Download the object
response = client.get_object(
    Bucket='examplebucket-1250000000',    # Replace with your own bucket name. He
    Key='example-object-1'                # Key value of the download object
)
fp = response['Body'].get_raw_stream()
download_object = fp.read()              # Get the object content
print "get object body: " + download_object
print 'ETag: ' + response['ETag']
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5'] # Get the custom parameter
```

#### (4) Verify the object

After successfully downloading the object, you can recalculate the checksum of the object (the verification algorithm should be the same as that used for upload) and check it against the custom parameter "x-cos-meta-md5" to verify whether the downloaded object is the same as the uploaded object.

```
# Calculate the MD5 checksum of the downloaded object
md5 = hashlib.md5()
md5.update(download_object)
md5_str = md5.hexdigest()
print 'download object md5: ' + md5_str

# Verify object consistency by checking the MD5 checksum of the downloaded object a
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```

### 3. Verify an object uploaded in parts

#### (1) Calculate the checksum of the object

Simulate object parts and calculate the checksum of the entire object. The MD5 checksum algorithm is used to obtain the checksum of the object in the sample below, and you can choose other algorithms as you wish.

```
OBJECT_PART_SIZE = 1024 * 1024      # Size of each simulated part
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123      # Total size of the object
object_body = '1' * OBJECT_TOTAL_SIZE      # Object content

# Calculate the MD5 checksum of the entire object content
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

#### (2) Initialize the multipart upload

When initializing the multipart upload, set the custom parameter "x-cos-meta-md5" and use the MD5 checksum of the entire object as the parameter value.

```
# Initialize the multipart upload
response = client.create_multipart_upload(
    Bucket='examplebucket-1250000000', #Replace with your own bucket name and APPI
    Key='exampleobject-2',           # Replace with the key value of the uploade
    StorageClass='STANDARD',        # Storage class of the object
    Metadata={
        'x-cos-meta-md5' : md5_str      # Set the custom parameter to the MD5 check
```

```
    }  
)  
#Get the UploadId of the multipart upload  
upload_id = response['UploadId']
```

### (3) Upload the object in parts

During a multipart upload, an object is divided into multiple (up to 10,000) parts for the upload. The size of each part can range from 1 MB to 5 GB, and the last part can be less than 1 MB. When uploading the parts, you need to set the PartNumber of each part. EnableMD5=True indicates enabling the part check, which increases the time it takes to upload the object. The Python SDK will calculate the Content-MD5 of each part. Only when the MD5 checksum of the object received by the COS server is the same as the Content-MD5 can the parts be successfully uploaded. After the upload succeeds, the ETag of each part will be returned.

```
#Upload an object in parts where the size of each part is OBJECT_PART_SIZE except t  
part_list = list()  
position = 0  
left_size = OBJECT_TOTAL_SIZE  
part_number = 0  
while left_size > 0:  
    part_number += 1  
    if left_size >= OBJECT_PART_SIZE:  
        body = object_body[position:position+OBJECT_PART_SIZE]  
    else:  
        body = object_body[position:]  
    position += OBJECT_PART_SIZE  
    left_size -= OBJECT_PART_SIZE  
  
# Upload parts  
response = client.upload_part(  
    Bucket='examplebucket-1250000000', #Replace with your own bucket name and  
    Key='exampleobject-2', # Key value of the object  
    Body=body,  
    PartNumber=part_number,  
    UploadId=upload_id,  
    EnableMD5=True # Enable part verification and the COS server will per  
)  
etag = response['ETag'] # ETag represents the MD5 checksum of each part  
part_list.append({'ETag' : etag, 'PartNumber' : part_number})  
print etag + ', ' + str(part_number)
```

### (4) Complete the multipart upload

After all parts are uploaded, you need to complete the multipart upload operation. The ETag and PartNumber of each part should be in one-to-one correspondence which will be used by the COS server to verify the part accuracy. After

the multipart upload completes, the returned ETag represents the unique tag value of the merged object but not the MD5 checksum of the entire object content. As a result, you can use the custom parameter to verify the object when downloading it.

```
#Complete the multipart upload
response = client.complete_multipart_upload(
    Bucket='examplebucket-1250000000',      # Replace with your own bucket name. He
    Key='exampleobject-2',                  # Key value of the object
    UploadId=upload_id,
    MultipartUpload={                       # Requires one-to-one correspondence between ETag and P
        'Part' : part_list
    },
)

# ETag represents the unique tag value of the merged object, which is not the MD5 c
print "ETag: " + response['ETag']
print "Location: " + response['Location']    #URL
print "Key: " + response['Key']
```

## (5) Download the object

Download the object and get the custom parameter.

```
# Download the object
response = client.get_object(
    Bucket='examplebucket-1250000000',    #Replace with your own bucket name and APPI
    Key='exampleobject-2'                 # Key value of the object
)
print 'ETag: ' + response['ETag']          # The ETag of the object i
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5'] # Get the custom parameter
```

## (6) Verify the object

After successfully downloading the object, you can recalculate the MD5 checksum of the object and check it against the custom parameter "x-cos-meta-md5" to verify whether the downloaded object is the same as the uploaded object.

```
# Calculate the MD5 checksum of the downloaded object
fp = response['Body'].get_raw_stream()
DEFAULT_CHUNK_SIZE = 1024*1024
md5 = hashlib.md5()
chunk = fp.read(DEFAULT_CHUNK_SIZE)
while chunk:
    md5.update(chunk)
    chunk = fp.read(DEFAULT_CHUNK_SIZE)
md5_str = md5.hexdigest()
print 'download object md5: ' + md5_str
```

```
# Verify object consistency by checking the MD5 checksum of the downloaded object a
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```



# CRC64 Check

Last updated : 2024-03-25 15:16:26

## Overview

Errors may occur when data is transferred between the client and the server. COS can not only verify data integrity through [MD5 and custom attributes](#), but also the CRC64 check code.

COS will calculate the CRC64 of the newly uploaded object and store the result as object attributes. It will carry `x-cos-hash-crc64ecma` in the returned response header, which indicates the CRC64 value of the uploaded object calculated according to ECMA-182 standard. If an object already has a CRC64 value stored before this feature is activated, COS will not calculate its CRC64 value, nor will it be returned when the object is obtained.

## Description

APIs that currently support CRC64 include:

APIs for simple upload

[PUT Object](#) and [POST Object](#): you can get the CRC64 check value for your file from the response headers.

Multipart upload APIs

[Upload Part](#): you can compare and verify the CRC64 value returned by COS against the value calculated locally.

[Complete Multipart Upload](#): returns a CRC64 value for the entire object only if each part has a CRC64 attribute.

Otherwise, no value is returned.

The [Upload Part - Copy](#) operation returns a corresponding CRC64 value.

When you call the [PUT Object - Copy](#), the CRC64 value is returned only if the source object has one.

The [HEAD Object](#) and [GET Object](#) operations return the CRC64 value provided the object has one. You can compare and verify the CRC64 value returned by COS against that calculated locally.

## API Samples

### Upload Part response

The following example shows the response to an Upload Part request. The `x-cos-hash-crc64ecma` header represents the CRC64 value of a part, which you can compare against the locally calculated CRC64 value to verify the part integrity.

```
HTTP/1.1 200 OK
content-length: 0
```

```
connection: close
date: Thu, 05 Dec 2019 01:58:03 GMT
etag: "358e8c8b1bfa35ee3bd44cb3d2cc416b"
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0MmJfMjBiNDU4NjRfNjkyZl80ZjZi****
```

## Complete Multipart Upload response

The following example shows the response to a Complete Multipart Upload request. The `x-cos-hash-crc64ecma` header represents the CRC64 value of an entire object, which you can compare against the locally calculated CRC64 value to verify the object integrity.

```
HTTP/1.1 200 OK
content-type: application/xml
transfer-encoding: chunked
connection: close
date: Thu, 05 Dec 2019 02:01:17 GMT
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0ZWRFMjNiMjU4NjRfOGQ4Ml81MDEw****

[Object Content]
```

# SDK Samples

## Python SDK

The following example uses the Python SDK to verify object integrity. The complete sample code is as follows.

### Note:

The code is based on Python 2.7. For more information on how to use the Python SDK, see [Object Operations](#).

### 1. Initialization configuration

Configure user attributes, including SecretId, SecretKey, and region, and create a client object.

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import logging
import hashlib
```

```
import crcmod

logging.basicConfig(level=logging.INFO, stream=sys.stdout)

# Configure user attributes, including SecretId, SecretKey, and region
# APPID has been removed from the configuration. Please specify it using the `Bucket
secret_id = COS_SECRETID          # Replace with your own SecretId
secret_key = COS_SECRETKEY        # Replace with your own SecretKey
region = 'ap-beijing'            # Replace with your own region (which is Beijing in this
token = None                      # If a temporary key is used, the token needs to be spec
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token)
client = CosS3Client(config)
```

## 2. Calculate the object checksum

Simulate object parts and calculate the checksum of the entire object.

```
OBJECT_PART_SIZE = 1024 * 1024      # Size of each simulated part
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123      # Total size of the object
object_body = '1' * OBJECT_TOTAL_SIZE      # Object content

#Calculate the checksum of the entire object.
c64 = crcmod.mkCrcFun(0x142F0E1EBA9EA3693, initCrc=0, xorOut=0xffffffffffffffff, re
local_crc64 =str(c64(object_body))
```

## 3. Initialize the multipart upload

```
# Initialize the multipart upload
response = client.create_multipart_upload(
    Bucket='examplebucket-1250000000', #Replace with your own bucket name and APPI
    Key='exampleobject',              # Replace with the key value of your uploaded
    StorageClass='STANDARD',         # Storage class of the object
)
#Get the UploadId of the multipart upload
upload_id = response['UploadId']
```

## 4. Upload the object using multipart upload

During a multipart upload, an object is divided into multiple (up to 10,000) parts for upload. The size of each part ranges from 1 MB to 5 GB, except the last part can be less than 1 MB. When uploading parts, configure the PartNumber of each part and calculate its corresponding CRC64 value. After the parts are successfully uploaded, check the returned CRC64 value with the locally calculated value to verify the object integrity.

```
#Upload an object in parts where the size of each part is OBJECT_PART_SIZE except t
part_list = list()
```

```
position = 0
left_size = OBJECT_TOTAL_SIZE
part_number = 0
while left_size > 0:
    part_number += 1
    if left_size >= OBJECT_PART_SIZE:
        body = object_body[position:position+OBJECT_PART_SIZE]
    else:
        body = object_body[position:]
    position += OBJECT_PART_SIZE
    left_size -= OBJECT_PART_SIZE
    local_part_crc_64 = c64(body)#Calculate CRC64 locally

    response = client.upload_part(
        Bucket='examplebucket-1250000000',
        Key='exampleobject',
        Body=body,
        PartNumber=part_number,
        UploadId=upload_id,
    )
    part_crc_64 = response['x-cos-hash-crc64ecma']# CRC64 returned by the server
    if local_part_crc64 != part_crc_64:# Data Check
        print 'crc64 check FAIL'
        exit(-1)
    etag = response['ETag']
    part_list.append({'ETag' : etag, 'PartNumber' : part_number})
```

## 5. Complete the multipart upload

After all parts are successfully uploaded, you need to complete the multipart upload. Then, you can verify the CRC64 value returned by COS against that of the local object.

```
#Complete the multipart upload
response = client.complete_multipart_upload(
    Bucket='examplebucket-1250000000', #Replace with your own bucket name and APPI
    Key='exampleobject', #Key value of the object
    UploadId=upload_id,
    MultipartUpload={ #Require one-to-one correspondence between ETag and Par
        'Part' : part_list
    },
)
crc64ecma = response['x-cos-hash-crc64ecma']
if crc64ecma != local_crc64:# Data Check
    print 'check crc64 Failed'
    exit(-1)
```

## Java SDK

You are advised to use advanced APIs of the Java SDK to upload objects. For more information, please see [Object Operations](#).

### Calculating CRC64 locally

```
String calculateCrc64(File localFile) throws IOException {
    CRC64 crc64 = new CRC64();

    try (FileInputStream stream = new FileInputStream(localFile)) {
        byte[] b = new byte[1024 * 1024];
        while (true) {
            final int read = stream.read(b);
            if (read <= 0) {
                break;
            }
            crc64.update(b, read);
        }
    }

    return Long.toUnsignedString(crc64.getValue());
}
```

### Getting CRC64 of a COS file and verifying it with the local one

```
// For more information about how to create COSClient, see [Getting Started](https://
ObjectMetadata cosMeta = COSClient().getObjectMetadata(bucketName, cosFilePath);
String cosCrc64 = cosMeta.getCrc64Ecma();
String localCrc64 = calculateCrc64(localFile);

if (cosCrc64.equals(localCrc64)) {
    System.out.println("ok");
} else {
    System.out.println("fail");
}
```

# Big Data Practice

## Using COS as Deep Storage of Druid

Last updated : 2024-03-25 15:16:26

### Environment Dependencies

[HADOOP-COS](#) and Hadoop-COS-Java-SDK (included in the `dep` directory of HADOOP-COS)

Druid version: Druid-0.12.1

### Download and Installation

#### Downloading HADOOP-COS

Download [HADOOP-COS](#) on Github.

#### Installing HADOOP-COS

Druid-hdfs-extension is required if Druid uses COS for Deep Storage.

After downloading HADOOP-COS, copy the version you want displayed as `hadoop-cos-2.x.x.jar` under the `dep` directory to the Druid installation path `extensions/druid-hdfs-storage` and the `hadoop-dependencies/hadoop-client/2.x.x`. Since Druid accesses COS using HDFS plugin, the version you selected needs to be the same as that of the HDFS plugin.

### Directions

#### Modifying configuration

1. Modify the file `conf/druid/_common/common.runtime.properties` under Druid installation path, add the extension of hdfs to `druid.extensions.loadList`, specify hdfs as Druid's deep storage, and enter the path of cosn:

```
properties
druid.extensions.loadList=["druid-hdfs-storage"]
druid.storage.type=hdfs
druid.storage.storageDirectory=cosn://bucket-appid/<druid-path>
```

2. Create a hdfs configuration file `hdfs-site.xml` under the directory `conf/druid/_common/`, and enter your COS keys and other information:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.cosn.userinfo.secretId</name>
    <value>xxxxxxxxxxxxxxxxxxxxxxxx</value>
  </property>
  <property>
    <name>fs.cosn.userinfo.secretKey</name>
    <value>xxxxxxxxxxxxxxxx</value>
  </property>
  <property>
    <name>fs.cosn.impl</name>
    <value>org.apache.hadoop.fs.CosFileSystem</value>
  </property>
  <property>
    <name>fs.cosn.userinfo.region</name>
    <value>ap-xxxx</value>
  </property>
  <property>
    <name>fs.cosn.tmp.dir</name>
    <value>/tmp/hadoop_cos</value>
  </property>
</configuration>
```

The supported items for the above configuration are exactly the same as those described in the HADOOP-COS official documentation. For more information, see [HADOOP Tool](#).

## Getting started

After the Druid processes are started in turn, the Druid data can be loaded into the COS.

# Importing/Exporting COS Using DataX

Last updated : 2024-03-25 15:16:26

## Environmental Dependencies

[HADOOP-COS](#) and the corresponding [cos\\_api-bundle](#).

DataX version: DataX 3.0

## Download and Installation

### Downloading HADOOP-COS

Download [HADOOP-COS](#) and the corresponding [cos\\_api-bundle](#) on Github.

### Downloading DataX package

Download [DataX](#) on Github.

### Installing HADOOP-COS

After HADOOP-COS is downloaded, copy `hadoop-cos-2.x.x-${version}.jar` and `cos_api-bundle-${version}.jar` to the DataX decompression paths `plugin/reader/hdfsreader/libs/` and `plugin/writer/hdfswriter/libs/`.

## How to Use

### DataX configuration

#### Modifying datax.py script

Open the `bin/datax.py` script in the DataX decompression directory, and modify the CLASS\_PATH variable in the script as follows:

```
CLASS_PATH = ("%s/lib/*:%s/plugin/reader/hdfsreader/libs/*:%s/plugin/writer/hdfswri
```

#### Configuring `hdfsreader` and `hdfswriter` in JSON configuration file

A sample JSON file is as shown below:

```
{  
  "job": {
```



```
"setting": {
  "speed": {
    "byte": 10485760
  },
  "errorLimit": {
    "record": 0,
    "percentage": 0.02
  }
},
"content": [{
  "reader": {
    "name": "hdfsreader",
    "parameter": {
      "path": "testfile",
      "defaultFS": "cosn://examplebucket-1250000000/",
      "column": ["*"],
      "fileType": "text",
      "encoding": "UTF-8",
      "hadoopConfig": {
        "fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
        "fs.cosn.userinfo.region": "ap-beijing",
        "fs.cosn.tmp.dir": "/tmp/hadoop_cos",
        "fs.cosn.userinfo.secretId": "COS_SECRETID",
        "fs.cosn.userinfo.secretKey": "COS_SECRETKEY"
      }
    },
    "fieldDelimiter": ","
  }
},
  "writer": {
    "name": "hdfswriter",
    "parameter": {
      "path": "/user/hadoop/",
      "fileName": "testfile1",
      "defaultFS": "cosn://examplebucket-1250000000/",
      "column": [{
        "name": "col",
        "type": "string"
      },
      {
        "name": "col1",
        "type": "string"
      },
      {
        "name": "col2",
        "type": "string"
      }
    ]
  }
},
],
```

```
        "fileType": "text",
        "encoding": "UTF-8",
        "hadoopConfig": {
            "fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
            "fs.cosn.userinfo.region": "ap-beijing",
            "fs.cosn.tmp.dir": "/tmp/hadoop_cos",
            "fs.cosn.userinfo.secretId": "COS_SECRETID",
            "fs.cosn.userinfo.secretKey": "COS_SECRETKEY"
        },
        "fieldDelimiter": ":",
        "writeMode": "append"
    }
}
}
}
}
```

#### Notes:

Configure `hadoopConfig` as required for cosn.

Use `defaultFS` to specify the cosn path, e.g. `cosn://examplebucket-1250000000/`.

In `fs.cosn.userinfo.region`, enter the region where your bucket resides, such as `ap-beijing`. For more information, see [Regions and Access Endpoints](#).

For `COS_SECRETID` and `COS_SECRETKEY`, use your own COS key information.

The other fields can be the same as those for hdfs.

## Migrating data

Save the configuration file as `hdfs_job.json` in the `job` directory by running

```
bin/datax.py job/hdfs_job.json
```

The resulting output is as shown below:

```
2020-03-09 16:49:59.543 [job-0] INFO JobContainer -
  [total cpu info] =>
    averageCpu          | maxDeltaCpu          | m
    -1.00%              | -1.00%               | -

  [total gc info] =>
    NAME                 | totalGCCCount         | maxDeltaGCCCount      | m
    PS MarkSweep         | 1                     | 1                     | 1
    PS Scavenge          | 1                     | 1                     | 1

2020-03-09 16:49:59.543 [job-0] INFO JobContainer - PerfTrace not enable!
2020-03-09 16:49:59.543 [job-0] INFO StandAloneJobContainerCommunicator - Total 2
```

```
2020-03-09 16:49:59.544 [job-0] INFO JobContainer -
Job start time           : 2020-03-09 16:49:48
Job end time             : 2020-03-09 16:49:48
Job duration              :                11s
Average job traffic      :                3B/s
Recorded write speed     :                0rec/s
Recorded read count      :                2
Read/Write failure count :                0
```

# Configuring COSN for CDH

Last updated : 2024-03-25 15:16:26

## Overview

CDH (Cloudera's distribution, including Apache Hadoop) is one of the most popular Hadoop distributions in the industry. This document describes how to use the COSN storage service in a CDH environment to separate big data computing from storage.

### Note:

COSN refers to the Hadoop-COS file system.

Currently, the support for big data modules by COSN is as follows:

Module	Supported	Service Module to Restart
YARN	Yes	NodeManager
Hive	Yes	HiveServer and HiveMetastore
Spark	Yes	NodeManager
Sqoop	Yes	NodeManager
Presto	Yes	HiveServer, HiveMetastore, and Presto
Flink	Yes	None
Impala	Yes	None
EMR	Yes	None
Self-built component	To be supported in the future	No
HBase	Not recommended	None

## Versions

This example uses software versions as follows:

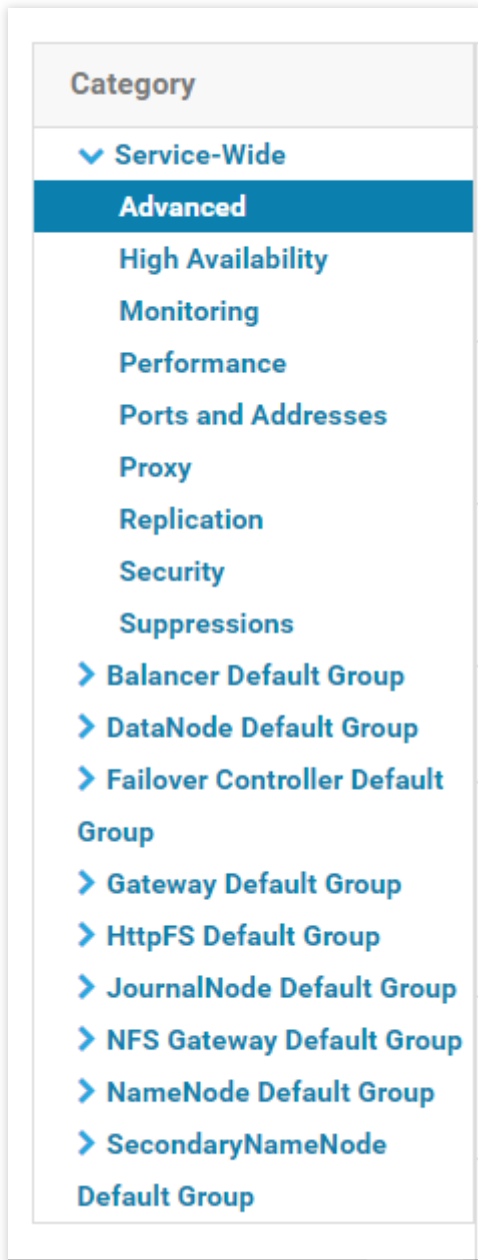
CDH 5.16.1

Hadoop 2.6.0

# How to Use

## Configuring storage environment

1. Log in to the CDH management page.
2. On the homepage, select **Configuration** > **Service-Wide** > **Advanced** as shown below:



3. Specify your COSN settings in the configuration snippet `Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml`.

```
<property>
<name>fs.cosn.userinfo.secretId</name>
<value>AK***</value>
```

```

</property>
<property>
<name>fs.cosn.userinfo.secretKey</name>
<value></value>
</property>
<property>
<name>fs.cosn.impl</name>
<value>org.apache.hadoop.fs.CosFileSystem</value>
</property>
<property>
<name>fs.AbstractFileSystem.cosn.impl</name>
<value>org.apache.hadoop.fs.CosN</value>
</property>
<property>
<name>fs.cosn.bucket.region</name>
<value>ap-shanghai</value>
</property>

```

The following lists the required COSN settings (to be added to `core-site.xml` ). For other settings, see [Hadoop](#).

COSN Configuration Item	Value	Description
fs.cosn.userinfo.secretId	AKxxxx	API key information of the account
fs.cosn.userinfo.secretKey	Wpxxxx	API key information of the account
fs.cosn.bucket.region	ap-shanghai	Bucket region
fs.cosn.impl	org.apache.hadoop.fs.CosFileSystem	The implementation class of COSN for FileSystem, which is fixed at <code>org.apache.hadoop.fs.CosFileSystem</code>
fs.AbstractFileSystem.cosn.impl	org.apache.hadoop.fs.CosN	The implementation class of COSN for AbstractFileSystem, which is fixed at <code>org.apache.hadoop.fs.CosN</code>

4. Take action on your HDFS service by clicking. Now, the `core-site.xml` settings above will apply to servers in the cluster.

5. Place the latest SDK package of COSN in the path of the JAR package of the CDH HDFS service and replace the relevant information with the actual value as shown below:

```

cp hadoop-cos-2.7.3-shaded.jar /opt/cloudera/parcels/CDH-5.16.1-
1.cdh5.16.1.p0.3/lib/hadoop-hdfs/

```

#### Note:

The SDK JAR file needs to be put in the same location on each server in the cluster.

## Data migration

Use Hadoop Distcp to migrate your data from CDH HDFS to COSN. For details, see [Migrating Data Between HDFS and COS](#).

## Using COSN for big data suites

### 1. MapReduce

#### Directions

- (1) Configure HDFS settings as instructed in [Data migration](#) and put the JAR file of the COSN SDK in the correct HDFS directory.
- (2) On the CDH homepage, find YARN and restart the NodeManager service (recommended). You can choose not to restart it for the TeraGen command, but must restart it for the TeraSort command because of the internal business logic.

#### Sample

The example below shows TeraGen and TeraSort in Hadoop standard test:

```
hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar teragen -Dmapred.job.maps=500
-Dfs.cosn.upload.buffer=mapped_disk -Dfs.cosn.upload.buffer.size=-1 1099
cosn://examplebucket-1250000000/terasortv1/1k-input
```

```
hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar terasort -
Dmapred.max.split.size=134217728 -Dmapred.min.split.size=134217728 -
Dfs.cosn.read.ahead.block.size=4194304 -Dfs.cosn.read.ahead.queue.size=32
cosn://examplebucket-1250000000/terasortv1/1k-input cosn://examplebucket-
1250000000/terasortv1/1k-output
```

#### Note:

cosn:// Replace the content behind `schema`` with your own bucket path

### 2. Hive

#### 2.1 MR engine

#### Directions

- (1) Configure HDFS settings as instructed in [Data migration](#) and put the JAR file of the COSN SDK in the correct HDFS directory.
- (2) On the CDH homepage, find Hive and restart the HiveServer2 and HiveMetastore roles.

#### Sample

To query your actual business data, use the Hive command line to create a location as a partitioned table on CHDFS:

```
CREATE TABLE `report.report_o2o_pid_credit_detail_grant_daily` (
  `cal_dt` string,
  `change_time` string,
```

```
`merchant_id` bigint,  
`store_id` bigint,  
`store_name` string,  
`wid` string,  
`member_id` bigint,  
`meber_card` string,  
`nickname` string,  
`name` string,  
`gender` string,  
`birthday` string,  
`city` string,  
`mobile` string,  
`credit_grant` bigint,  
`change_reason` string,  
`available_point` bigint,  
`date_time` string,  
`channel_type` bigint,  
`point_flow_id` bigint)  
PARTITIONED BY (  
  `topicdate` string)  
ROW FORMAT SERDE  
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'  
STORED AS INPUTFORMAT  
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat '  
  OUTPUTFORMAT  
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat '  
LOCATION  
  'cosn://examplebucket-1250000000/user/hive/warehouse/report.db/report_o2o_pid_cre  
TBLPROPERTIES (  
  'last_modified_by'='work',  
  'last_modified_time'='1589310646',  
  'transient_lastDdlTime'='1589310646')
```

Perform a SQL query:

```
select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
```

The output is as shown below:



```

hive> select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
Query ID = root_20200713121818_3a911a0c-2496-4e7e-b59d-b2a26266a6ab
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1594351711155_0014, Tracking URL = http://hadoop01:8088/proxy/application_1594351711155_0014/
Kill Command = /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/hadoop/bin/hadoop job -kill job_1594351711155_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-07-13 12:18:19,189 Stage-1 map = 0%, reduce = 0%
2020-07-13 12:18:25,391 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 8.89 sec
2020-07-13 12:18:30,544 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 10.8 sec
MapReduce Total cumulative CPU time: 10 seconds 800 msec
Ended Job = job_1594351711155_0014
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 10.8 sec HDFS Read: 17383
HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 800 msec
OK
27677
Time taken: 19.128 seconds, Fetched: 1 row(s)

```

## 2.2 Tez engine

You need to import the COSN JAR file as part of a Tez tar.gz file. The following example uses apache-tez.0.8.5:

### Directions

- (1) Locate and decompress the Tez tar.gz file installed in the CDH cluster, e.g., /usr/local/service/tez/tez-0.8.5.tar.gz.
- (2) Put the COSN JAR file in the resulting directory, and then compress it into a new tar.gz file.
- (3) Upload this new file to the path as specified by tez.lib.uris, or simply replace the existing file with the same name.
- (4) On the CDH homepage, find Hive and restart HiveServer and HiveMetaStore.

## 3. Spark

### Directions

- (1) Configure HDFS settings as instructed in [Data migration](#) and put the JAR file of the COSN SDK in the correct HDFS directory.
- (2) Restart NodeManager.

### Sample

The following takes the `spark example word count` test conducted with COSN as an example.

```

spark-submit --class org.apache.spark.examples.JavaWordCount --executor-memory
4g --executor-cores 4 ./spark-examples-1.6.0-cdh5.16.1-hadoop2.6.0-
cdh5.16.1.jar cosn://examplebucket-1250000000/wordcount

```

The output is as shown below:

```
20/07/17 20:39:03 INFO cluster.YarnScheduler: Removed TaskSet 1.0, whose tasks have all completed, from pool
20/07/17 20:39:03 INFO scheduler.DAGScheduler: ResultStage 1 (collect at JavaWordCount.java:68) finished in 0.077 s
20/07/17 20:39:03 INFO scheduler.DAGScheduler: Job 0 finished: collect at JavaWordCount.java:68, took 6.533844 s
And: 4
day.: 1
God: 4
Let: 1
light:: 1
it: 1
light,: 1
evening: 1
was: 2
that: 1
light.: 1
be: 1
Day,: 1
said,: 1
darkness.: 1
he: 1
first: 1
there: 2
light: 2
Night.: 1
morning: 1
darkness: 1
were: 1
saw: 1
divided: 1
and: 4
good:: 1
from: 1
called: 2
the: 8
20/07/17 20:39:03 INFO ui.SparkUI: Stopped Spark web UI at http://10.0.0.12:4040
```

## 4. Sqoop

### Directions

(1) Configure HDFS settings as instructed in [Data migration](#) and put the JAR file of the COSN SDK in the correct HDFS directory.

(2) Put the JAR file of the COSN SDK in the Sqoop directory, for example, `/opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/sqoop/`.

(3) Restart NodeManager.

### Sample

For example, to export MySQL tables to COSN, refer to [Import/Export of Relational Database and HDFS](#).

```
sqoop import --connect "jdbc:mysql://IP:PORT/mysql" --table sqoop_test --
username root --password 123** --target-dir cosn://examplebucket-
1250000000/sqoop_test
```

The output is as shown below:

```

20/07/17 20:45:23 INFO mapreduce.Job: map 0% reduce 0%
20/07/17 20:45:29 INFO mapreduce.Job: map 100% reduce 0%
20/07/17 20:45:33 INFO mapreduce.Job: Job job_1594976906551_0021 completed successfully
20/07/17 20:45:33 INFO mapreduce.Job: Counters: 35
  File System Counters
    COSN: Number of bytes read=0
    COSN: Number of bytes written=104
    COSN: Number of read operations=0
    COSN: Number of large read operations=0
    COSN: Number of write operations=0
    FILE: Number of bytes read=0
    FILE: Number of bytes written=529146
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=295
    HDFS: Number of bytes written=0
    HDFS: Number of read operations=3
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=0
  Job Counters
    Launched map tasks=3
    Other local map tasks=3
    Total time spent by all maps in occupied slots (ms)=45464
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=11366
    Total vcore-milliseconds taken by all map tasks=11366
    Total megabyte-milliseconds taken by all map tasks=46555136
  Map-Reduce Framework
    Map input records=3
    Map output records=3
    Input split bytes=295
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=273
    CPU time spent (ms)=6820
    Physical memory (bytes) snapshot=1267851264
    Virtual memory (bytes) snapshot=19217276928
    Total committed heap usage (bytes)=6662127616
  File Input Format Counters
    Bytes Read=0
  File Output Format Counters
    Bytes Written=104
20/07/17 20:45:33 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 17.3912 seconds (0 bytes/s)
20/07/17 20:45:33 INFO mapreduce.ImportJobBase: Retrieved 3 records.
20/07/17 20:45:33 INFO fs.BufferPool: Close a buffer pool instance.
20/07/17 20:45:33 INFO fs.BufferPool: Begin to release the buffers.
[6] 1:cdh* 2:cdh2- 3:onlytest 4:bash 5:expect 6:expect 7:bash 8:vim 9:vim 10:expect 11:e

```

## 5. Presto

### Directions

(1) Configure HDFS settings as instructed in [Data migration](#) and put the JAR file of the COSN SDK in the correct HDFS directory.

(2) Put the JAR file of the COSN SDK in the Presto directory, for example,

```
/usr/local/services/cos_presto/plugin/hive-hadoop2 .
```

(3) Presto does not load the gson-2...jar JAR file (only used for CHDFS) from Hadoop Common, so you need to manually put it into the presto directory, for example, `/usr/local/services/cos_presto/plugin/hive-hadoop2`.

(4) Restart HiveServer, HiveMetaStore, and Presto.

### Sample

The example below queries the COSN scheme table as a Hive-created location:

```
select * from cosn_test_table where bucket is not null limit 1;
```

### Note:

`cosn_test_table` is a table with location as `cosn scheme`.

The output is as shown below:

```
[root@TENCENT64 /usr/local/services/cos_presto_logging-1.0/plugin]# presto-cli --server 10.0.0.0:8080 --catalog hive --schema inventory_search --debug;
presto:inventory_search> select * from oppo_list_gz_table where bucket is not null limit 1;
 appid | bucket | path | size
-----+-----+-----+-----
 125 | migrate80105841qq1 | 127454151/20180125/3/5a9df97740b54ab2bac813a606c687d0 | 1167800
(1 row)
(END)
```

# COS Ranger Permission System Solution

Last updated : 2024-03-25 15:16:26

## Background

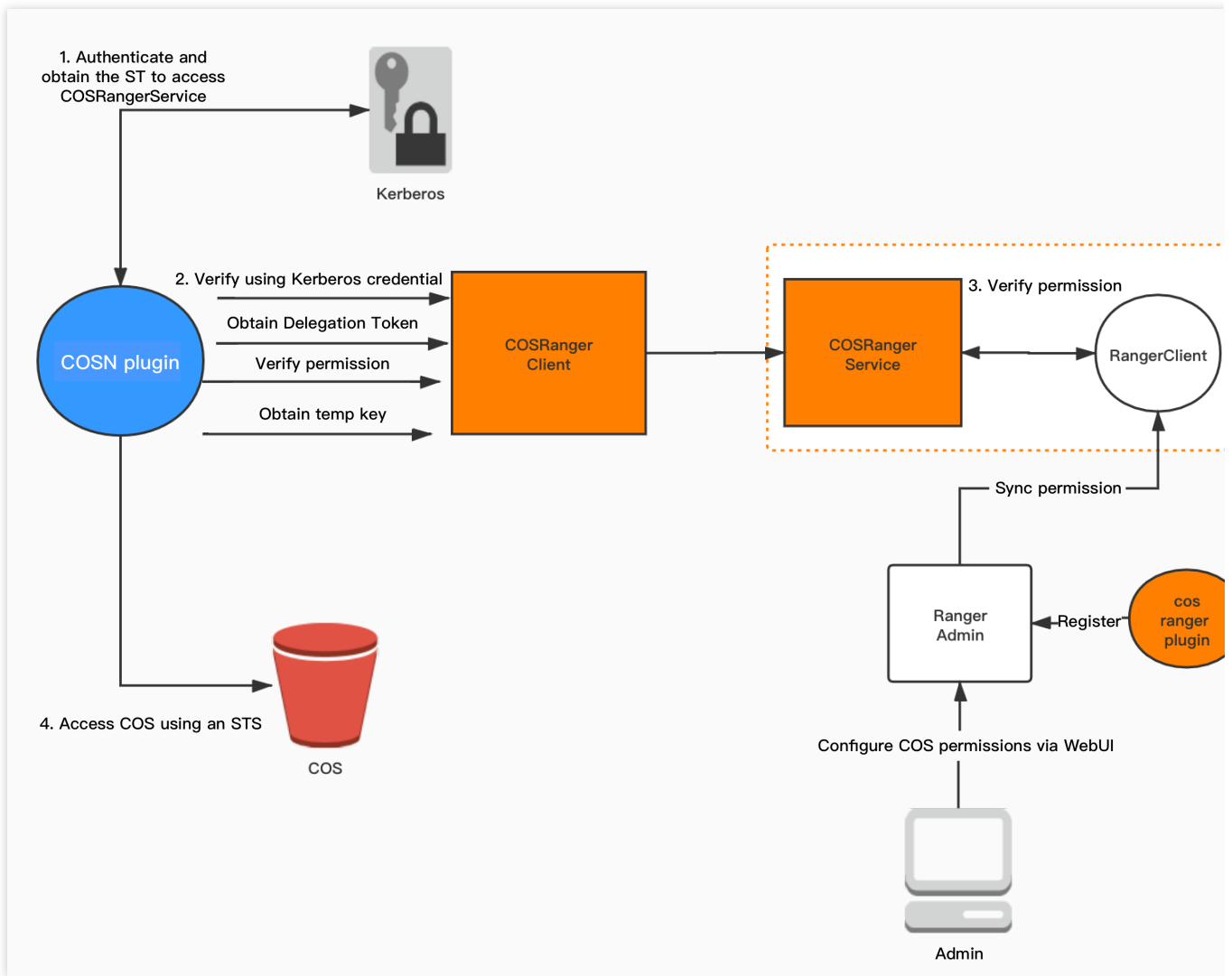
Hadoop Ranger is a permission solution for big data scenarios. By adopting the computing/storage separation mode, you can host data in COS. However, COS uses the CAM permission system, meaning that user roles and permission policies may be different from those of Hadoop Ranger. Therefore, a solution is introduced here to integrate COS with Ranger.

## Strengths

Fine-grained and Hadoop-compatible permission control allow you to centrally manage permissions for big data components and data hosted in the cloud.

There is no need to set keys in core-site on the plugin side; instead, keys are centrally set in COS Ranger Service to avoid key plaintext exposure.

## Solution Architecture



In the Hadoop permission system, authentication is offered by Kerberos and authorized by Ranger. On the basis of this, the following components are provided to support the COS Ranger permission solution:

1. COS Ranger Plugin: As a service defining plugin used on the Ranger server, it provides the COS service description on the Ranger side, including permission types and definitions of required parameters (such as bucket and region). Once it is deployed, you can set permission policies on the Ranger control panel.
2. COS Ranger Service: It integrates the Ranger client, periodically syncs permission policies from the Ranger server, and verifies the permission locally after receiving an authentication request. In addition, it also provides `DelegationToken` generation and renewal APIs in Hadoop, all of which are defined through Hadoop IPC.
3. Cos Ranger Client: It is dynamically loaded by the COSN plugin to forward permission verification requests to COS Ranger Service.

## Environment Deployment

Hadoop environment

ZooKeeper, Ranger, and Kerberos (if there are authentication requirements)

**Note:**

The components above are open-source and stable. You can install them on your own.

## Component Deployment

Deploy components in the following sequence: COS Ranger Plugin, COS Ranger Service, COS Ranger Client, COSN.

Deploying COS Ranger Plugin

Deploying COS Ranger Service

Deploying COS Ranger Client

Deploying COSN

COS Ranger Plugin extends the service types in the Ranger Admin console. You can set the operation permissions related to COS in the Ranger console.

### Code address

You can get the code from the `ranger-plugin` directory at [GitHub](#).

### Version

v1.0 or later.

### Deployment steps

1. Create a `cos` directory in the service definition directory of Ranger (note: make sure that the directory permissions include at least `x` and `r` permissions).

a. For an EMR environment, the path is `ranger/ews/webapp/WEB-INF/classes/ranger-plugins`.

b. For a self-built Hadoop environment, you can find the components connected to the Ranger service through `find hdfs` in the `ranger` directory in order to find the location of the directory.

```
[hadoop@10 ranger-plugins]$ ls -l
total 68
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 atlas
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 chdfs
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 cos
drwxr-xr-x 2 hadoop hadoop 4096 Feb 25 2020 hbase
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 hdfs
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 hive
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 kafka
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 kms
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 Knox
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 kylin
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 nifi
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 nifi-registry
drwxr-xr-x 2 hadoop hadoop 4096 Aug 5 20:48 presto
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 solr
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 sqoop
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 storm
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19 2020 yarn
```

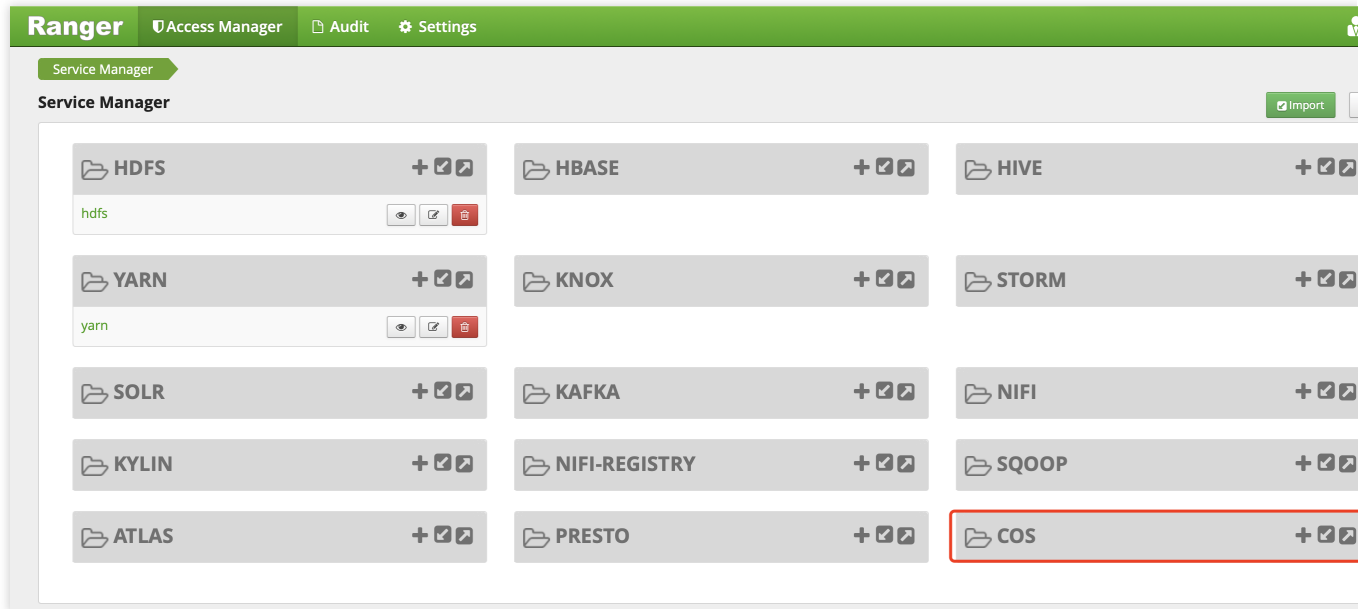
2. Place `cos-chdfs-ranger-plugin-xxx.jar` (with at least the `r` permission) and `cos-ranger.json` files in the COS directory. You can get them from [GitHub](#).
3. Restart Ranger.
4. Register the COS service on Ranger. You can refer to the following command:

```
## Create the service. The Ranger admin account and password as well as the
Ranger service address should be specified.
## For an EMR cluster, the admin user is `root`, and the password is the root
password set when the EMR cluster is created. Replace the IP of the Ranger
service with the primary node IP of EMR.
adminUser=root
## The password set during EMR cluster creation, which is also the login
password of the Ranger web service.
adminPasswd=xxxxxx
## If the Ranger service has multiple primary nodes, select any of them.
rangerServerAddr=10.0.0.1:6080
## Specify the .json file in step 2 as `-d` in the command.
curl -v -u${adminUser}:${adminPasswd} -X POST -H "Accept:application/json" -H
"Content-Type:application/json" -d @./cos-ranger.json
http://${rangerServerAddr}/service/plugins/definitions
## To delete the service just defined, you need to pass in the service ID
returned during creation.
serviceId=102
curl -v -u${adminUser}:${adminPasswd} -X DELETE -H "Accept:application/json" -H
"Content-Type:application/json"
```



```
http://${rangerServerAddr}/service/plugins/definitions/${serviceId}
```

5. After the service is successfully created, you can see the COS service in the Ranger console.



6. Click + next to the COS service to define a new service instance. The service instance name is customizable; for example, you can enter `cos` or `cos_test`. The service configuration is as follows:

The screenshot shows the 'Edit Service' configuration page in the Tencent Cloud Ranger console. The page has a green header with 'Ranger' and navigation tabs for 'Access Manager', 'Audit', and 'Settings'. Below the header, there are breadcrumb links for 'Service Manager' and 'Edit Service'. The main content area is titled 'Edit Service' and is divided into two sections: 'Service Details' and 'Config Properties'.

**Service Details:**

- Service Name \*:
- Description:
- Active Status:  Enabled  Disabled
- Select Tag Service:

**Config Properties:**

Add New Configurations

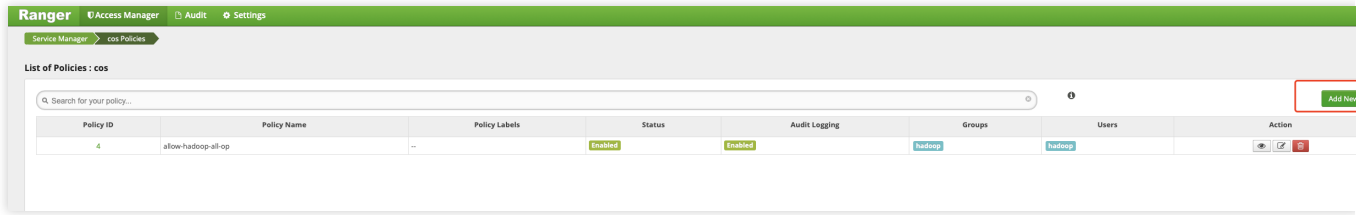
Name	Value
<input type="text" value="policy.grantrevoke.auth.users"/>	<input type="text" value="hadoop"/>

You need to set the username subsequently used to start COS Ranger Service (i.e., the user allowed to pull permission policies) as `policy.grantrevoke.auth.users`. We generally recommend you set it to `hadoop`.

7. Click the newly created COS service instance.



Add a policy.



8. On the page that is displayed, configure the following parameters:

**Bucket:** Bucket name, such as `examplebucket-1250000000`, which can be viewed in the [COS console](#).

**Path:** Path of the COS object. Note that it does not start with a slash (/).

**include:** Indicates whether the set permission applies to the specified path itself or other paths except it.

**recursive:** Indicates that the permission applies to not only the specified path but also the subpaths under it (i.e., recursive subpaths). It is usually used when the path is set as a directory.

**User/Group:** Username and user group in logical OR relationship; that is, the operation is authorized as long as the username or user group condition is met.

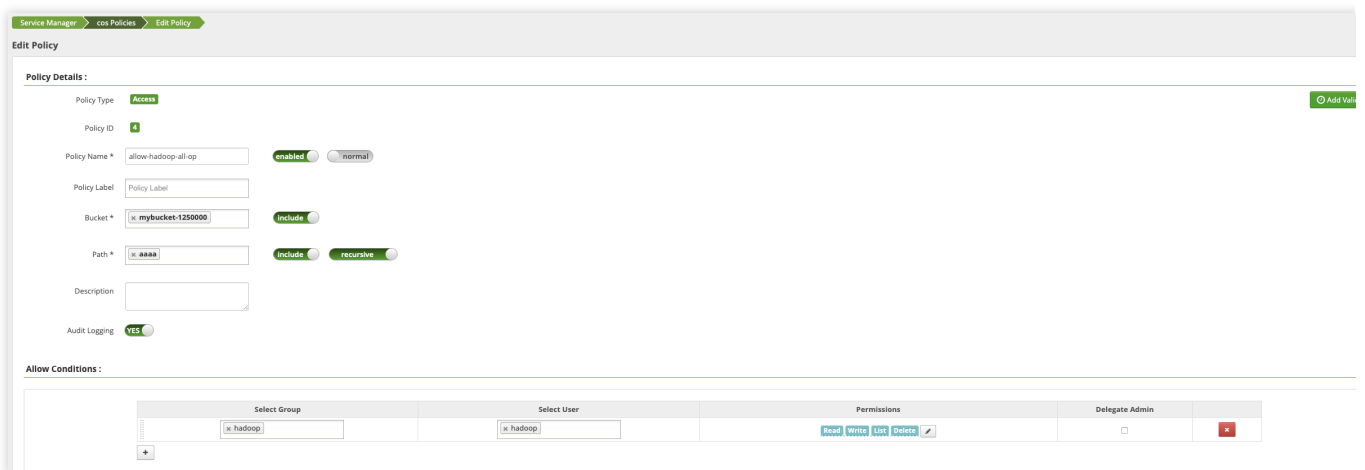
#### Permissions:

**Read:** Read operation, which corresponds to the GET and HEAD operations in COS, such as downloading objects and querying object metadata.

**Write:** Write operation, which corresponds to the PUT operation in COS, such as uploading objects.

**Delete:** Deletion operation, which corresponds to the object deletion operation in COS. To rename a path in Hadoop, you need to have the deletion permission for the original path and write permission for the new path.

**List:** Traversal permission, which corresponds to the `List Object` operation in COS.



COS Ranger Service is the core of the entire permission system. It is responsible for integrating the Ranger client and receiving its authentication requests, token generation and renewal requests, and temporary key generation requests. It is also where sensitive information (Tencent Cloud key information) resides. Generally, it is deployed on a bastion host, and only the cluster admin is allowed to manipulate it and view its configuration.

COS Ranger Service supports the one-primary-multiple-secondary HA deployment. `DelegationToken` can be persisted to HDFS, and the primary and secondary nodes can be determined by ZooKeeper lock obtaining. Then, the primary node (leader) will write the address to ZooKeeper so that COS Ranger Client can perform address routing.

## Code address

You can get the code from the `cos-ranger-server` directory at [GitHub](#).

## Version

v5.1.2 or later.

## Deployment steps

1. Copy the code of COS Ranger Service to the servers in the cluster. We recommend you configure at least two servers (one primary server and one secondary server) in the production environment. As sensitive information is involved, we recommend you use bastion hosts or servers with tight permission control.
2. Modify the relevant configuration items in the `cos-ranger.xml` file. Those that must be modified are as follows. For the description of the configuration items, see the comments in the file. You can get the configuration file in the `cos-ranger-service/conf` directory at [GitHub](#).

`qcloud.object.storage.rpc.address`

`qcloud.object.storage.status.port`

`qcloud.object.storage.enable.cos.ranger`

`qcloud.object.storage.zk.address` (ZooKeeper address. After COS Ranger Service starts, it will be registered in ZooKeeper.)

`qcloud.object.storage.cos.secret.id`

`qcloud.object.storage.cos.secret.key`

3. Modify the relevant configuration items in the `ranger-cos-security.xml` file. Those that must be modified are as follows. For the description of the configuration items, see the comments in the file. You can get the configuration file in the `cos-ranger-service/conf` directory at [GitHub](#).

`ranger.plugin.cos.policy.cache.dir`

`ranger.plugin.cos.policy.rest.url`

`ranger.plugin.cos.service.name`

4. In the `start_rpc_server.sh` file, modify the configuration of `hadoop_conf_path` and `java.library.path`, which correspond to the directory of the Hadoop configuration file (for example, `core-site.xml` or `hdfs-site.xml`) and Hadoop native libraries, respectively.

5. Run the following command to start the service:

```
chmod +x start_rpc_server.sh
nohup ./start_rpc_server.sh &> nohup.txt &
```

6. If the start failed, check whether an error message is contained in the error log.

7. COS Ranger Service supports displaying the HTTP port status (port name:

`qcloud.object.storage.status.port` ; default value: `9998` ). You can run the following command to get the status information, such as whether the leader is contained and the authentication statistics:

```
# Replace `10.xx.xx.xxx` with the IP address of the server deployed with the
Ranger service.
# Replace `9998` in the command with the value of
`qcloud.object.storage.status.port` in the configuration file.
curl -v http://10.xx.xx.xxx:9998/status
```

If only one COS Ranger Service node is deployed, you can see that the current node becomes the leader in the above API response.

If multiple COS Ranger Service nodes are deployed, you can see that another node becomes the leader in the above API response. After all nodes are restarted, you can see that the first restarted node becomes the leader.

COS Ranger Client is dynamically loaded by the Hadoop COSN plugin. It is a proxy that encapsulates all COS Ranger Service access requests, such as obtaining temporary keys, obtaining tokens, and performing authentication.

### Code address

You can get the code from the `cos-ranger-client` and `cosn-ranger-interface` directories at [GitHub](#).

### Version

v5.0 or later for COS Ranger Client.v1.0.4 or later for COSN Ranger Interface.

### Deployment directions

1. Copy the JAR packages of COS Ranger Client and COSN Ranger Interface to the same directory as COSN (in the `/usr/local/service/hadoop/share/hadoop/common/lib/` directory generally). Be sure to select JAR packages with the same major version as Hadoop and make sure that they have the read permission.

2. Add the following configuration in `core-site.xml` :

```
...
<configuration>
  <!--*****Required configuration*****-->
  <!-- Address of the COS Ranger server deployed in the previous step
-->

  <property>
    <name>qcloud.object.storage.ranger.service.address</name>
    <value>10.0.0.8:9999,10.0.0.10:9999</value>
  </property>

  <!--***Optional configuration***-->
  <!-- Set the Kerberos credential used in COS Ranger Service. It
should be the same as that configured in COS Ranger Service. If verification is
not involved, you can skip this configuration. -->
```

```
<property>
  <name>qcloud.object.storage.kerberos.principal</name>
  <value>hadoop/_HOST@EMR-XXXX</value>
</property>

<!--***Optional configuration***-->
<!-- IP address path of the Ranger server recorded in ZooKeeper. The
default value is used here. The value must the same as that configured in COS
Ranger Service. -->
  <property>
    <name>qcloud.object.storage.zk.leader.ip.path</name>
    <value>/ranger_qcloud_object_storage_leader_ip</value>
  </property>
  <!-- IP address path of the COS Ranger Service follower recorded in
ZooKeeper. The default value is used here. The value must the same as that
configured in COS Ranger Service. Both the primary and secondary nodes provide
services at the same time. -->
  <property>
    <name>qcloud.object.storage.zk.follower.ip.path</name>
    <value>/ranger_qcloud_object_storage_follower_ip</value>
  </property>
</configuration>
...

```

## Version

v8.0.1 or later.

## Deployment directions

For detailed directions on the deployment of COSN, see [Hadoop](#). Note the following:

1. If Ranger is used, the key information of `fs.cosn.userinfo.secretId` and `fs.cosn.userinfo.secretKey` does not need to be configured. COSN will get temporary keys through COS Ranger Service.
2. `fs.cosn.credentials.provider` needs to be set to `org.apache.hadoop.fs.auth.RangerCredentialsProvider`, so that the verification and authentication can be passed through Ranger. Below is an example:

```
...
<property>
  <name>fs.cosn.credentials.provider</name>
  <value>org.apache.hadoop.fs.auth.RangerCredentialsProvider</value>
</property>
...

```

## Verification

1. Use Hadoop commands to perform operations involving COSN access to check whether the current operations comply with the permissions set by the root account. Below is an example:

```
# Replace the bucket, path, and other information with that of the root account.
hadoop fs -ls cosn://examplebucket-1250000000/doc
hadoop fs -put ./xxx.txt cosn://examplebucket-1250000000/doc/
hadoop fs -get cosn://examplebucket-1250000000/doc/exampleobject.txt
hadoop fs -rm cosn://examplebucket-1250000000/doc/exampleobject.txt
```

2. Use MR Job for verification. Before the verification, be sure to restart related services such as YARN and Hive.

## FAQs

### Do I have to install Kerberos?

Kerberos meets the authentication needs. If users in a cluster are trusted, and the purpose of the authentication is only to avoid maloperations performed by unauthorized users, you can skip installing Kerberos and only use Ranger for authentication. As a matter of fact, Kerberos also compromises the performance. Therefore, you can balance your needs for security and performance as needed. If authentication is needed, you can enable Kerberos and then configure COS Ranger Service and COS Ranger Client.

### What would happen if I enable Ranger but don't set any policy or no policy is matched?

If no policy is matched, the operation will be denied by default.

### Can a sub-account configure the key in COS Ranger Service?

Yes. A sub-account with relevant permissions of the manipulated bucket can generate a temporary key for the COSN plugin to perform corresponding operations. Normally, you can grant all permissions of the bucket to the configured key.

### How do I update a temporary key? Do I need to get it from COS Ranger Service every time before I access COS?

The temporary key is cached in the COSN plugin. It will be periodically updated asynchronously.

### What should I do if the policy modified on the Ranger page doesn't take effect?

Decrease the `ranger.plugin.cos.policy.pollIntervalMs` value (in milliseconds) in the `ranger-cos-security.xml` file and restart COS Ranger Service. After the policy is tested, we recommend you change it back

---

to the original value (if the time interval is too short, the polling frequency will be high, causing a high CPU utilization).



# Connecting Oceanus to COS

Last updated : 2024-03-25 15:16:26

## Oceanus Overview

Oceanus is a powerful real-time analysis tool in the big data ecosystem. With it, you can easily build various applications in just a few minutes, such as website clickstream analysis, targeted ecommerce recommendation, and IoT. Oceanus is developed based on Apache Flink and provides fully managed cloud services, so you don't need to care about the Ops of infrastructure. It can also be connected to data sources in the cloud for a complete set of supporting services.

Oceanus comes with a convenient console for you to write SQL analysis statements, upload and run custom JAR packages, and manage jobs. Based on the Flink technology, it can achieve a sub-second processing latency in datasets at the petabyte level.

This document describes how to connect Oceanus to COS. Currently, Oceanus is available in the dedicated cluster mode, where you can run various jobs and manage related resources in your own cluster.

## Prerequisites

### Creating Oceanus cluster

Log in to the [Oceanus console](#) and create an Oceanus cluster.

### Creating COS bucket

1. Log in to the [COS console](#).
2. Click **Bucket List** on the left sidebar.
3. Click **Create Bucket** to create a bucket as instructed in [Creating a Bucket](#).

#### Note:

When you write data to COS, the Oceanus job must run in the same region as COS.

## Directions

Go to the [Oceanus console](#), create an SQL job, and select a cluster in the same region as COS.

### 1. Create a source

```
CREATE TABLE `random_source` (
```

```

f_sequence INT,
f_random INT,
f_random_str VARCHAR
) WITH (
'connector' = 'datagen',
'rows-per-second'='10',           -- Number of date rows generated per sec
'fields.f_sequence.kind'='random', -- Random number
'fields.f_sequence.min'='1',      -- Minimum sequential number
'fields.f_sequence.max'='10',    -- Maximum sequential number
'fields.f_random.kind'='random',  -- Random number
'fields.f_random.min'='1',       -- Minimum random number
'fields.f_random.max'='100',     -- Maximum random number
'fields.f_random_str.length'='10' -- Random string length
);

```

**Note:**

Here, the built-in connector `datagen` is selected. Select a data source based on your actual business needs.

**2. Create a sink**

```

-- Replace `<bucket name>` and `<folder name>` with your actual bucket and folder name
CREATE TABLE `cos_sink` (
  f_sequence INT,
  f_random INT,
  f_random_str VARCHAR
) PARTITIONED BY (f_sequence) WITH (
  'connector' = 'filesystem',
  'path'='cosn://<bucket name>/<folder name>/',      --- Directory path
  'format' = 'json',                                --- Format of written data
  'sink.rolling-policy.file-size' = '128MB',       --- Maximum file size
  'sink.rolling-policy.rollover-interval' = '30 min', --- Maximum file write interval
  'sink.partition-commit.delay' = '1 s',          --- Partition commit delay
  'sink.partition-commit.policy.kind' = 'success-file' --- Partition commit mode
);

```

**Note:**

For more `WITH` parameters of a sink, see "Filesystem (HDFS/COS)".

**3. Configure the business logic**

```

INSERT INTO `cos_sink`
SELECT * FROM `random_source`;

```

**Note:**

This is for demonstration only and has no actual business purposes.

#### 4. Set job parameters

Select `flink-connector-cos` as the **Built-in Connector** and configure the COS URL in **Advanced**

**Parameters** as follows:

```
fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvider
fs.cosn.bucket.region: <COS region>
fs.cosn.userinfo.appid: <COS user appid>
```

The job is configured as follows:

Replace `<COS region>` with your actual COS region, such as `ap-guangzhou` .

Replace `<COS user appid>` with your actual `APPID` , which can be viewed in [Account Center](#).

**Note:**

For more information on job parameter settings, see "File System (HDFS/COS)".

#### 5. Start the job

Click **Save** > **Check Syntax** > **Release Draft**, wait for the SQL job to start, and go to the corresponding COS directory to view the written data.

# Using COS in the Third-party Applications

## Use the general configuration of COS in third-party applications compatible with S3

Last updated : 2024-03-25 15:16:26

Amazon Simple Storage Service (S3) is one of the earliest cloud services launched by AWS. After many years of development, the S3 protocol has become a de facto standard in the object storage field. Tencent Cloud Object Storage (COS) provides an S3-compatible implementation scheme, so you can directly use the COS service in most S3-compatible applications. This document describes how to configure such applications to use COS.

## Prerequisites

### Checking whether the application can use COS

An application with `S3 Compatible` in its description can use COS in most cases. If you find that some of its features cannot work properly, [contact us](#) for assistance, and be sure to indicate that you followed the steps in this document and provide information such as the application name and screenshots.

If your application description only states that `Amazon S3` is supported, it means that the application can use the S3 service, but whether it can use COS needs to be further evaluated in relevant configurations as detailed below.

### Preparing COS service

#### Step 1. Sign up for a Tencent Cloud account

(If you already have a Tencent Cloud account, skip this step.)

#### Step 2. Verify your identity

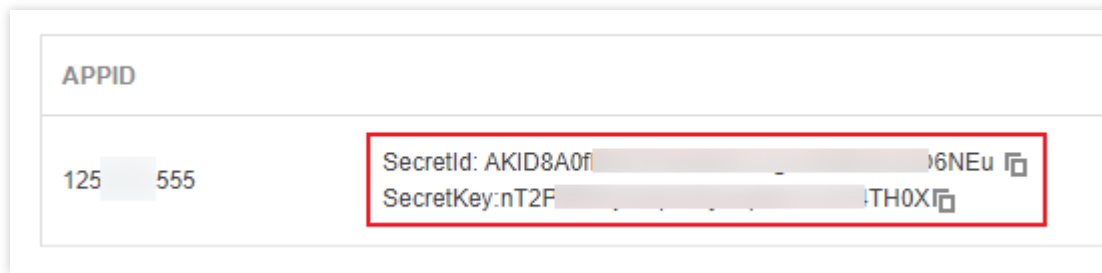
(If you have already done so, skip this step.)

For more information on how to verify your identity, see [Identity Verification Guide](#).

#### Step 3. Activate the COS service

#### Step 4. Prepare the APPID and access key

Get and note down the **APPID**, **SecretId**, and **SecretKey** on the [API Key](#) page in the CAM console.



### Step 5. Create a bucket

Create a COS bucket as instructed in [Creating a Bucket](#).

Some applications have a built-in process for creating buckets. If you want such applications to create buckets, skip this step.

## Configuring COS Service in Application

### Basic configuration

Most applications have similar configuration items for using a storage service. The common names and descriptions of these configuration items are as listed below:

#### Note:

If you have any questions during the configuration, [contact us](#) for assistance, and be sure to indicate that you followed the steps in this document and provide information such as the application name and screenshots.

Common Configuration Item Name	Description
Provider, service provider, storage service provider, storage provider, etc.	<p>Select which storage service the application should use. There may be the following cases:</p> <p>If an option has text like S3 Compatible Storage/S3 Compatible, then it will be used first.</p> <p>If an option only has text like Amazon Web Services/AWS/Amazon S3, then use it but pay attention to our further instructions during configuration.</p> <p>If there is no similar option, but the application description mentions that the application supports S3 services or S3-compatible services, then you can continue with the configuration below, but you also need to pay attention to our further instructions.</p> <p>In other cases, the application may not be able to use COS.</p>
Service endpoint, service address, service URL, endpoint, custom	<p>This indicates the address of an S3-compatible service. If you use COS, enter the COS service address here in the format of <code>cos.&lt;Region&gt;.myqcloud.com</code> or <code>https://cos.&lt;Region&gt;.myqcloud.com</code></p>

<p>endpoint, server URL, etc.</p>	<p>.Whether <code>https://</code> needs to be entered is determined by the application, and you can make some attempts by yourself. Here, <code>&lt;Region&gt;</code> indicates the availability region of COS.</p> <p>In the application, you can only create or select a bucket in the region specified in the service address.</p> <p>For example, if your bucket is in Guangzhou region, the service address should be configured as <code>cos.ap-guangzhou.myqcloud.com</code>; otherwise, you cannot find the bucket in Guangzhou in the application.</p> <p>If only Amazon S3 can be selected as the application service provider and the service address can be configured, then you can change the service address to the aforementioned <code>cos.&lt;Region&gt;.myqcloud.com</code> or <code>https://cos.&lt;Region&gt;.myqcloud.com</code>.</p> <p>If the service address cannot be configured or there is no such configuration item, the application cannot use COS.</p>
<p>Access key, access key ID, etc.</p>	<p>Enter the SecretId obtained in <a href="#">step 4</a>.</p>
<p>Secret key, secret, secret access key, etc.</p>	<p>Enter the SecretKey obtained in <a href="#">step 4</a>.</p>
<p>Region, etc.</p>	<p>Select "Default", "Auto", or "Automatic".</p>
<p>Bucket, etc.</p>	<p>You can select or enter the name of an existing bucket in the format of <code>&lt;BucketName-APPID&gt;</code>, such as <code>examplebucket-1250000000</code>. Here, <code>BucketName</code> is the name you entered when you created the bucket in <a href="#">step 5</a>, and APPID is the `APPID` obtained in <a href="#">step 4</a>.</p> <p>As described above, the bucket must be in the region specified by the service address, and buckets in other regions will not be listed or cannot work properly. If you need to create a bucket, the name of the new bucket should be in the format of <code>&lt;BucketName-APPID&gt;</code> as mentioned above; otherwise, it cannot be properly created.</p>

## Other advanced configuration items

In addition to the above basic configuration items, some applications have other advanced configuration items. The following describes some COS features for you to better use COS in such applications.

### Service port and protocol

COS supports both HTTP and HTTPS protocols, with the default ports 80 and 443 used by default. For security considerations, we recommend you use COS over the HTTPS protocol preferably.

### Path-Style and Virtual Hosted-Style

COS supports both styles.

### AWS v2 and AWS v4 signatures

COS supports both signature formats.

## Summary

COS does not guarantee full compatibility with S3. If you have any questions when using COS in your application, [contact us](#) for assistance, and be sure to indicate that you followed the steps in this document and provide information such as the application name and screenshots.

# Storing Remote WordPress Attachments to COS

Last updated : 2024-03-25 15:16:26

## Overview

[WordPress](#) is a blogging platform built with PHP. You can use it to build your own website on a server that supports PHP and MySQL databases, or simply as a content management system (CMS).

WordPress is a powerful, scalable, and easy-to-expand platform with a wide range of plugins. With third-party plugins, it offers everything that a website should have.

This document describes how to use a plugin to store remote attachments from the WordPress media library in [COS](#). Since COS is highly scalable, reliable, secure, and cost-effective, storing your media library attachments in COS offers the following benefits:

Higher reliability for your attachments.

No need to prepare additional storage capacity on your server for attachments.

Faster access to image attachments through the COS server rather than taking up downstream bandwidth/increasing the traffic on your own server.

Accelerated user access to image attachments through [CDN](#).

## Prerequisites

1. You have created a bucket; otherwise, create one as instructed in [Creating Bucket](#).
2. You have created a server such as a CVM instance as instructed in [Cloud Virtual Machine](#).

## Directions

### Deploying a WordPress website

To quickly build a WordPress website in CVM, you can deploy it via an image or manually. If you have high requirements for extensibility of your business website, you can deploy it manually as instructed in the following documents:

[Building a WordPress Website \(Linux\)](#)

[Building a WordPress Website \(Windows\)](#)

You can deploy a WordPress website via an image easily as follows:



1. Deploy a WordPress website via an image.
  - 1.1 Log in to the [CVM console](#) and click **Create Instance** on the instance management page.
  - 1.2 Select a model as prompted. In **Instance Configuration > Image**, click **Image Marketplace** and select **Select from image marketplace**.
  - 1.3 In the **Image Marketplace** pop-up window, select **Basic Software** and enter **wordpress** for search.
  - 1.4 Select an image as needed. Here, **WordPress Blog Program\_v5.5.3(CentOS | LAMP)** is selected. Then, click **Free Trial**.
  - 1.5 After purchase, log in to the CVM console, associate the newly created instance with a security group, and open port 80 in the inbound rules.
2. On the CVM instance management page, copy the **public IP** of the instance and access `http://public IP/wp-admin` in your local browser to start installing the WordPress website:
  - 2.1 Select the WordPress language and click **Continue**.
  - 2.2 Enter the WordPress website title and admin username, password, and email address.
  - 2.3 Click **Install WordPress**.
  - 2.4 Click **Log In**.
3. Upgrade WordPress to v6.0.2.

On the left sidebar, click **Dashboard > Updates** to update WordPress to v6.0.2.

## Creating a COS bucket

1. Create a **Public Read/Private Write** bucket as instructed in [Creating Buckets](#), preferably in the same region as the CVM instance where WordPress is running.
2. Locate the bucket you created on the **Bucket List** page and click its name to enter the details page.
3. Click **Overview** on the left sidebar. Then, find and record the endpoint.

## Installing and configuring the plugin

You can install the plugin in the plugin library or via the source code.

### Installation in the plugin library (recommended)

On the WordPress backend, click **Plugins**, directly search for the **tencentcloud-cos** plugin, and click **Install Now**.

### Installation via the source code

Download the plugin source code, upload it to the WordPress plugin directory `wp-content/plugins`, and run it on the backend.

The following steps take Ubuntu as an example to describe how to install a plugin:

1. Go to the parent directory of `wp-content` :

```
cd /var/www/html/
```

2. Add permissions:

```
chmod -R 777 wp-content
```

### 3. Create a plugin directory:

```
cd wp-content/plugins/  
mkdir tencent-cloud-cos  
cd tencent-cloud-cos
```

### 4. Download the plugin to the plugin directory:

```
wget https://cos5.cloud.tencent.com/cosbrowser/code/tencent-cloud-cos.zip  
unzip tencent-cloud-cos.zip  
rm tencent-cloud-cos.zip -f
```

### 5. Click **Plugins** on the left sidebar, and you can see the plugin. Click **Activate** to activate it.

## Configuring the plugin

Configure the COS bucket information in the tencent-cloud-cos plugin:

1. Click **Settings** to configure the tencent-cloud-cos plugin.
2. Configure the COS information as follows:

Configuration Item	Value
SecretId and SecretKey	Enter the access key information, which can be created and obtained on the <a href="#">Manage API Key</a> page.
Region	The region selected during bucket creation
Bucket Name	The bucket name customized during bucket creation, such as `examplebucket-1250000000`
Endpoint	The COS bucket's default domain name, which is automatically generated when the bucket is created. Buckets residing in different regions have different default domain names. To view the default domain name, go to the <a href="#">COS console</a> , click the name of the target bucket, and view it in Overview > Domain Information.
Auto-Rename	This option can automatically rename files uploaded to COS based on the specified format to avoid conflicts with existing files with the same name.
Do Not Save Locally	After this option is enabled, source files will not be retained locally.
Retain Remote File	After this option is enabled, if a file is deleted, only the local file copy will be deleted, and the file copy in the remote COS bucket will still be retained in case you want to recover it.
Forbid	After this option is enabled, thumbnail files will not be uploaded.

Thumbnail	
Cloud Infinite	After the CI service is enabled, you can perform various operations on images, such as editing, compression, format conversion, and watermarking. For more information, see <a href="#">Cloud Infinite</a> .
File Moderation	After file moderation is enabled, it intelligently moderates the multimedia content of images, videos, audios, text, files, and webpages. It helps you effectively identify non-compliant content such as pornographic, vulgar, illegal, disgusting, and offensive information to avoid operational risks.
File Preview	After file preview is enabled, it converts files into images, PDF files, or HTML5 pages to display the file content on webpages. For more information, see <a href="#">File Preview Overview</a> .
Debugging	This feature logs errors, exceptions, and warnings.

3. After completing the configuration, click **Save**.

## Verifying WordPress Attachment Storage to COS

You can create a post with an image in WordPress and check whether the image is stored in COS.

1. On the WordPress dashboard, click **Posts** on the left sidebar to create a post with an image.
2. Edit the "Hello world!" post generated by WordPress by default.
3. Click **+** on the right.
4. Upload an image.
5. Then, check whether the URL of the uploaded image is a COS URL such as `https://wd-125000000.cos.ap-nanjing.myqcloud.com/2022/10/Start of Summer-1200x675.jpeg` in the format of `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/<ObjectKey>`, and if so, the image has been uploaded to the COS bucket.
6. Log in to the COS console, and you can see the newly uploaded image in the bucket.

### Note:

Once confirmed, sync your old WordPress resources to the COS bucket by using [COSCMD](#) or [COS Migration](#). **This step must be performed; otherwise, you will not be able to view these resources in COS.** Then, you can optionally enable origin-pull as instructed below in [Setting origin-pull](#).

## Others

1. Using CDN for access acceleration

To configure CDN acceleration for your bucket, see [Setting CDN Acceleration](#). You will then need to change the URL

prefix to the default or custom CDN acceleration domain name in the WordPress plugin settings.

## 2. Replacing the resource URL in the WordPress database

Unless it's a newly created WordPress site, you will need to use the plugin to replace the old resource URLs in your WordPress database with new ones (if it's your first time, we recommend backing up your information before proceeding with this step).

Enter the old domain name for the resource, e.g. `https://example.com/`.

Enter the current domain name for the resource, e.g. `https://img.example.com/`.

## 3. Setting cross-origin access

When you reference a WordPress resource link in a document, you may receive the following prompt on the Tencent

Cloud console: `No 'Access-Control-Allow-Origin' header is present on the requested resource`. This is most likely due to the fact that you haven't configured the `HTTP Header` parameter in your CORS settings. You can configure this parameter in one of the following two ways:

Configuring on the COS console

### Note:

For more information, please see [Setting Cross-Origin Access](#).

Configuring on the CDN console

To allow all domain names, configure the following:

```
Access-Control-Allow-Origin: *
```

To allow access for only your own domain name, configure the following:

```
Access-Control-Allow-Origin: https://example.com
```

## 4.

### Setting origin-pull

If you do not upload resources to the WordPress media library through the COS console, we recommend using COS origin-pull. For detailed directions, see [Setting Origin-Pull](#).

Once origin-pull is enabled, when a client accesses a source object in COS for the first time, if COS cannot hit the object, it will return a 302 HTTP status code and automatically redirect the request to the origin-pull address. Then, the origin will provide the object for access. Meanwhile, COS will copy this object from the origin and store it in the corresponding COS directory so that COS will be able to directly return the object to the client in subsequent requests.

# Storing Ghost Attachment to COS

Last updated : 2024-03-25 15:16:26

## Overview

[Ghost](#) is a Node.js-based framework for quickly building a blog website. You can use its official CLI tool to quickly generate your personal website and deploy it in CVM or Docker.

As a blog website, attachment upload is a necessary feature. Ghost stores attachments locally by default. This document describes how to save an attachment in [COS](#) through the plugin. Saving forum attachments in COS has the following benefits:

Higher reliability for your attachments.

No need to prepare additional storage capacity on your server for forum attachments.

Faster access to image attachments through the COS server rather than taking up downstream bandwidth/increasing the traffic on your own server.

Accelerated forum user access to image attachments through [CDN](#).

## Preparations

### Building a Ghost website

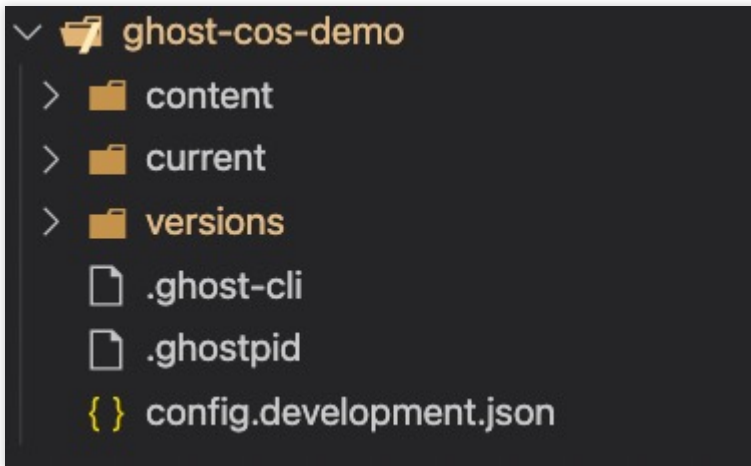
1. Install the [Node.js](#) environment.
2. Install ghost-cli.

```
npm install ghost-cli@latest -g
```

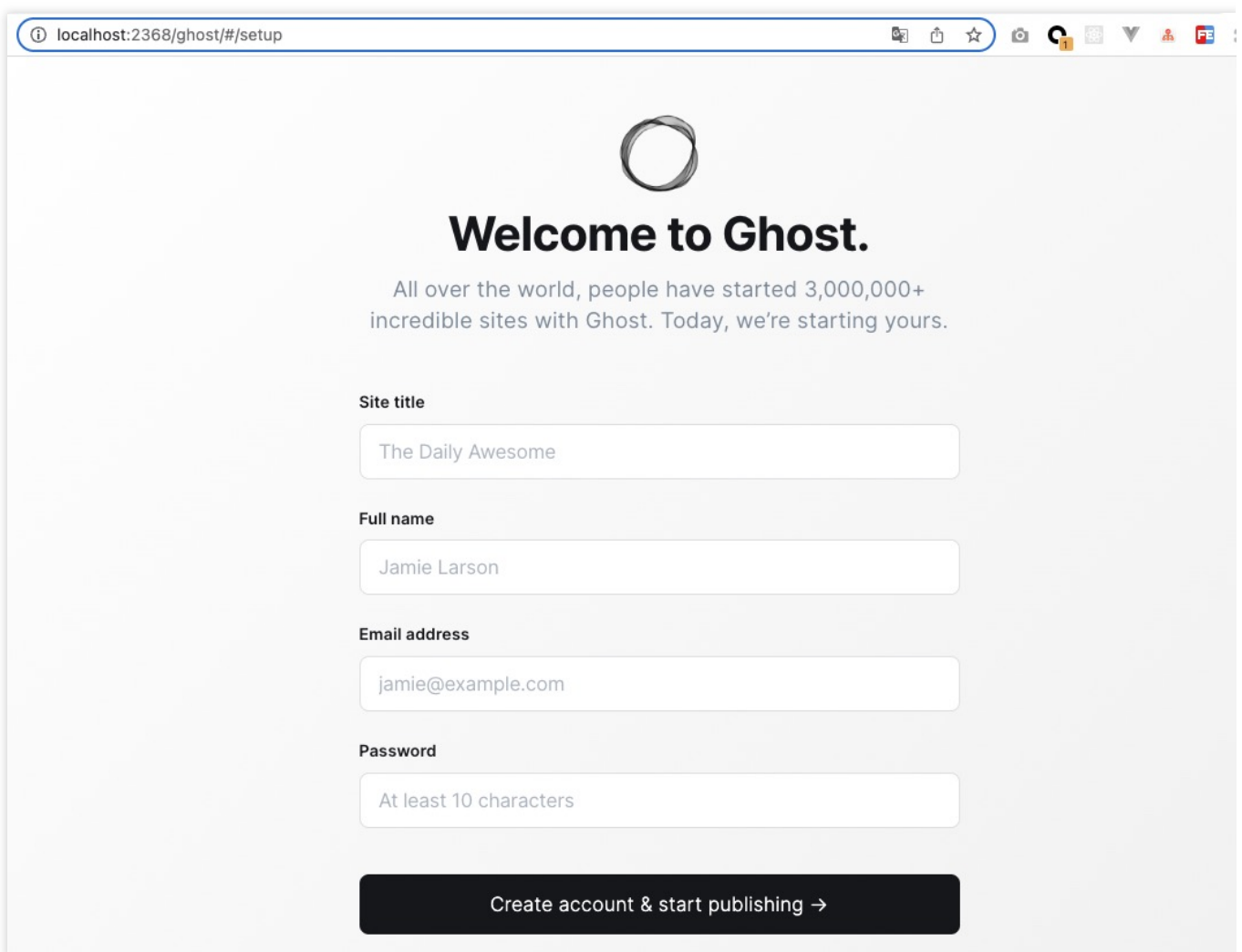
3. Create a project and run the following command in the project's root directory:

```
ghost install local
```

After successful creation, the project structure is as shown below:



4. Open the browser and access `localhost:2368`. On the sign-up page, sign up for an account and go to the management backend.



## Creating a COS bucket

1. In the [COS console](#), create a bucket with access permissions of **public read/private write** as instructed in [Creating Bucket](#).
2. Click **Security Management > CORS (Cross-Origin Resource Sharing)** and add a CORS configuration as instructed in [Setting Cross-Origin Resource Sharing \(CORS\)](#). You can use the following configuration to facilitate debugging:

## Associating Ghost with a COS Bucket

### Note:

We recommend you use a sub-account key and follow the [principle of least privilege](#) to reduce risks. For information about how to obtain a sub-account key, see [Access Key](#).

1. Add the following configuration to the `config.development.json` configuration file in the Ghost project's root directory:

```
"storage": {
  "active": "ghost-cos-store",
  "ghost-cos-store": {
    "BasePath": "ghost/", // You can change it to your directory name. If you leave
    "SecretId": "AKID*****",
    "SecretKey": "*****",
    "Bucket": "xxx-125*****",
    "Region": "**-*****"
  }
}
```

The parameters are described as follows:

Configuration Item	Value
BasePath	The COS path where files are stored. You can modify it as needed. If you leave it empty, the root directory will be used by default.
SecretId	Enter the access key information, which can be created and obtained on the <a href="#">Manage API Key</a> page.
SecretKey	Your access key information, which can be created and obtained on the <a href="#">Manage API Key</a> page.
Bucket	The name customized during bucket creation such as `examplebucket-1250000000`.
Region	The region selected during bucket creation

2. Create a custom storage directory and run the following command in the project's root directory:

```
mkdir -p content/adapters/storage
```

3. Install the [ghost-cos-store](#) plugin provide by Tencent Cloud.

3.1 Install it through npm.

```
npm install ghost-cos-store
```

3.2 Create the `ghost-cos-store.js` file with the following content in the `storage` directory:

```
// content/adapters/storage/ghost-cos-store.js
module.exports = require('ghost-cos-store');
```

3.3 Run `git clone` for installation.

```
cd content/adapters/storage
git clone https://github.com/tencentyun/ghost-cos-store.git
cd ghost-cos-store
npm i
```

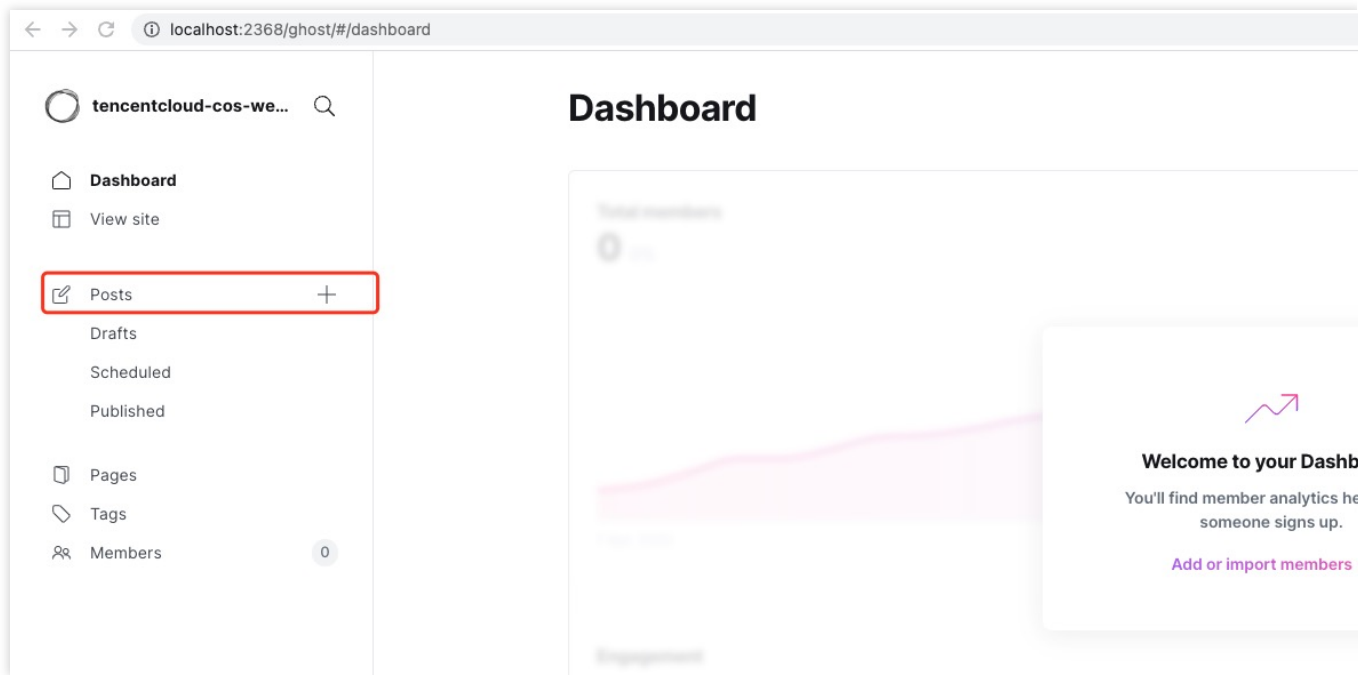
3.4 After the installation, restart Ghost.

```
ghost restart
```

## Publishing a Post and Testing Upload

1. In the Ghost console, click **+** to publish a post.





2. Click + to upload an image. From the packets captured by the browser, you can see that the `upload` request succeeds and the COS URL of the image is returned.

# Backing up Files from PC to COS

Last updated : 2024-03-25 15:16:26

## Background

Data matters, we all know that. Digital photos, e-documents, work products, saved game...none of them we can easily afford to lose. It will be a huge headache if we lose all of our files due to a disk failure, or a single file due to misoperation, computer shutdown, or software crash, or if we cannot provide a requested "callback version" just because we haven't saved one. This is why backups are absolutely important.

When it comes to backups, the first idea that occurs to our mind is most probably using a portable hard drive or building a NAS within an individual network so that we can just move files into it. Well, is it really as simple as all that? In fact, backup involves a lot of work to do. Copy our files onto backup media, check if the backups are correct, and we may have to do both of them regularly in order to minimize lost files. Besides, backup media require maintenance, so we need to replace our hard drives promptly once they are dead.

Given all this, is there an easier way to keep our files safe? The answer is Yes.

As Tencent Cloud's business grows, it has already developed a suite of enterprise cloud storage services, including COS. Now, we need backup software to connect the files in our PC with cloud storage services. It will help us back up files automatically onto the cloud, and check backup correctness on a regular basis.

## Software Introduction

[Arq® Backup](#) is commercial backup software for Windows and macOS. Running in the background, it automatically backs up specified directories at intervals you configure. Besides, it retains backup files for each point in time so that you can easily get an old version. This software offers minimal backup size and maximum backup speed by backing up only files that are different from those at the last point in time, and by backing up repeated files across paths only once. It encrypts backups with a password only you know before they leave your computer. Therefore, you can be assured that your sensitive data is well-protected from being stolen during the transfer over Internet or on-cloud storage.

To get a commercial license of Arq® Backup, each user needs to pay \$49.99. This software, which is used on a single computer, offers a 30-day free trial which you may want to try before purchase.

### Note:

Arq® Backup currently does not support the simplified Chinese language. You can download, purchase it and read its instructions from the [Official Website](#).

## Preparing Tencent Cloud COS

**Note:**

Please skip Steps 1-2 if you are using COS.

1. [Sign up for Tencent Cloud](#), and complete [Identity Verification](#).
2. Log in to the [COS Console](#), and activate COS service as instructed.
3. In the COS console, click **Bucket List** in the left sidebar first, and then **Create Bucket** to add a new bucket.

Name: bucket name, e.g. "backups".

Region: you can choose a region closest to your location. Currently, we offer price discounts for regions in southwest China, so you may alternatively choose "Chengdu" or "Chongqing" to enjoy this offer.

### Create Bucket

Name  -12500000000 ⓘ ✓  
Only support lowercase letters, numbers and "-". Up to 50 characters.

Region    
Services within the same region can be accessed through private network

Access Permissions  Private Read/Write  Public Read/Private Write  Public Read/Write  
Identity verification is required before accessing objects.

Endpoint   
Request endpoint

Bucket Tag   +

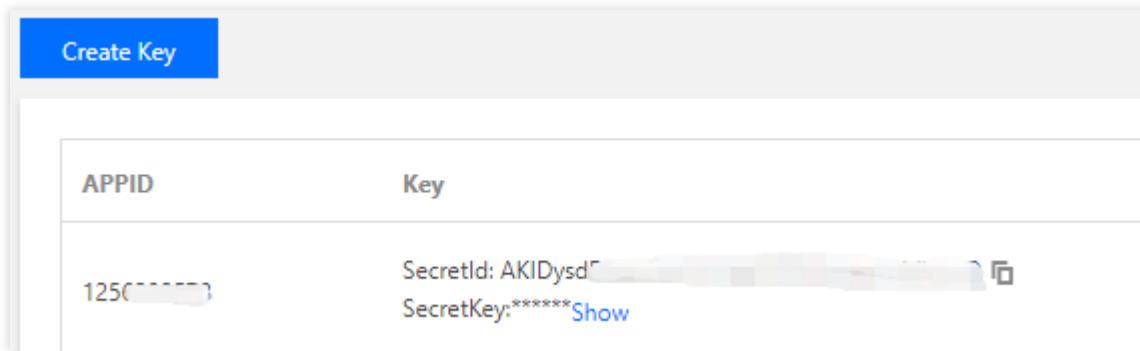
Server-Side Encryption  None  SSE-COS

For the other fields, leave the default. Copy and save the **Request endpoint**, and click **OK**.

**Note:**

Now, you have created a bucket. For more information, see [Creating a Bucket](#).

4. Log in to the [API Key Management Console](#), and create and save your SecretId and SecretKey.



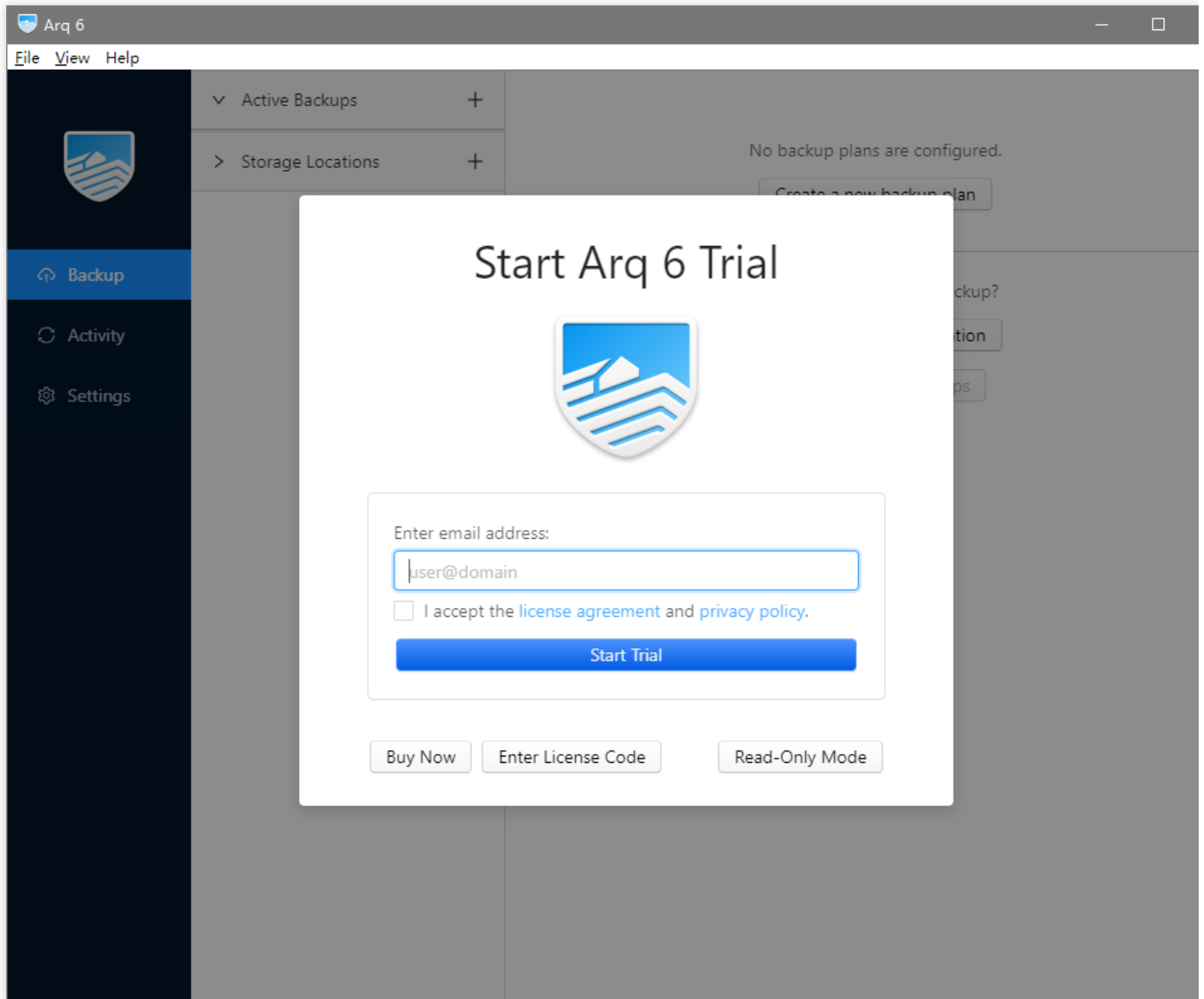
APPID	Key
1250000000	SecretId: AKIDysdF... SecretKey:*****Show

## Installing and Configuring Arq® Backup

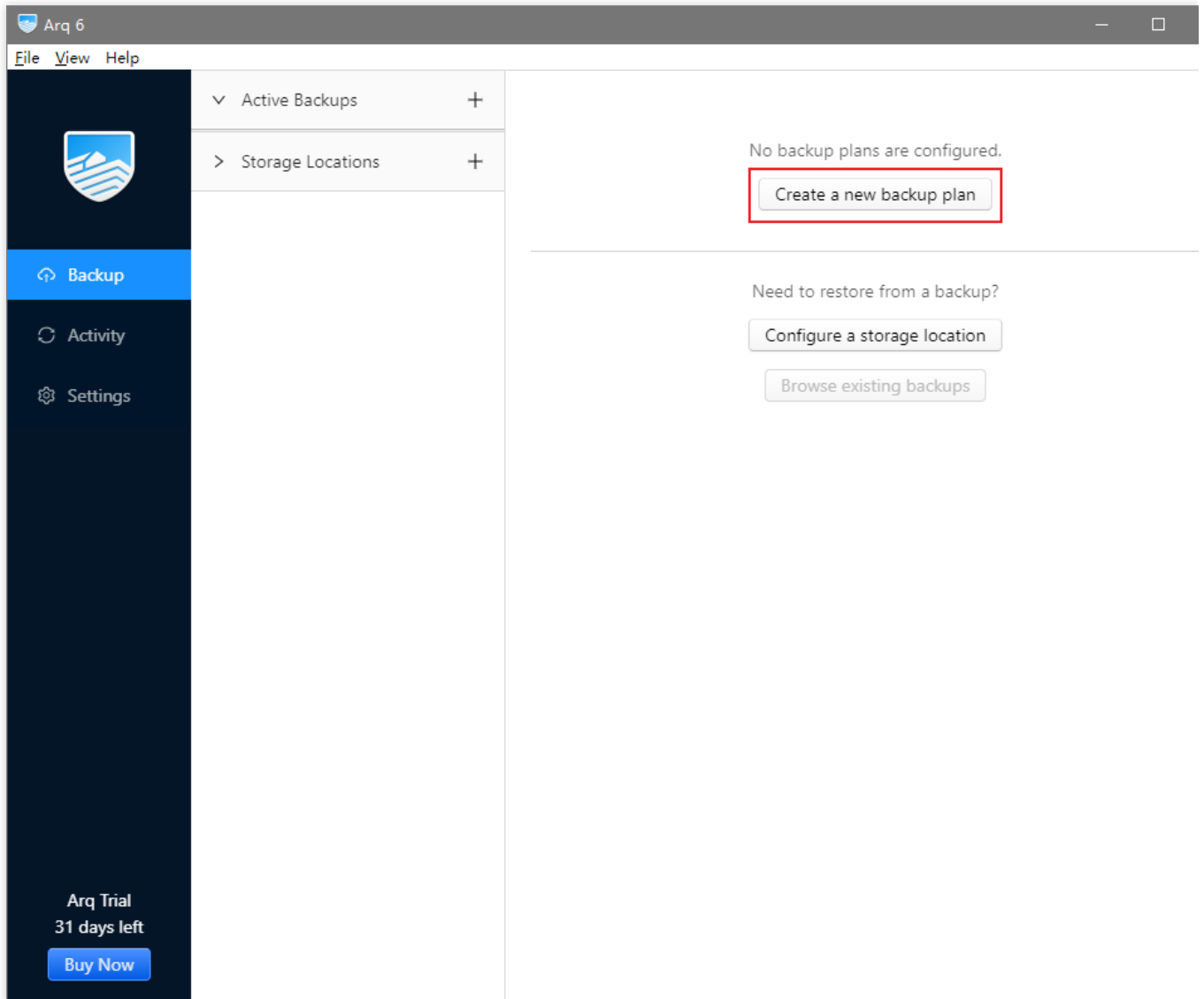
### Note:

Take Arq® Backup Version 6.2.11 for Windows as an example.

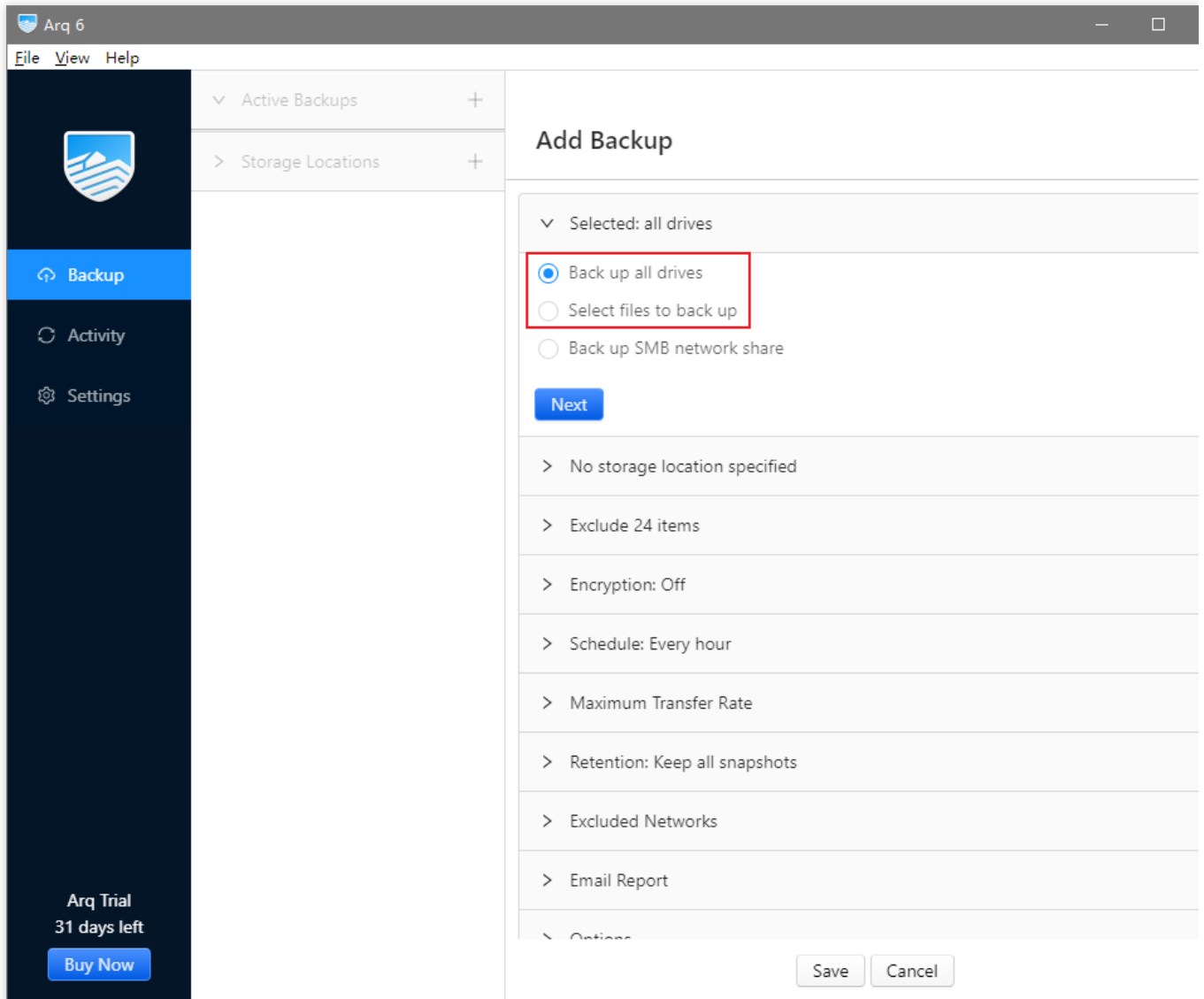
1. Download it from [Arq® Backup Website](#).
2. Follow the wizard to install the software. Once completed, it will start automatically while prompting you to log in. Then, enter your email address and click **Start Trial**.



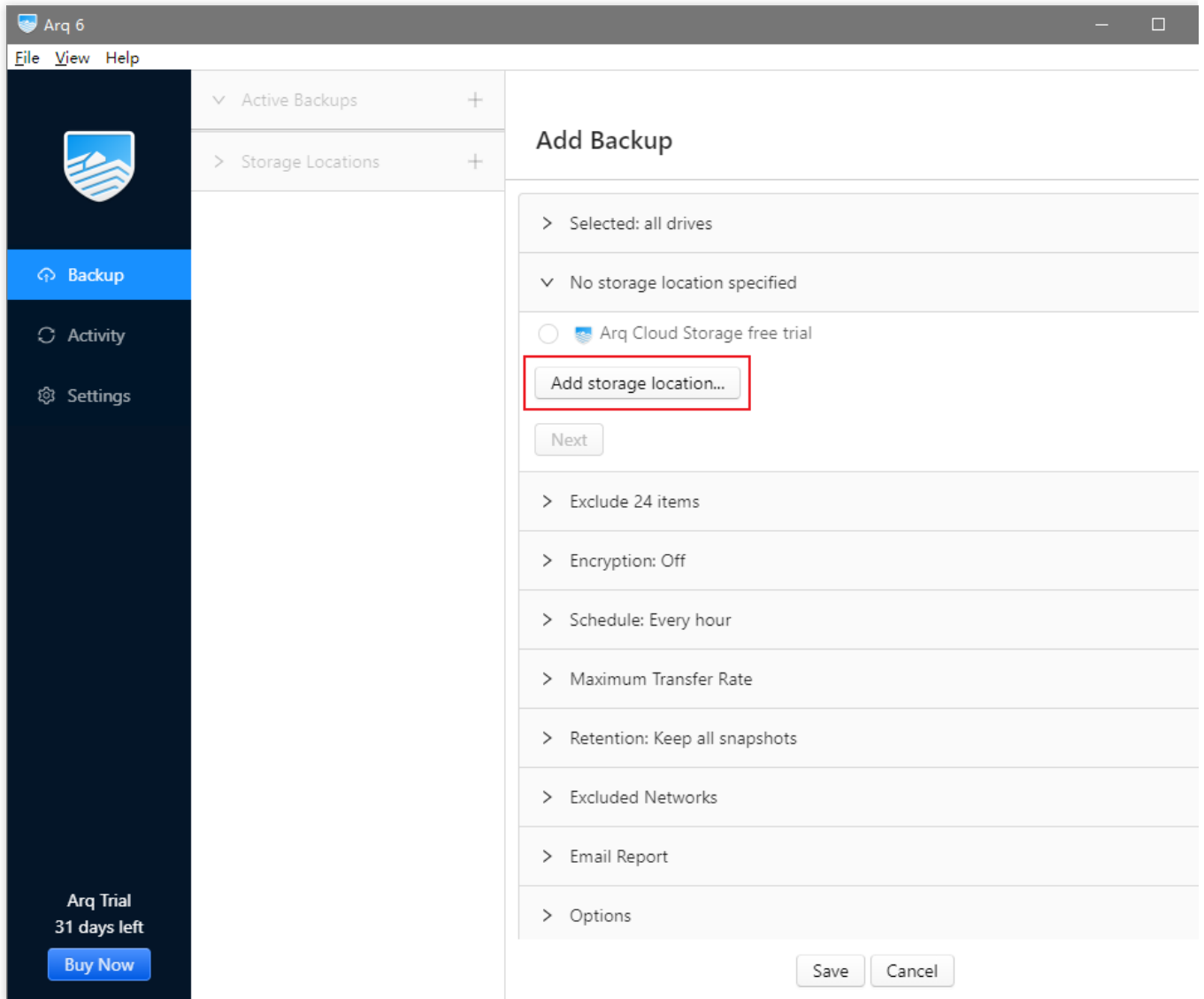
3. In the **Backup** pane, click **Create a new backup plan** to add a backup plan.



4. In the opened window, select *Back up all drives\** or **Select files to back up**.

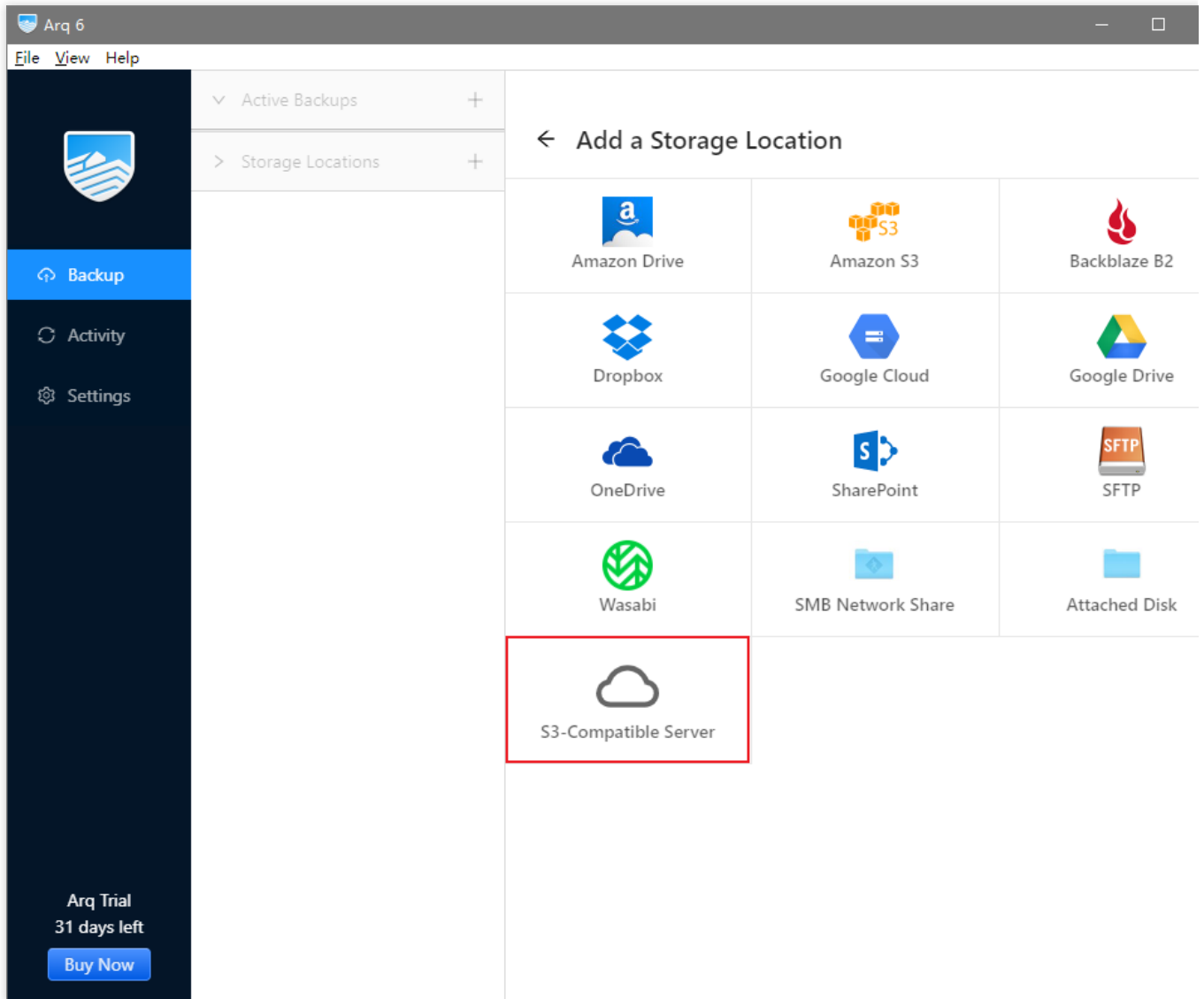


5. Click **Add storage location** to add a location for storing your backups as shown below:



6. In this example, we select **S3-Compatible Server**.





7. In the opened window, configure the following as instructed, and click **Continue**.

Server URL: enter the above-mentioned request endpoint, starting from `cos` and prepending `https://` to the beginning of it, such as `https://cos.ap-chengdu.myqcloud.com`. Note that the bucket name is excluded here.

Access Key ID: the above-mentioned SecretId.

Secret Access Key: the above-mentioned SecretKey.

## ← Add S3-Compatible Storage Location

Server URL:

Access Key ID:

Secret Access Key:

[Continue](#)

8. In the new window, click *Use an existing bucket*, select the above bucket you created, such as `backups-1250000000`, and click **Save**.

## ← S3-Compatible Bucket

Use an existing bucket

Create a bucket

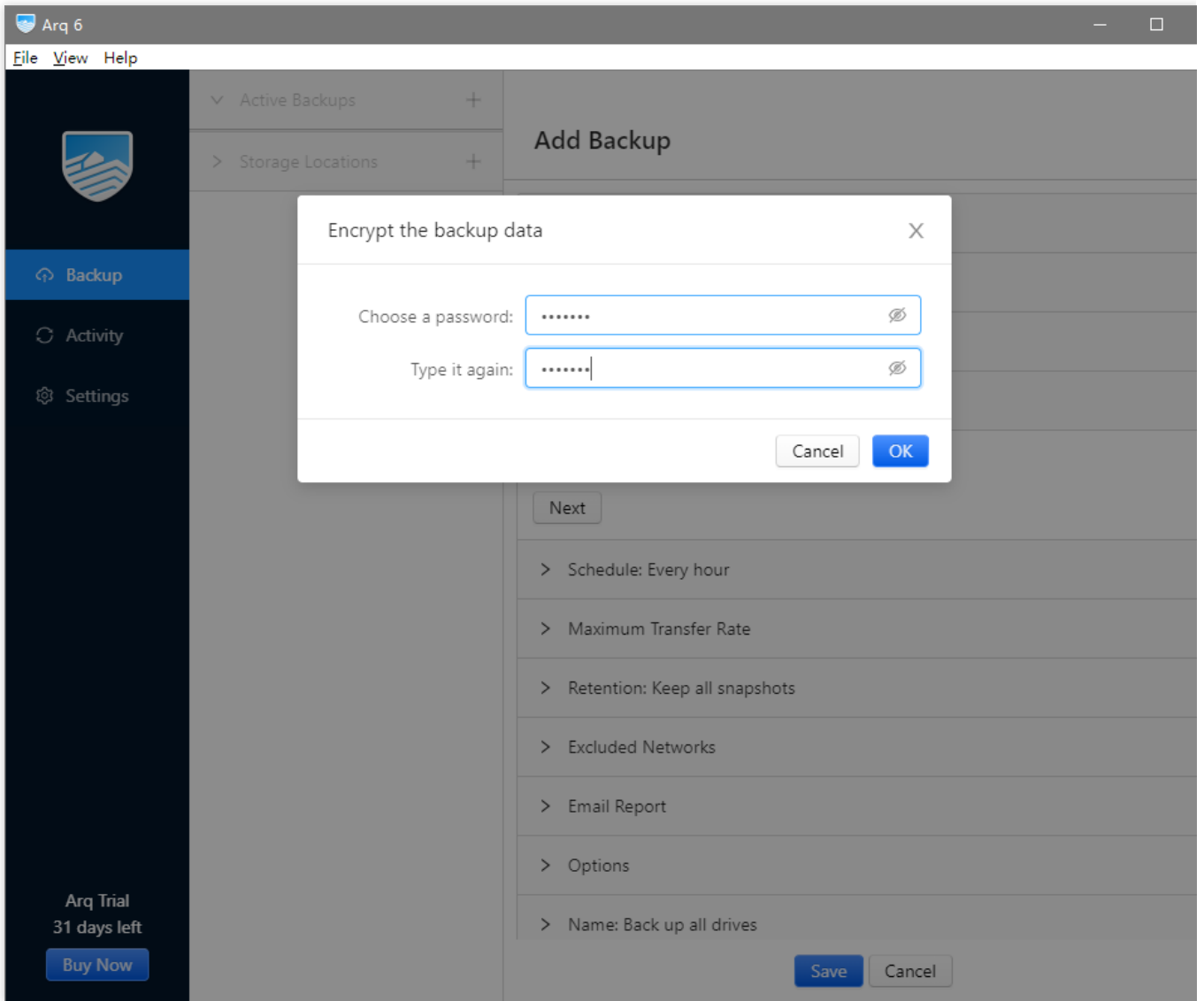
[Save](#)

9. (Optional) You may choose to encrypt backup data. Here, we select **On**.

## Add Backup

- > Selected: all drives
- > Storage location: backups-125
- > Exclude 24 items
- ▼ Encryption: Off
- Encrypt backup data:
- Next
- > Schedule: Every hour
- > Maximum Transfer Rate
- > Retention: Keep all snapshots
- > Excluded Networks
- > Email Report
- > Options
- > Name: Back up all drives

10. In the pop-up window, set your encryption password. Enter it twice and click **OK**. **Please keep your password in mind, otherwise, you may not be able to restore your files from backup.**



11. (Optional) You may configure a backup schedule.

## Add Backup

> Selected: all drives

> Storage location: backups-125. [redacted]

> Exclude 24 items

> Encryption: On

▼ Schedule: Every hour

Hourly  
Every  hour at  minutes after the hour

Mon  Tue  Wed  Thu  Fri  Sat  Sun

Daily

Weekly

Not scheduled

Last backup ended at: --

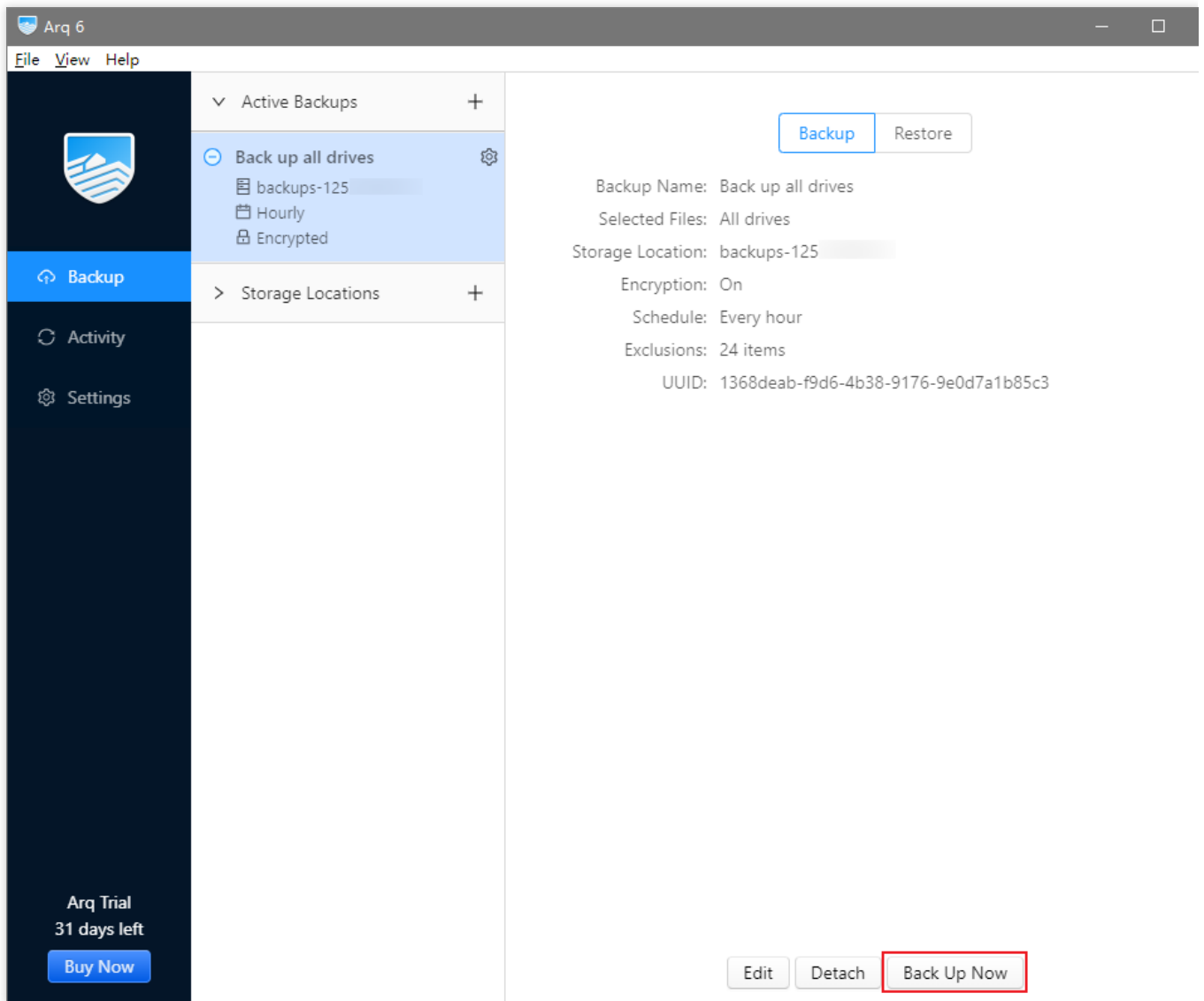
Next backup starts at: 2020/4/27 [redacted] 5:00:00

Start backup when a volume is connected

Next

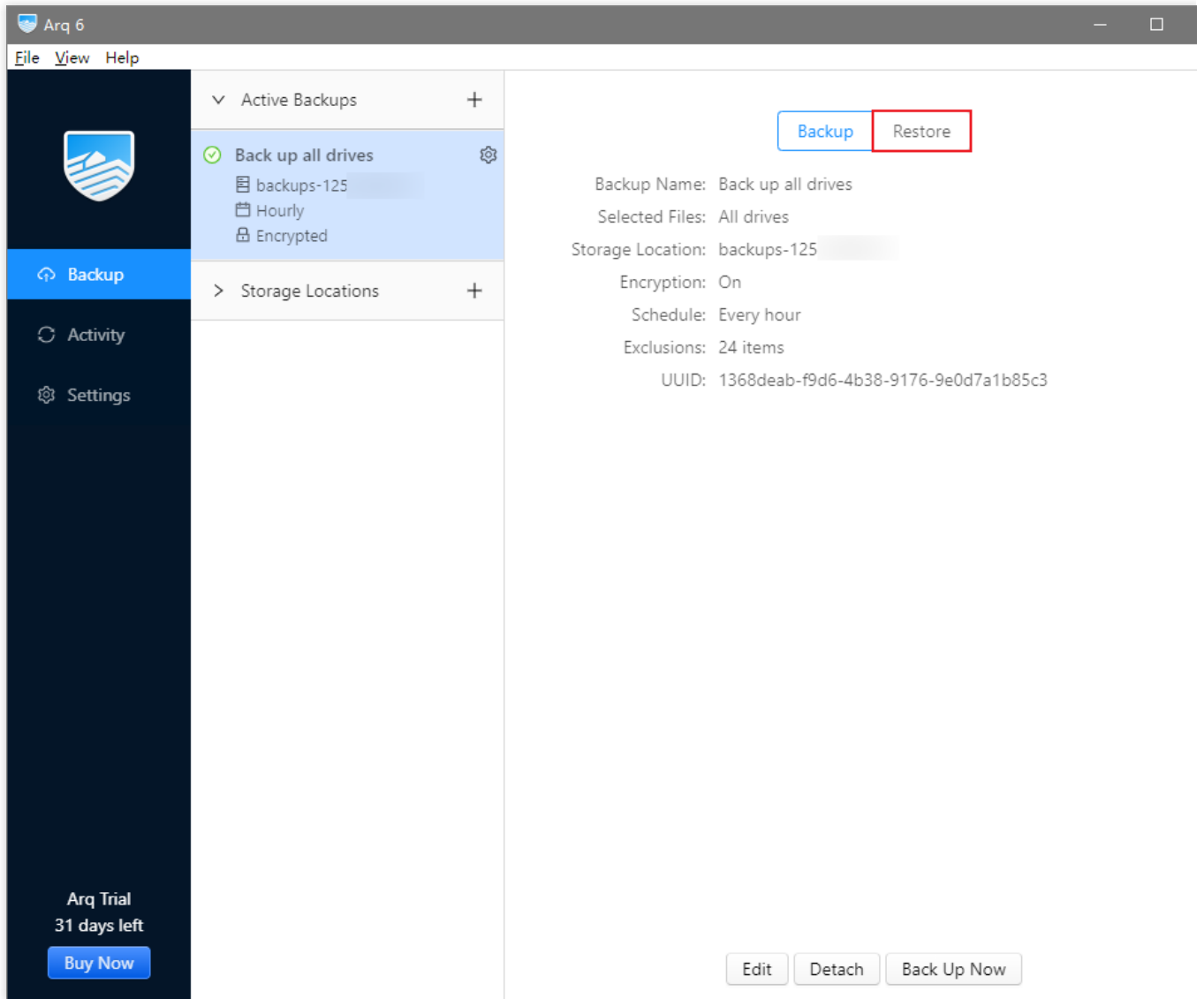
Save Cancel

12. Click **Save**, and then **Back Up Now** to start the backup.

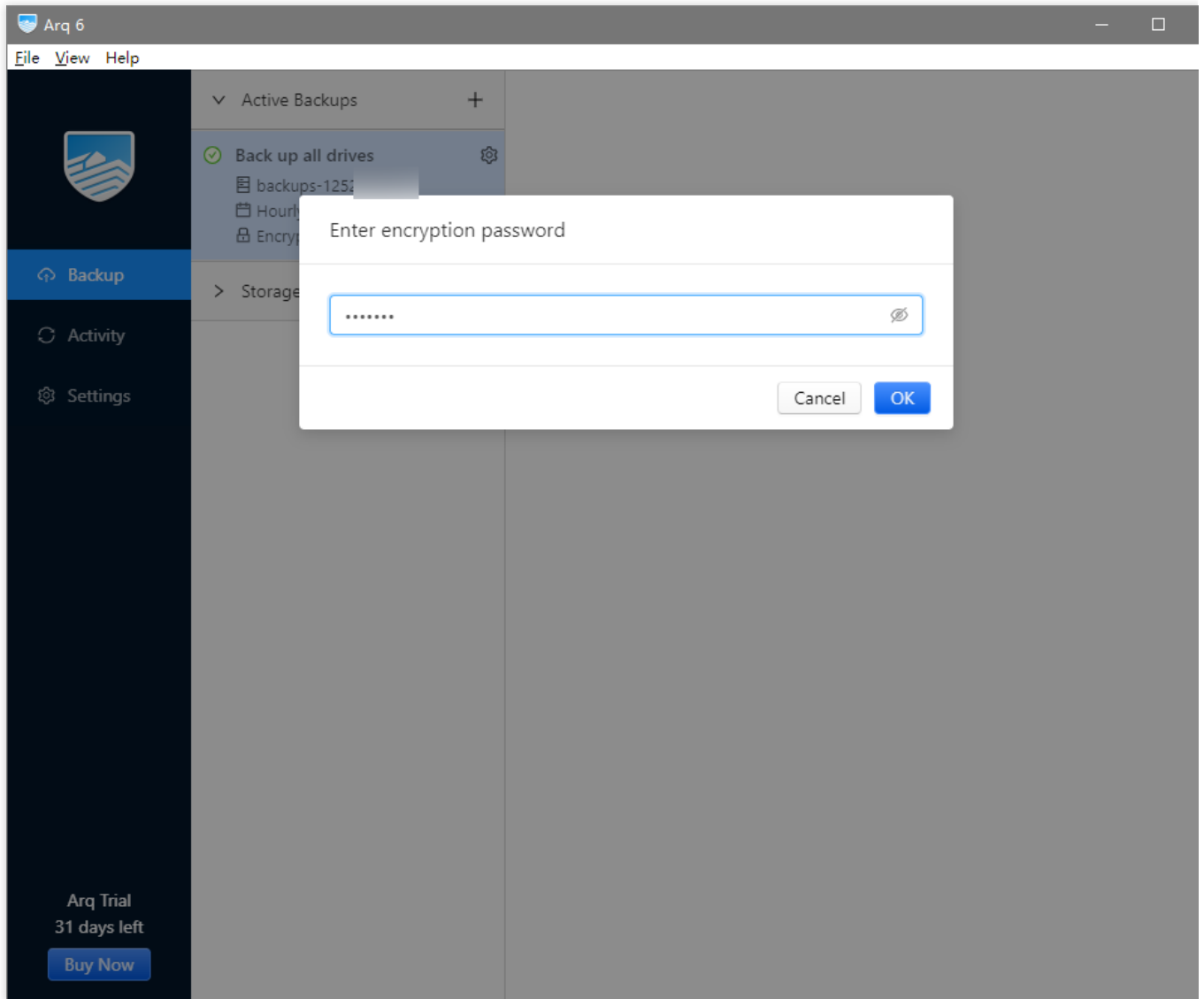


## Restoring Files from Backup

1. Click **Restore** in the list of **Backup** in the left sidebar.

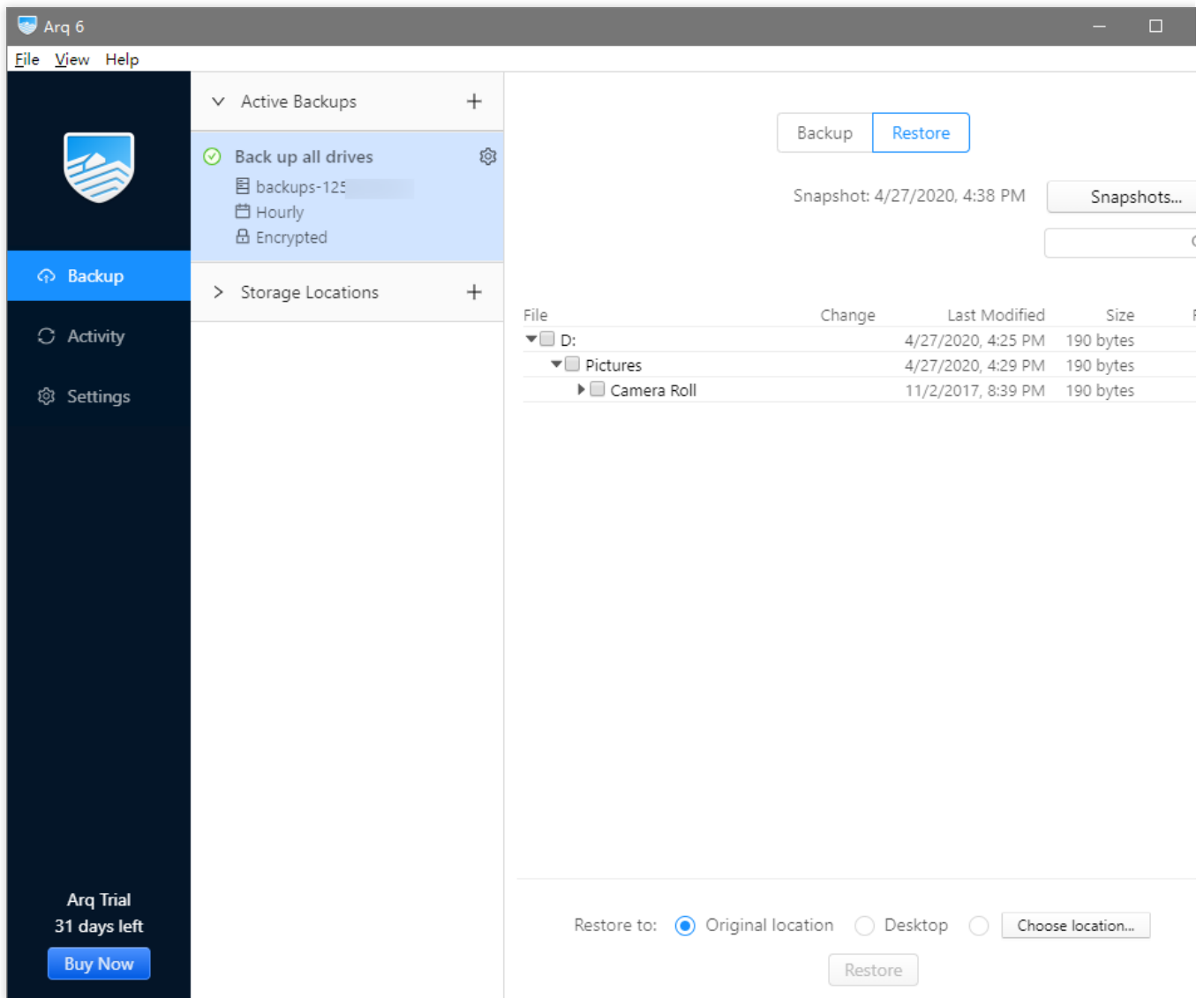


2. Enter your password if you have enabled **Encrypt backup data** in Step 9.

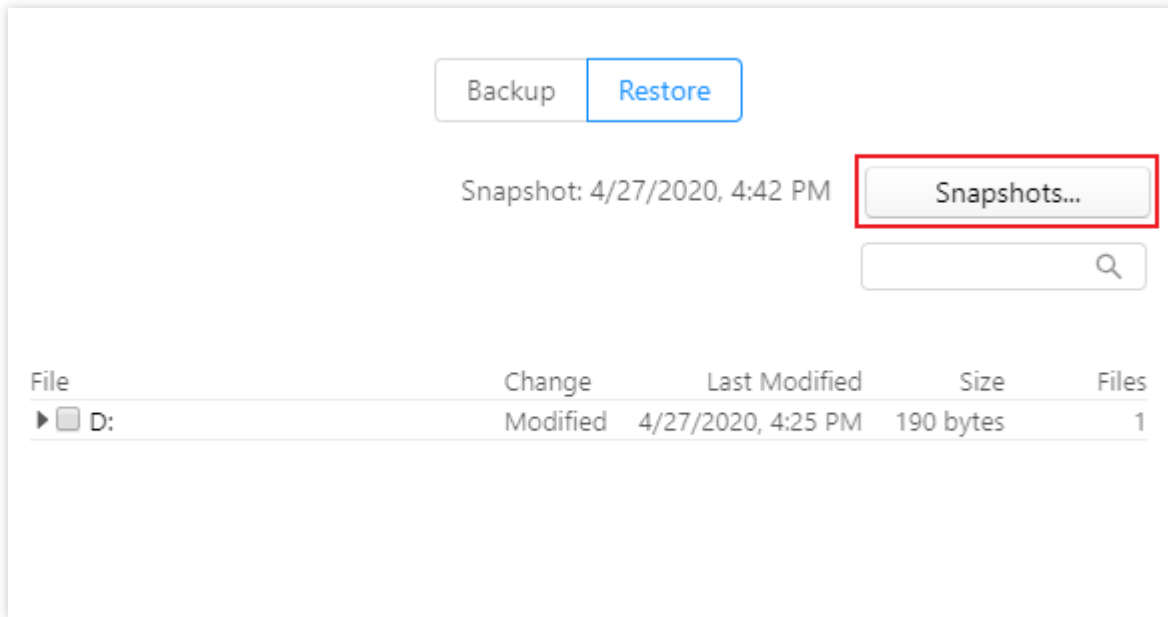


3. Select the directories or files to restore, and the location to save them, and then click **Restore**.

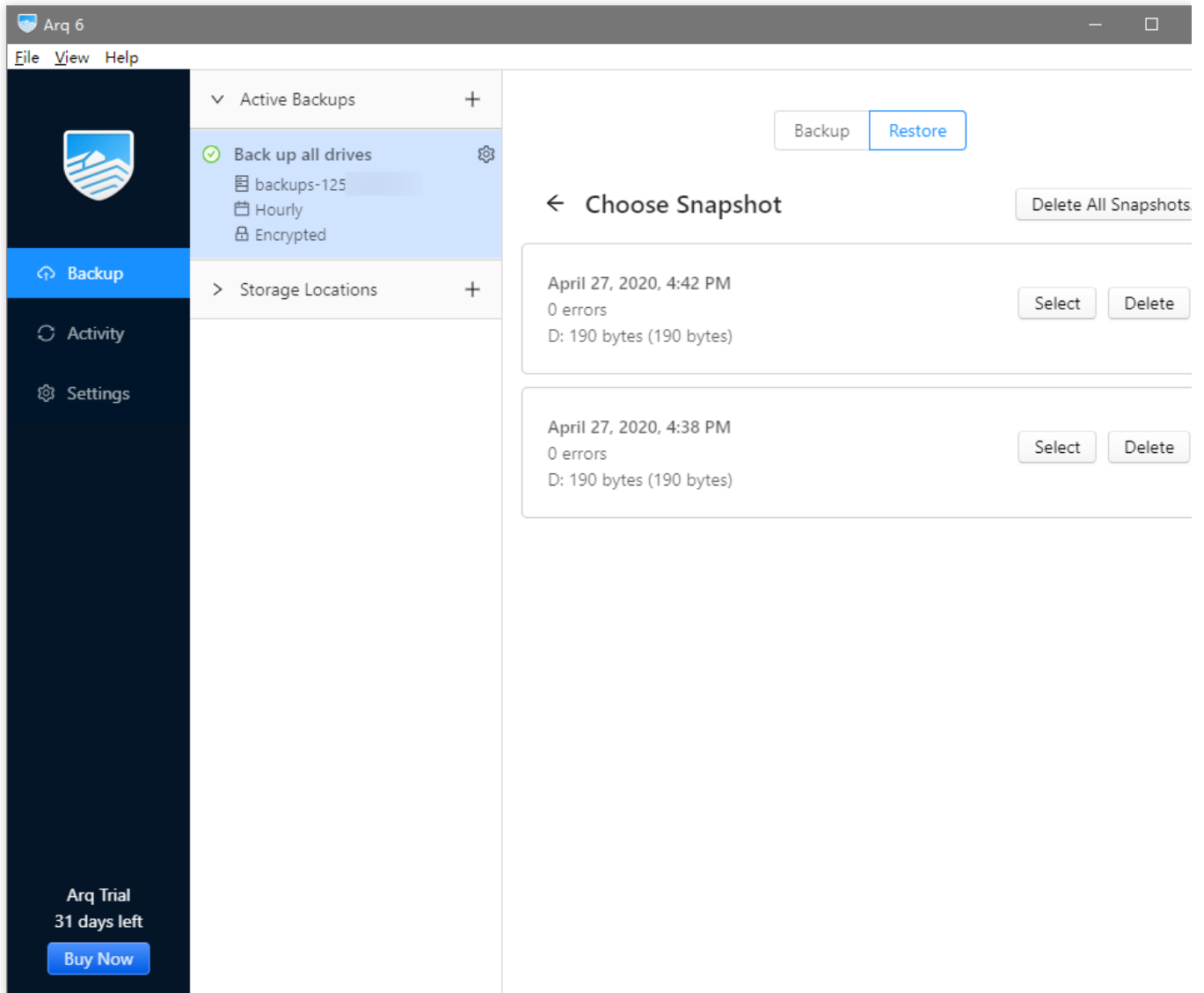




4. By default, files are restored from their latest backup. If necessary, you can choose to restore from an old version of backup in snapshots, which can be viewed by clicking **Snapshots**.



5. Choose a snapshot.



6. Select the snapshot directories or files to restore, and the location to save them, and then click **Restore**.

7. Once prompted that the restore operation is completed, you can go to the specified directories and view your restored files.

# Using Nextcloud and COS to Build Personal Online File Storage Service

Last updated : 2024-11-20 15:38:51

## Overview

Nextcloud is an open-source client and server software application that helps you create a personal online file storage service on your own.

The Nextcloud server is written in PHP and uses the local disk on the server for underlying storage. You can modify the Nextcloud configuration to use COS as the underlying storage, so as to enjoy a higher reliability, more powerful disaster recovery capabilities, and unlimited storage space at lower storage costs.

This document describes the environment on which the Nextcloud server depends, compares and analyzes the differences between local storage and COS, and explains how to build a personal online file storage service.

### Note:

Switching an existing Nextcloud server instance from local storage to COS may render existing files invisible. To change the storage method of an existing instance, we recommend you build a new Nextcloud server, configure COS as the storage, and migrate the data from the old instance to the new one.

## Nextcloud Server Environment Overview

The Nextcloud server is written in PHP and can use SQLite, MySQL, MariaDB, or PostgreSQL as the database. Due to the limits in performance, we generally recommend you not use SQLite in your actual business. Although PHP, MySQL, and relevant server software applications support Windows, according to the Nextcloud community, running the Nextcloud server on Windows may cause problems such as garbled text. Therefore, the Nextcloud team has declared that the Nextcloud server cannot be deployed on Windows.

### Server configuration

Cloud Virtual Machine (CVM) currently provides multiple instance families, with multiple sub-families each. Different instance families have different strengths, such as large memory or high I/O. As Nextcloud is targeted at individual, family, and SME users, it has low requirements for hardware resources, and you can select a Standard model with balanced resources. As for the CVM instance sub-families, the latest ones generally have a higher cost performance, so just select the latest sub-family.

After determining the instance family and sub-family, you need to choose specific vCPU and memory specifications (the private network bandwidth and PPS vary by specification). You can use OPcache for PHP to improve the

performance, and the Nextcloud server can use APCu memory cache to further improve the performance, so we recommend you select a large memory.

As you can adjust the CVM instance configuration after purchase, you can first purchase an instance with low specifications such as 1-core and 4 GB MEM and determine whether to upgrade the specifications to improve the performance based on the numbers of users and files as well as CVM monitoring data after your online file storage service is built and launched to the production environment. If you need to use the service in a multi-user scenario such as family or SME, we recommend you purchase a 2-core 8 GB MEM or 4-core 16 GB MEM instance to offer a sufficient performance.

## Server operating system

All mainstream Linux distributions can well sustain the Nextcloud server. Their configurations are basically the same except for the command (i.e., package management tool) used for software package installation.

### Note:

This document takes a CVM instance on CentOS 7.7 as an example.

## Databases

As mentioned above, MySQL is generally used together with PHP in the actual business. As MariaDB is a fork from MySQL and highly compatible with MySQL, the Nextcloud server can run well with MySQL 5.7+ or MariaDB 10.2+. Tencent Cloud offers TencentDB for MySQL and TencentDB for MariaDB. Both services adopt a source-replica high-availability architecture and have a higher reliability compared with self-built databases in CVM. In addition, they support easy-to-use Ops features like automatic backup. Therefore, we strongly recommend you use TencentDB in your actual business.

### Note:

This document takes a TencentDB for MySQL 5.7 instance as an example.

## Web server and PHP runtime

Some Nextcloud server configurations are specified in `.htaccess` files, so you can directly use Nextcloud's built-in configuration items when using the Apache server software application. Nginx is a web server software application developing rapidly in recent years and features ease of installation and configuration, less resource usage, and higher load capacity compared with Apache. You can transcribe `.htaccess` configurations on the Nextcloud server to Nginx configurations to better sustain the Nextcloud server. This document uses the Nginx server software application and provides a complete Nginx configuration example for reference.

The PHP runtime has been upgraded to PHP 7, and the main maintained versions include 7.2, 7.3, and 7.4, all of which support the Nextcloud server. Here, use the latest version 7.4. Moreover, Nextcloud depends on certain PHP modules. The requirements for specific modules are as detailed below.

## Tencent Cloud network environment

Currently, Tencent Cloud offers the classic network and VPC environments. The classic network is a public network resource pool shared by all Tencent Cloud users, where the private IPs of all CVM instances are assigned by Tencent Cloud and you cannot customize IP ranges or IP addresses. A VPC is a logically isolated network space in Tencent Cloud. In a VPC, you can customize IP ranges, IP addresses, and routing policies. Currently, as the classic network resources are insufficient and cannot be expanded, the classic network is no longer supported for new accounts and in some new AZs. Therefore, this document takes a VPC as an example.

**Note:**

For more information on VPC, see [Overview](#).

## Comparison Between CBS and COS

Cloud Block Storage (CBS) disks are mounted to the operating system as local disks on CVM instances. As Nextcloud uses the file system to store the online file storage data by default, you can directly store Nextcloud data to CBS disks on the operating system. Compared with CBS, COS offers the following benefits:

### Use cases

#### CBS

CBS is a type of block storage and can be directly mounted to the CVM operating system as disks. Generally, a CBS disk is used exclusively by the operating system and can be mounted to only one CVM instance. However, it has a high read/write performance and is suitable for scenarios where a high I/O and a low latency are required and it is used by only one CVM instance exclusively.

#### COS

COS opens up its read/write APIs over the HTTP protocol, so you need to access the objects (files) stored in COS through programming. It uses object keys (like file paths) as indexes and have an unlimited storage capacity. As its data is transferred over the network, COS has a lower speed and higher latency. However, as operations are performed on objects, after an application manipulates an object, another application can immediately manipulate the same object. In conclusion, COS is suitable for scenarios where a high performance is not required but a large cost-effective storage capacity or shared access is needed. It is more suitable to use the online file storage application together with COS for the following reasons: the online file storage application transfers data over the network and doesn't require a low latency; the speed and latency on the linkage from the online file storage client to server and then to COS are mainly subject to the client network; COS doesn't limit its own speed.

### Maintenance

#### CBS

CBS has a fixed capacity, which can be expanded in the console or via TencentCloud API. After expansion, you need to add partitions on the operating system, and partition exceptions may occur, which incur certain maintenance costs.

## COS

COS is used on demand and doesn't limit the total capacity or the number of objects (files), so it requires no maintenance at all.

## Data security

Both CBS and COS use methods such as multi-copy to guarantee the data reliability.

# Building the Nextcloud Server Runtime Environment

## Preparing Tencent Cloud products on which the Nextcloud server depends

### CVM

To get started with CVM, see [Custom Configurations](#).

### TencentDB for MySQL

To get started with TencentDB for MySQL, see [Creating MySQL Instance](#).

### COS

1. Log in to the [COS console](#) (you need to activate COS first if you use it for the first time), go to the **Bucket List** page, click **Create Bucket**, and configure as follows:

Configuration Item	Value
Name	Enter a custom bucket name such as `nextcloud`. Note that the name cannot be changed once confirmed.
Region	Select the region of the CVM instance.
Other	Keep the default settings.

2. After setting the above configuration items, click **OK**.

## Installing and configuring the web server and PHP runtime

### Installing Nginx

1. Use an SSH tool to log in to the newly purchased server.
2. Run the following command to install Nginx:

```
yum install nginx
```

If the following information is displayed, press `Y` and `Enter` to confirm the installation (repeat the same operation for similar information).

```
Is this ok [y/d/N]:
```

3. If the following information is displayed, the installation is completed:

```
Complete!  
[root@VM-0-10-centos ~]#
```

4. Run the following command to check whether the Nginx version can be viewed normally:

```
nginx -v
```

If the following information is displayed, the installation is completed:

```
nginx version: nginx/1.16.1
```

## Installing PHP

1. Use an SSH tool to log in to the newly purchased server.
2. Run the following command to install PHP 7.4:

```
yum install epel-release yum-utils
```

3. Run the following commands in sequence:

### Command 1:

```
yum install http://rpms.remirepo.net/enterprise/remi-release-7.rpm
```

### Note:

If command execution is too slow or stuck for a long time, you can press `Ctrl + C` to cancel the execution and run the command again (same for the following commands).

### Command 2:

```
yum-config-manager --enable remi-php74
```

### Command 3:

```
yum install php php-fpm
```

4. After the installation, run the following command to check whether the PHP version can be viewed normally:

```
php -v
```



If the following information is displayed, the installation is completed:

```
PHP 7.4.8 (cli) (built: Jul 9 2020 08:57:23) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

## Installing PHP modules

In addition to the basic part of PHP, Nextcloud also depends on other PHP modules to implement some features. For more information on the modules on which Nextcloud depends, see [Prerequisites for manual installation](#).

In this example, the PHP modules required by Nextcloud are installed. If you want to use other optional features of Nextcloud, find and install the PHP modules depended on by yourself.

1. Use an SSH tool to log in to the newly purchased server.
2. Run the following command to install the PHP modules:

```
yum install php-xml php-gd php-mbstring php-mysqlnd php-intl php-zip
```

3. After the installation, run the following command to view the installed PHP modules:

```
php -m
```

4. To install another module, simply run `yum install <php-module-name>` again.

## Uploading and decompressing the Nextcloud server code

1. Download the latest installation package of the Nextcloud server from the [Nextcloud website](#) and upload it to the `/var/www/` directory on the server as follows:

1. Run the `wget` command to directly download the installation package from the server. For example, after entering the `/var/www/` directory, run the following command: `wget`

```
https://download.nextcloud.com/server/releases/nextcloud-19.0.1.zip .
```

2. Download the installation package to the local PC and upload it to the `/var/www/` directory through SFTP or SCP.

3. Download the installation package to the local PC and upload it through lrzsz as follows:

- 3.1 Use an SSH tool to log in to the newly purchased server.

- 3.2 Run `yum install lrzsz` to install lrzsz.

- 3.3 Run `cd /var/www/` to enter the target directory.

- 3.4 Run `rz -bye` and select the Nextcloud server installation package downloaded to the local PC in the SSH tool (the operation varies by tool).

4. Use an SSH tool to log in to the newly purchased server.

5. Run `unzip nextcloud-<version>.zip` to decompress the installation package such as `unzip nextcloud-19.0.1.zip .`

## Configuring PHP

1. Use an SSH tool to log in to the newly purchased server.
2. Run `vim /etc/php-fpm.d/www.conf` to open the PHP-FPM configuration file and modify the following configuration items (for detailed directions on how to use Vim, see its documentation; you can also modify the configuration file in other ways):
  1. Change `user = apache` to `user = nginx`.
  2. Change `group = apache` to `group = nginx`.
3. After the modification, enter `:wq` to save the file and exit Vim (for detailed directions on how to use Vim, see its documentation).
4. Run the following command to change the directory owner to make PHP compatible with Nginx:

```
chown -R nginx:nginx /var/lib/php
```

5. Run the following commands in sequence and start the PHP-FPM service:

### Command 1:

```
systemctl enable php-fpm # Command 1
```

### Command 2:

```
systemctl start php-fpm # Command 2
```

## Configuring Nginx

1. Use an SSH tool to log in to the newly purchased server.
2. Run the following command to change the website directory owner:

```
chown -R nginx:nginx /var/www
```

3. Back up the current Nginx configuration file `/etc/nginx/nginx.conf` as follows:

1. Run `cp /etc/nginx/nginx.conf ~/nginx.conf.bak` to back up the current configuration file to the `HOME` directory.
2. Use SFTP or SCP to download the current configuration file to the local PC.
3. Change `/etc/nginx/nginx.conf` to the following content:

```
# For more information on configuration, see:
# * Official English Documentation: http://nginx.org/en/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;
```

```
# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/nginx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    server {
        listen 80 default_server;
        listen [::]:80 default_server;
        server_name _;
        root /var/www/nextcloud;

        add_header Referrer-Policy "no-referrer" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header X-Download-Options "noopen" always;
        add_header X-Frame-Options "SAMEORIGIN" always;
        add_header X-Permitted-Cross-Domain-Policies "none" always;
        add_header X-Robots-Tag "none" always;
        add_header X-XSS-Protection "1; mode=block" always;

        client_max_body_size 512M;
        fastcgi_buffers 64 4K;

        gzip on;
        gzip_vary on;
```

```
gzip_comp_level 4;
gzip_min_length 256;
gzip_proxied expired no-cache no-store private no_last_modified no_etag
auth;
    gzip_types application/atom+xml application/javascript application/json
application/ld+json application/manifest+json application/rss+xml
application/vnd.geo+json application/vnd.ms-fontobject application/x-font-ttf
application/x-web-app-manifest+json application/xhtml+xml application/xml
font/opentype image/bmp image/svg+xml image/x-icon text/cache-manifest text/css
text/plain text/vcard text/vnd.rim.location.xloc text/vtt text/x-component
text/x-cross-domain-policy;

# Load configuration files for the default server block.
include /etc/nginx/default.d/*.conf;

location / {
    try_files $uri $uri/ =404;
    index index.php;
}

location ~ ^\/(?:build|tests|config|lib|3rdparty|templates|data)\{/ {
    deny all;
}

location ~ ^\/(?:\.|autotest|occ|issue|indie|db_|console) {
    deny all;
}

location ~
^\/(?:index|remote|public|cron|core\/ajax\/update|status|ocs\/v[12]|updater
\/.+|oc[ms]-provider\/.+)\.php(?:$|\/) {
    fastcgi_split_path_info ^(.+?\.php)(\/.*|)$;
    set $path_info $fastcgi_path_info;
    try_files $fastcgi_script_name =404;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $path_info;
    fastcgi_param modHeadersAvailable true;
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_intercept_errors on;
    fastcgi_request_buffering off;
}

location ~ ^\/(?:updater|oc[ms]-provider)(?:$|\/) {
    try_files $uri/ =404;
    index index.php;
}
```

```
location ~ /\.(css|js|svg|gif)$ {
    add_header Cache-Control "max-age=15778463";
}

location ~ /\.woff2?$ {
    add_header Cache-Control "max-age=604800";
}

}

# Settings for a TLS enabled server.
#
# server {
#     listen      443 ssl http2 default_server;
#     listen      [::]:443 ssl http2 default_server;
#     server_name _;
#     root        /usr/share/nginx/html;
#
#     ssl_certificate "/etc/pki/nginx/server.crt";
#     ssl_certificate_key "/etc/pki/nginx/private/server.key";
#     ssl_session_cache shared:SSL:1m;
#     ssl_session_timeout 10m;
#     ssl_ciphers HIGH:!aNULL:!MD5;
#     ssl_prefer_server_ciphers on;
#
#     # Load configuration files for the default server block.
#     include /etc/nginx/default.d/*.conf;
#
#     location / {
#     }
#
#     error_page 404 /404.html;
#         location = /40x.html {
#     }
#
#     error_page 500 502 503 504 /50x.html;
#         location = /50x.html {
#     }
# }
}
```

4. Run the following commands in sequence and start the Nginx service:

**Command 1:**

```
systemctl enable nginx
```

**Command 2:**

```
systemctl start nginx
```


## Configuring the Nextcloud Server to Use COS

### Getting COS information

1. Log in to the [COS console](#).
2. Click the name of the newly created bucket.

<a href="#">examplebucket-1250000000</a>	Specified user	Chengdu (China) (ap-chengdu)	2019-03-20 15:29:59	<a href="#">Monitor</a> <a href="#">Configure</a> <a href="#">More</a>
--	----------------	------------------------------	---------------------	--

3. On the left sidebar, select **Overview** and record the information in **Bucket Name** and **Region** in **Basic Information**.

Basic Information	
Bucket Name	examplebucket-1250000000 
Region	Chengdu (China) (ap-chengdu)
Creation Time	2019-03-20 15:29:59
Access Permissions	Private Read/Write

### Getting the API key

We recommend you use a sub-account key and follow the [principle of least privilege](#) to reduce risks. For information about how to obtain a sub-account key, see [Access Key](#).

### Modifying the Nextcloud server configuration file

1. Use a text editor to create the `config.php` file, enter the following content, and modify relevant values according to the comments:

```
<?php
$CONFIG = array(
    'objectstore' => array(
```

```
'class' => '\\\\OC\\\\Files\\\\ObjectStore\\\\S3',
'arguments' => array(
    'bucket' => 'nextcloud-1250000000', // Bucket name
    'autocreate' => false,
    'key' => 'AKIDxxxxxxxx', // Replace it with your `SecretId`
    'secret' => 'xxxxxxxxxxxx', // Replace it with your `SecretKey`
    'hostname' => 'cos.<Region>.myqcloud.com', // Change `<Region>` to the bucket re
    'use_ssl' => true,
),
),
);
```

As shown below:

```
<?php
$CONFIG = array(
    'objectstore' => array(
        'class' => '\\OC\\Files\\ObjectStore\\S3',
        'arguments' => array(
            'bucket' => 'nextcloud-125 555', //
            'autocreate' => false,
            'key' => 'AKID 6NEu', // SecretId
            'secret' => 'nT2P TH0X', // SecretKey
            'hostname' => 'cos.ap-shanghai.myqcloud.com', //
            'use_ssl' => true,
        ),
    ),
);
```

2. Save the file and upload it to the `/var/www/nextcloud/config/` directory (keep the filename `config.php`) through SFTP or SCP or by running the `rz -bye` command.
3. Run the following command to change the owner of the configuration file:

```
chown nginx:nginx /var/www/nextcloud/config/config.php
```

## Configuring a Domain Name

If you want to use your own domain name but not an IP address to access your Nextcloud server, you can register a domain name, resolve it to your CVM IP address, and get the ICP filing for it as instructed in the documentation of your domain registrar.

As the Nextcloud server will record the domain name or IP address used during installation, we recommend you register, resolve, and get the ICP filing for a domain name before installation and use the domain name to go to the

security page of the Nextcloud server.

If you need to change the domain name or IP address after installing the Nextcloud server, you can modify

`trusted_domains` in the `/var/www/nextcloud/config/config.php` configuration file on your own as instructed in [Trusted domains](#).

## Installing the Nextcloud Server

1. Access the Nextcloud server in the browser. Create an admin account and record the admin username and password.
2. Expand **Storage & database** and configure as follows:

Configuration Item	Value
Data folder	<code>/var/www/nextcloud/data</code> (keep the default value)
Configure the database	MySQL/MariaDB
Database user	root
Database password	Password of the `root` user entered during TencentDB for MySQL initialization.
Database name	nextcloud (or another unused name)
Database host (`localhost` is displayed by default)	Private network address of the TencentDB for MySQL instance

3. Click **Finish setup** and wait for the Nextcloud server to be installed.
4. If an error such as `504 Gateway Timeout` is displayed during installation, you can directly refresh the page and try again.
5. After the installation, log in to the Nextcloud server with the admin account to use Nextcloud on web.

## Nextcloud Server Debugging

### Background job

The Nextcloud server sometimes needs to execute some background jobs like database cleanup when there are no user interactions. As the PHP operation characteristics prevent PHP-based programs from maintaining an independent worker process or thread internally, you need to proactively call the corresponding PHP program externally to execute such background jobs.



The Nextcloud server offers three background job call methods. By default, the browser will initiate an Ajax request to start a server background job to execute it if you log in on the web client. However, this method depends greatly on your login status. If there are no users logged in, such background jobs cannot be executed, so this method is the least reliable.

To solve the problem of low reliability of Ajax-based background job execution, we recommend you use cron on Linux to configure background jobs. cron can accurately control the time when a job is started such as every five minutes (the number of minutes must be an integral multiple of five) or the 10th minute at each o'clock. The customizable time granularities include minute, hour, day of month, month, and day of week and even second and year on certain operating systems. Therefore, this method is highly flexible. For more information on and configuration of cron, see cron documentation.

The following steps describe how to configure cron to support Nextcloud server background jobs:

1. Use an SSH tool to log in to the newly purchased server.
2. Run the following command to install the PHP modules:

```
yum install php-posix
```

3. Run the following command to open or create a cron configuration for the `nginx` account:

```
crontab -u nginx -e
```

4. The editor used here is Vi/Vim. Press `i` to enter the editing mode and insert the following line:

```
*/5 * * * * php -f /var/www/nextcloud/cron.php
```

Press `ESC` to exit the editing mode and enter `:wq` to save the file and exit the editor (for detailed directions on how to use Vi/Vim, see its documentation).

In the above configurations, the officially recommended execution frequency of once every five minutes is set. After the background jobs are executed in five minutes, you can open the browser, log in to the Nextcloud server, click the first letter of your username in the top-right corner to enter the **Settings** page, select **Administration > Basic Settings** on the left sidebar, and you can see that **Cron** is selected by default in **Background jobs**.

## Memory cache

PHP can use OPcache to improve the performance, and the Nextcloud server also can use the APCu memory cache to further improve the performance. You can configure as follows:

1. Use an SSH tool to log in to the newly purchased server.
2. Run the following command to install the PHP modules:

```
yum install php-pecl-apcu
```

3. Run the following commands in sequence to restart Nginx and PHP-FPM:

### Command 1:

```
systemctl restart nginx
```

**Command 2:**

```
systemctl restart php-fpm
```

4. Run `vim /var/www/nextcloud/config/config.php` to open the configuration file of the Nextcloud server, add the line `'memcache.local' => '\\OC\\Memcache\\APCu'`, in `$CONFIG = array (`, save the file, and exit the editor.

```
[root@VM-0-10-centos config]# vim /var/www/nextcloud/config/config.php
<?php
$CONFIG = array (
  'objectstore' =>
  array (
    'class' => '\\OC\\Files\\ObjectStore\\S3',
    'arguments' =>
    array (
      'bucket' => 'nextcloud-125[REDACTED]555',
      'autocreate' => false,
      'key' => 'AKID[REDACTED]6NEu',
      'secret' => 'nT2P[REDACTED]TH0X',
      'hostname' => 'cos.ap-shanghai.myqcloud.com',
      'use_ssl' => true,
    ),
  ),
  'instanceid' => '[REDACTED]',
  'passwordsalt' => '[REDACTED]',
  'secret' => '[REDACTED]',
  'trusted_domains' =>
  array (
    0 => '[REDACTED]',
  ),
  'datadirectory' => '/var/www/nextcloud/data',
  'dbtype' => 'mysql',
  'version' => '19.0.1.1',
  'overwrite.cli.url' => 'http://[REDACTED]',
  'dbname' => 'nextcloud',
  'dbhost' => '172.17.0.13:3306',
  'dbport' => '',
  'dbtableprefix' => 'oc_',
  'mysql.utf8mb4' => true,
  'dbuser' => 'oc_[REDACTED]',
  'dbpassword' => '[REDACTED]',
  'installed' => true,
  'memcache.local' => '\\OC\\Memcache\\APCu',
);
```

5. If cron is configured to optimize the background jobs, you need to modify the apc configuration in PHP as follows:

Run `vim /etc/php.d/40-apcu.ini` to open the PHP APCu configuration file, change

`;apc.enable_cli=0` to `apc.enable_cli=1` (note that the semicolon at the beginning needs to be

removed), save the file, and exit the editor. If the `/etc/php.d/40-apcu.ini` path doesn't exist, search for and edit the `.ini` configuration file whose name contains `apc` or `apcu` in the `/etc/php.d/` directory.

```
[root@VM-0-10-centos data]# vim /etc/php.d/40-apcu.ini
; Enable APCu extension module
extension = apcu.so

; This can be set to 0 to disable APCu
apc.enabled=1

; Setting this enables APCu for the CLI version of PHP
; (Mostly for testing and debugging).
apc.enable_cli=1
```

## Configuring Client Access

Nextcloud provides official desktop sync clients and mobile clients, which can be downloaded from the Nextcloud official website or the app stores of different platforms. When configuring Nextcloud, you need to enter its server address (domain name or IP) and your username and password to log in to and use the client.

# Mounting COS to Windows Server as Local Drive

Last updated : 2024-03-25 15:16:26

## Overview

COS can be used on Windows mainly with APIs, COSBrowser, or COSCMD.

However, users using Windows Server can only use COSBrowser as cloud storage, which is not friendly to run programs or perform operations. In this case, you can mount the cost-effective COS to Windows Server as local drive by reading this guide.

### Note:

Examples given in this document apply only to Windows 7 or Windows Server 2012/2016/2019/2022.

## Directions

### Download and installation

Three types of software are involved in examples given in this document. You can install the software version that is compatible with your system.

1. Go to [GitHub](#) to download WinFsp.

`winfsp-1.12.22301` is downloaded as an example. You can then install it with the default options.

### Note:

For Windows Server 2012 R2, WinFsp 1.12.22242 is not compatible, but WinFsp 1.11.22176 is compatible.

2. Go to [Git](#) or [GitHub](#) to download Git.

`Git-2.38.1-64-bit` is downloaded as an example. You can then install it with the default options.

3. Go to [Rclone](#) or [GitHub](#) to download Rclone.

`rclone-v1.60.1-windows-amd64` is downloaded as an example. Note that you only need to decompress it to any directory named in English (decompressing to a path containing Chinese characters might cause an error). In this example, the package is decompressed to the `E:\\AutoRclone`.

### Note:

If GitHub cannot be opened or offers a low download speed, you can find another way for the download.

### Rclone configuration

#### Note:

The following configuration process takes `rclone-v1.60.1-windows-amd64` as an example. Note that the configuration process may vary by version.

1. Open any folder, find **This PC** in the left navigation pane, right-click and select **Properties > Advanced system settings > Environment Variables > System variables > Path\*\***, and click **New**.
2. In the pop-up window, enter the path where Rclone is decompressed (E:\\AutoRclone) and then click **OK**.
3. Open Windows PowerShell and run the `rclone --version` command to check whether Rclone is installed successfully.
4. If Rclone is installed successfully, run the `rclone config` command in Windows PowerShell.
5. In Windows PowerShell, enter **n** and then press **Enter** to create a New remote.
6. In Windows PowerShell, enter the name of the disk (for example, `myCOS` ) and then press **Enter**.
7. From the options that are displayed, select the option containing `Tencent COS` (that is, enter **5**) and then press **Enter**.

```
Option Storage.
Type of storage to configure.
Choose a number from below, or type in your own value.
 1 / IFichier
   \ (fichier)
 2 / Akamai NetStorage
   \ (netstorage)
 3 / Alias for an existing remote
   \ (alias)
 4 / Amazon Drive
   \ (amazon cloud drive)
 5 / Amazon S3 Compliant Storage Providers including AWS, Alibaba, Ceph, China Mobile, Cloudflare, ArvanCloud, DigitalOcean, Dreamhost, Huawei OBS, IBM COS, IDrive e2, IOMOS Cloud, Lyve Cloud, Minio, Netease, RackCorp, Scaleway, SeaweedFS, Storj, Tencent COS, Qiniu and Wasabi
   \ (s3)
 6 / Backblaze B2
   \ (b2)
```

8. From the options that are displayed, select the option containing `TencentCOS` (that is, enter **21**) and then press **Enter**.

```
20 / Storj (S3 Compatible Gateway)
   \ (Storj)
21 / Tencent Cloud Object Storage (COS)
   \ (TencentCOS)
22 / Wasabi Object Storage
   \ (Wasabi)
23 / Qiniu Object Storage (Kodo)
   \ (Qiniu)
24 / Any other S3 compatible provider
   \ (Other)
provider> 21_
```

9. When `env_auth>` is displayed, press **Enter**.
10. When `access_key_id>` is displayed, enter the `SecretId` of COS and then press **Enter**.

**Note:**

Using sub-account permissions is recommended. You can go to [Manage API Key](#) to view your `SecretId` and `SecretKey`.

11. When `secret_access_key>` is displayed, enter the `SecretKey` of COS and then press **Enter**.

12. Choose the region of the bucket according to the gateway addresses of the Tencent Cloud regions that are displayed.

The Guangzhou region is used as an example herein. Therefore, you can enter `4` ( `cos.ap-guangzhou.myqcloud.com` ) and then press **Enter**.

13. Select the object permission (such as `default` or `public-read` ) as needed, which takes effect only for objects that are uploaded later. `default` is used as an example herein. Therefore, you can enter `1` and then press **Enter**.

```

/ Owner gets Full_CONTROL.
1 | No one else has access rights (default).
  \ (default)
/ Owner gets FULL_CONTROL.
2 | The AllUsers group gets READ access.
  \ (public-read)
/ Owner gets FULL_CONTROL.
3 | The AllUsers group gets READ and WRITE access.
  \ Granting this on a bucket is generally not recommended.
  \ (public-read-write)
/ Owner gets FULL_CONTROL.
4 | The AuthenticatedUsers group gets READ access.
  \ (authenticated-read)
/ Object owner gets FULL_CONTROL.
5 | Bucket owner gets READ access.
  \ If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.
  \ (bucket-owner-read)
/ Both the object owner and the bucket owner get FULL_CONTROL over the object.
6 | If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.
  \ (bucket-owner-full-control)
acl> 1

```

14. Select the storage class for your objects uploaded to COS. `Default` is used as an example herein. Therefore, you can enter `1` and then press **Enter**.

```

Option storage_class.
The storage class to use when storing new objects in Tencent COS.
Choose a number from below, or type in your own value.
Press Enter to leave empty.
1 / Default
  \ ()
2 / Standard storage class
  \ (STANDARD)
3 / Archive storage mode
  \ (ARCHIVE)
4 / Infrequent access storage mode
  \ (STANDARD_IA)
storage_class> 1

```

Default: default option

Standard storage class: STANDARD

Archive storage mode: ARCHIVE

Infrequent access storage mode: STANDARD\_IA

**Note:**

To use the INTELLIGENT TIERING or DEEP ARCHIVE storage class, use the **modifying the configuration file** method and then set the value of `storage_class` to `INTELLIGENT_TIERING` or `DEEP_ARCHIVE`. For more information about the storage class, see [Overview](#).

15. When `Edit advanced config? (y/n)` is displayed, press **Enter**.

16. After confirming the information, press **Enter**.

17. Enter **q** to complete the configuration.

```
Configuration complete.
Options:
- type: s3
- provider: TencentCOS
- access_key_id: AKIDd8009ettefT
- secret_access_key: oNgjWyCBuXy
- endpoint: cos.ap-guangzhou.myqcloud.com
- acl: default
Keep this "myCOS" remote?
y) Yes this is OK (default)
e) Edit this remote
d) Delete this remote
y/e/d>

Current remotes:

Name          Type
-----
myCOS         s3
```

## Modifying the configuration file

After the preceding configuration is complete, a configuration file named `rclone.conf` will be generated in the `C:\\Users\\Username\\AppData\\Roaming\\rclone` directory. You can directly modify the file to update the Rclone configuration. If the file is not found, you can run the `rclone config file` command in the command line window to view the Rclone configuration file.

## Mounting COS as local drive

1. Open the installed Git Bash and enter the command in it. Two use cases are provided here for your choice as needed.

To mount COS as a shared drive on LAN (recommended), run the following command:

```
rclone mount myCOS:/ Y: --fuse-flag --VolumePrefix=\\server\\share --cache-dir E:\\
```



To mount COS as a local drive, run the following command:

```
rclone mount myCOS:/ Y: --cache-dir E:\\temp --vfs-cache-mode writes &
```

myCOS: Replace it with the user-defined disk name.

Y: Replace it with the drive letter you want to assign to the drive. Ensure that it does not overlap with other drive letters.

E:\\temp: A local cache directory, which can be set as needed. Note that you have the permission for the directory.

If “The service rclone has been started” is displayed, the mount is successful.

2. Enter **exit** to close the terminal.

3. Find the myCOS(Y:) drive in **This PC**.

If you open this drive, you can see all buckets in the Guangzhou region. You can upload, download, create, delete files, and do more via the drive as needed.

#### Note:

If any error is reported during the process, you can view the error messages from Git Bash.

If you delete a bucket from the drive, the bucket will be deleted, regardless of whether there are objects stored in the bucket or not.

If you change the name of a bucket from the drive, the bucket name in COS will also be changed.

## Running the mounted drive automatically at startup

The mounted drive will disappear after the server is restarted. Therefore, you can perform the following operations to set the drive to auto-run at startup.

1. Create the `startup_rclone.vbs` and `startup_rclone.bat` files in the `E:\\AutoRclone` directory.

#### Note:

Use correct encoding when you create text files through PowerShell. Otherwise, the generated .bat and .vbs files cannot be executed.

2. In `startup_rclone.bat`, write the following mount command:

If COS is mounted as a shared drive on a LAN, run the following command:

```
rclone mount myCOS:/ Y: --fuse-flag --VolumePrefix=\\server\\share --cache-dir E:\\
```

If COS is mounted as a local drive, run the following command:

```
rclone mount myCOS:/ Y: --cache-dir E:\\temp --vfs-cache-mode writes &
```

3. In `startup_rclone.vbs`, write the following code:

```
CreateObject("WScript.Shell").Run "cmd /c E:\\AutoRclone\\startup_rclone.bat",0
```

#### Note:

Please replace the path with the actual one.

4. Cut the `startup_rclone.vbs` file to the

`%USERPROFILE%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup` directory.

5. Restart the server.

**Note:**


Usually, the successful mounting prompt will be displayed tens of seconds after auto-mounting configuration is complete and the server is restarted.

## More

Using a third-party tool to mount COS to a Windows server as local drive is also available. The following shows the mounting procedure using the TntDrive tool:

1. Download and install TntDrive.
2. Open TntDrive and then click **Account > Add New Account** to create an account.

Edit Account
— □



## Edit Account

Edit account details and click Save changes

**Account Name:**

Assign any name to your account.

**Account Type:**

Amazon S3 Compatible Storage

Choose the storage you want to work with. Default is Amazon S3 Storage.

**REST Endpoint:**

Specify S3-compatible API endpoint. It can be found in storage documentation. Example: rest.server.com:8080

**Access Key ID:**

Required to sign the requests you send to Amazon S3, see more details at <https://tntdrive.com/keys>

**Secret Access Key:**

Required to sign the requests you send to Amazon S3, see more details at <https://tntdrive.com/keys>

**Use secure transfer (SSL/TLS)**

If checked, all communications with the storage will go through encrypted SSL/TLS channel

[Advanced S3-compatible storage settings](#)

✓ Save changes

✗ Cancel

The main parameters are described as follows:

**Account Name:** user-defined account name

**Account Type:** account type. As COS is compatible with S3, you can select **Amazon S3 Compatible Storage**.

**REST Endpoint:** region of your bucket. For example, if your bucket resides in the Guangzhou region, set this parameter to `cos.ap-guangzhou.myqcloud.com`.

**Access Key ID:** the value of `SecretId`, which can be created/obtained at [Manage API Key](#).

**Secret Access Key:** the value of `SecretKey`

3. Click **Add new account**.

4. In the TntDrive window, click **Add New Mapped Drive** to create a mapped drive.

**Add New Mapped Drive** [online help](#)

**Add New Mapped Drive**  
Specify new drive properties and click Add new drive

**Storage account:**  
New Account

Select account from the list. You may choose existing account or add the new one.

**Amazon S3 bucket:**  
yuming1-130:

Select or specify bucket name and optional path, for example bucket-name/path/

**Mapped drive letter:**  
W:

Select available letter for your drive.

[advanced properties..](#)

The main parameters are described as follows:

**Amazon S3 bucket:** path or name of the bucket. You can click the icon on the right to select a bucket. In this example, the bucket residing in the Guangzhou region that is set in step 2 is selected (mapping a bucket as a single network drive).

**Mapped drive letter:** drive letter of the mapped drive, which cannot overlap with existing drive letters.

5. Click **Add new drive**.

6. Find the drive in **This PC**. If you want to map all buckets to the Windows server, repeat the steps above.

# Setting up Image Hosting Service with PicGo, Typora, and COS

Last updated : 2024-03-25 15:16:26

## Overview

The image hosting service provides various features such as image storage, processing, and distribution to sustain countless blog sites and community forums worldwide on the backend. COS is a distributed storage service launched by Tencent Cloud to store massive numbers of files, with higher performance and reliability guaranteed. You can use **COS** to set up an image hosting service.

The strengths of COS in the image hosting scenarios include:

**Low costs:** The unit price of storage is low, and you only need to pay for what you use.

**Unrestricted speed:** Upload and download speeds are not restricted, so users no longer need to wait for slow loading, enjoying a better access experience.

**High availability:** COS offers an SLA for high availability, where stored data has a guaranteed durability of up to 99.9999999999%.

**Unlimited capacity:** COS stores high numbers of files in a distributed manner for on-demand capacity use.

## Practice Scenario

### Scenario 1: Adding images to set up an image hosting service with COS

The following tools are used in this scenario:

**PicGo:** A tool that supports multiple cloud storage configurations and quickly generates image URLs.

**Typora:** A lightweight Markdown file editor that supports multiple output formats and allows you to quickly upload local images to an image hosting service.

### Directions

1. Install PicGo and set relevant COS parameters.

#### Note:

PicGo 2.3.1 is used in this scenario. Note that the configuration process may vary by version.

After downloading PicGo from [PicGo website](#) and installing it, find **Tencent Cloud COS** in the image hosting service settings and configure the following parameters:

Choose COS version: Select COS v5.

Set SecretId: A developer-owned secret ID used for the project. It can be created and obtained at [Manage API Key](#).

Set `SecretKey`: A developer-owned secret key used for the project. It can be obtained at [Manage API Key](#).

Set `Bucket`: It is a bucket, i.e., a container used for data storage. For more information, see [Bucket Overview](#).

Set `Appld`: It is a unique user-level resource identifier for COS access, which can be obtained on the [Manage API Key](#) page.

Set `Storage Region`: It is the region information of the bucket. For enumerated values such as `ap-beijing`, `ap-hongkong`, and `eu-frankfurt`, see [Regions and Access Endpoints](#).

Set `Storage Path`: It is the path where the image is stored in the COS bucket.

Set `Custom URL`: This parameter is optional. If you have configured a custom origin domain name for the storage space specified above, you can enter it here. For more information, see [Enabling Custom Origin Domains](#).

Set `URL Suffix`: Add a COS data processing parameter to the URL suffix to implement image compression, cropping, format conversion, and other operations. For more information, see [Image Processing](#).

2. Configure Typora (optional).

#### Note:

If your editing requirement does not involve Markdown, you can skip this step and just use the PicGo tool installed in the previous step as the image hosting tool.

Configure as follows:

1. In **Image** of Typora's preferences, configure the following:

Select **Upload image** for **When Insert...**

In **Image Upload Settings**, select **PicGo.app** and set the location of PicGo.exe you just installed.

2. Restart Typora for the settings to take effect.

3. Enter the Typora editor area, drag and drop or paste an image directly to upload it and automatically replace it with a COS file URL. (If it is not automatically replaced with a COS URL after pasting, check whether the server in PicGo is enabled.)

## Scenario 2: Quickly migrating images in an image hosting repository to COS

Taking an image hosting service as an example, you can find the local image hosting folder, or download the entire folder from the internet, and then transfer all the images in the folder to a COS bucket. Then, uniformly replace the URL domain name to restore the website.

### Directions

#### Step 1. Download images in the original image hosting service

Log in to the original image hosting website and download the previously uploaded image folder.

#### Step 2. Create a COS bucket and set up hotlink protection

1. Sign up for a Tencent Cloud account and create a bucket with access permissions of **public read/private write** as instructed in [Creating a Bucket](#).

2. After the bucket is created, enable hotlink protection in the bucket as instructed in [Setting Hotlink Protection](#) to avoid images from being hotlinked.

### Step 3. Upload the folder to the bucket

In the COS bucket you just created, click **Upload Folder** to upload the prepared image folder to the bucket.

#### Note:

If the number of images is high, you can also use [COSBrowser](#) to upload images quickly.

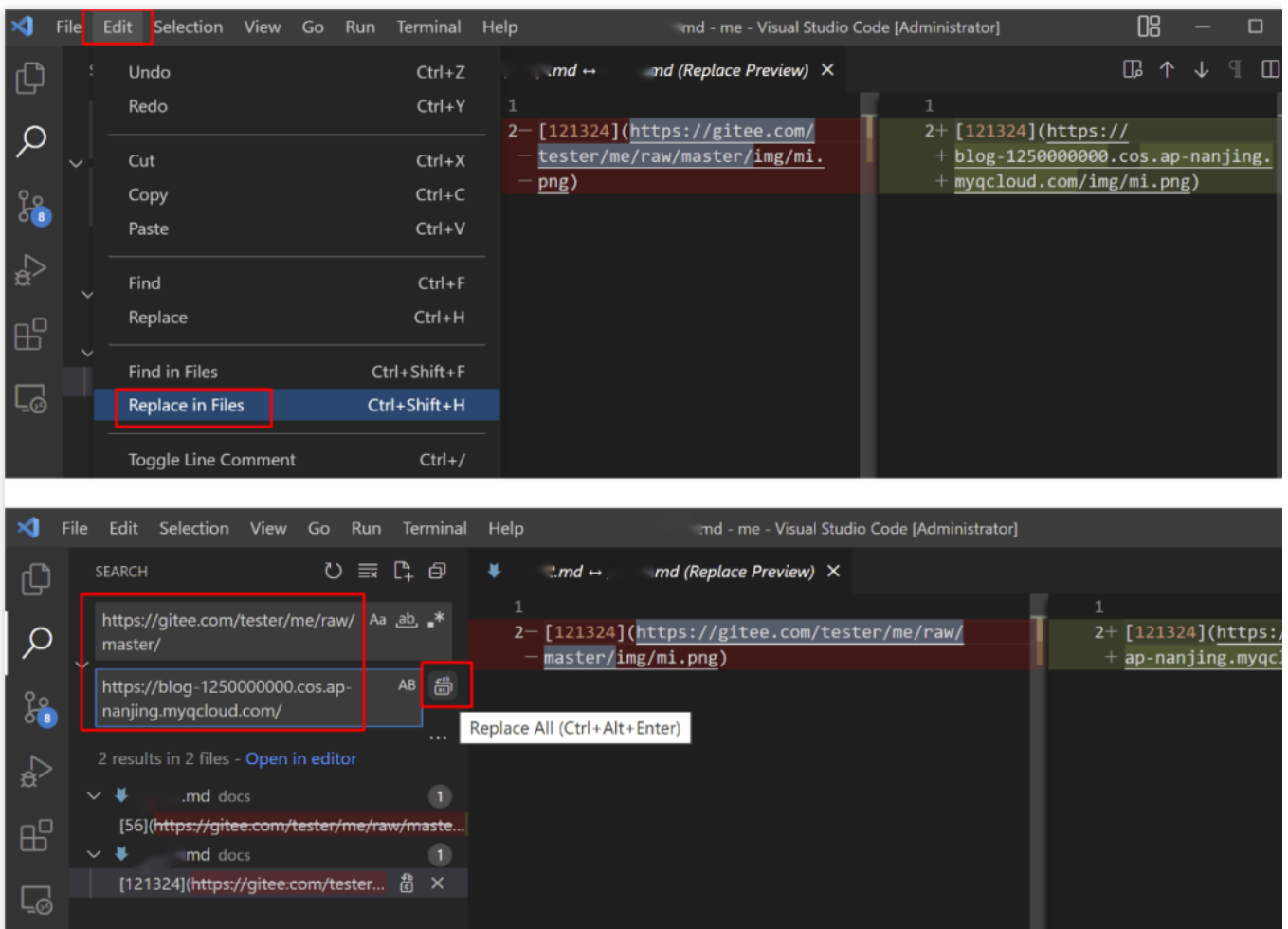
### Step 4. Globally replace the domain name

On the bucket overview page in the COS console, copy the default domain name of the bucket (you can also associate a custom CDN acceleration domain name). Then, use a common code editor to search for and replace the invalid URL prefix globally with the default domain name of the COS bucket.

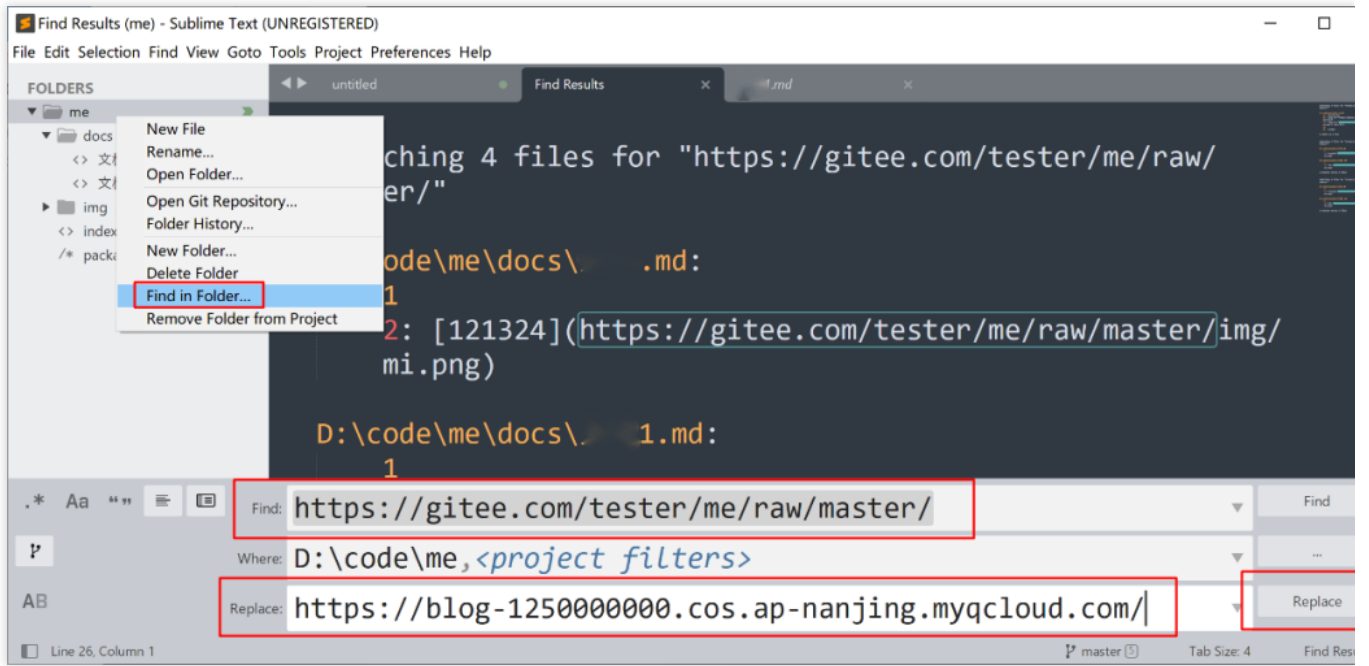
#### Note:

For more information on the default domain name, see [Regions and Access Endpoints](#).

Example search-and-replace with Visual Studio Code:



Example search-and-replace with Sublime Text:





# Managing COS Resource with CloudBerry Explorer

Last updated : 2024-03-25 15:16:26

## Overview

CloudBerry Explorer is a client tool for COS management. It can mount COS on Windows and other operating systems for you to easily access, move, and manage files in COS.

## Supported Systems

Windows and macOS.

## Download Address

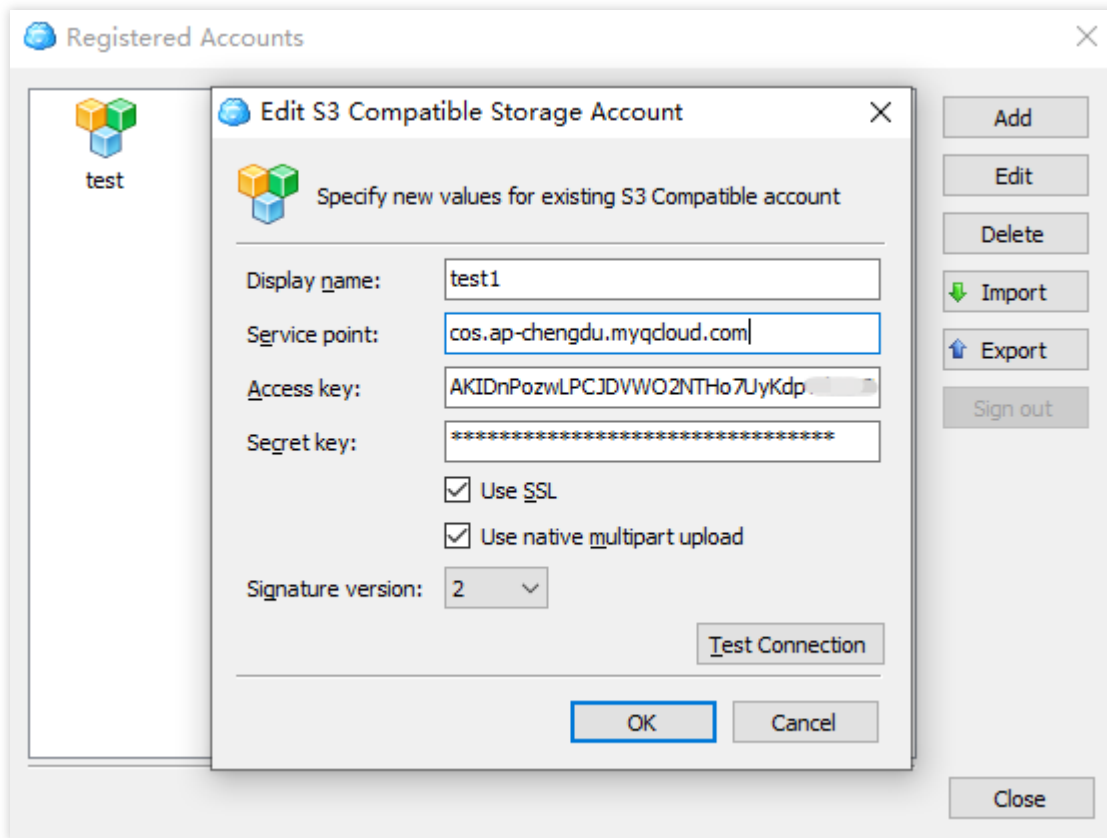
Download CloudBerry Explorer [here](#).

## Installation and Configuration

### Note:

The following configuration process takes CloudBerry Explorer Windows v6.3 as an example. Note that the configuration process may vary by version.

1. Double-click the installation package and complete the installation as prompted.
2. Open the tool and double-click **S3 Compatible**.
3. Configure the following information in the pop-up window, click **Test Connection**, and wait until the connection is successful.



The configuration items are as described below:

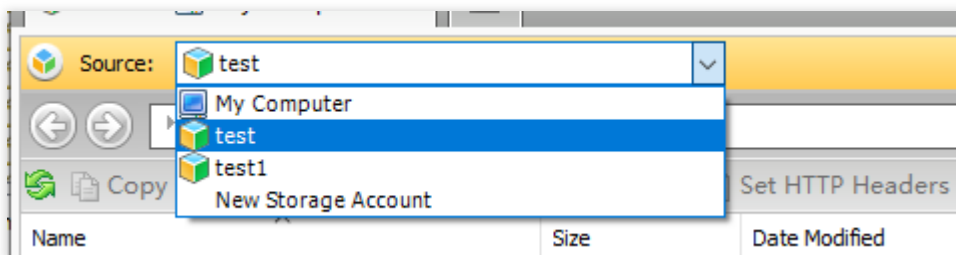
Display name: Enter a custom username.

Service point: The format is `cos.<Region>.myqcloud.com`; for example, to access a bucket in Chengdu region, enter `cos.ap-chengdu.myqcloud.com`. For applicable region abbreviations, see [Regions and Access Endpoints](#).

Access key: Enter the `SecretId`, which can be created and obtained on the [Manage API Key](#) page.

Secret key: Enter the `Secretkey`, which can be created and obtained on the [Manage API Key](#) page.

4. After adding the account information, select the configured username in **Source** to view the list of buckets under the username. At this point, the configuration is completed.



# Managing COS File

## Querying bucket list

Select the configured username in **Source** to view the list of buckets under the username.

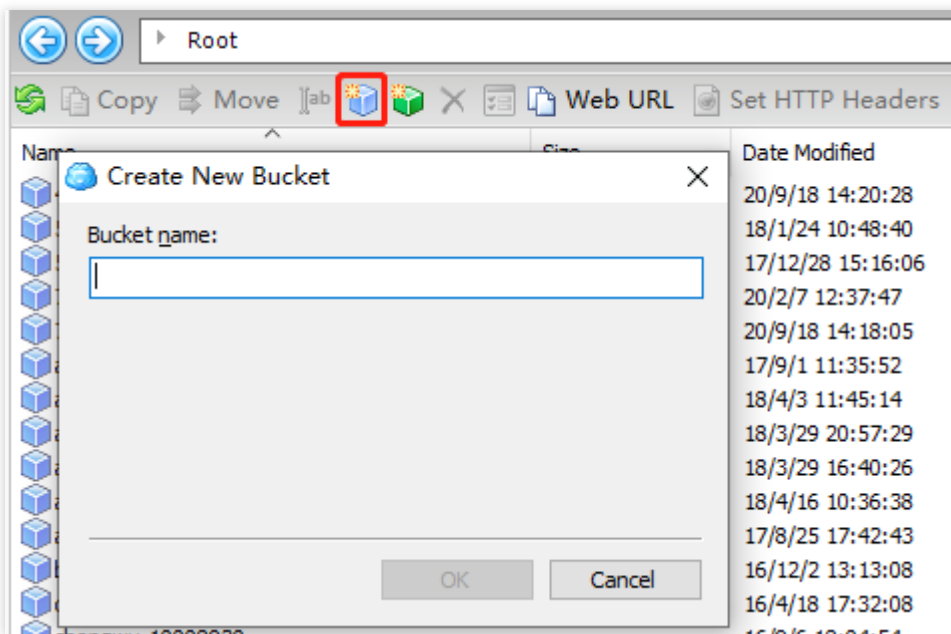
### Note:

With this operation, you can only view the buckets in the region configured by the **Service point**. To view buckets in other regions, click **File > Edit Accounts**, select a username, and change **Service point** to another region.

## Creating a bucket

Click the icon as shown below, enter the full bucket name in the pop-up window such as `examplebucket-1250000000`, and click **OK**.

For bucket naming conventions, see [Bucket Overview](#).

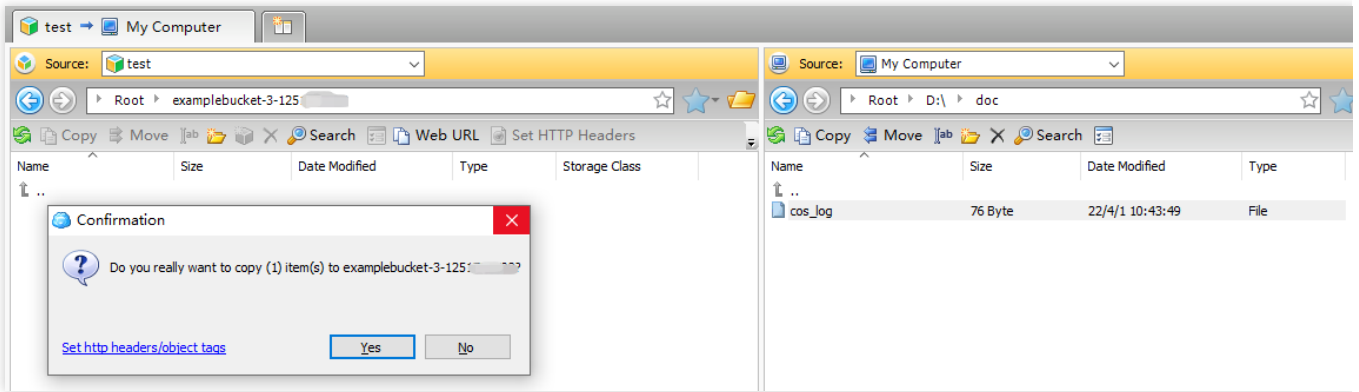


## Deleting a bucket

Right-click the target bucket in the bucket list and select **Delete** in the context menu.

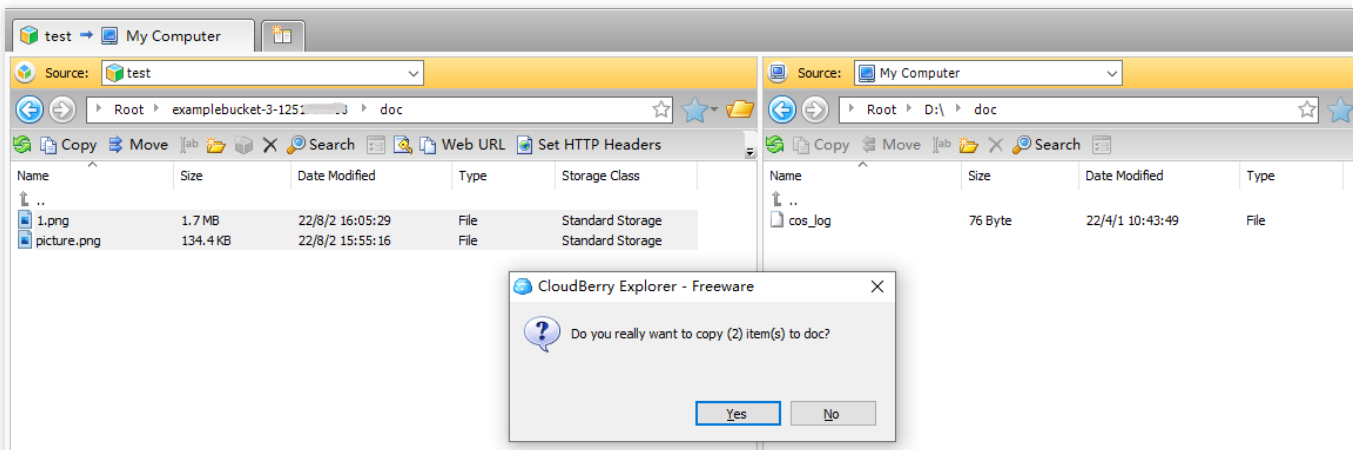
## Uploading an object

In the bucket list, select the destination bucket or path, select the object to be uploaded on the local computer, and drag and drop it to window on the left.



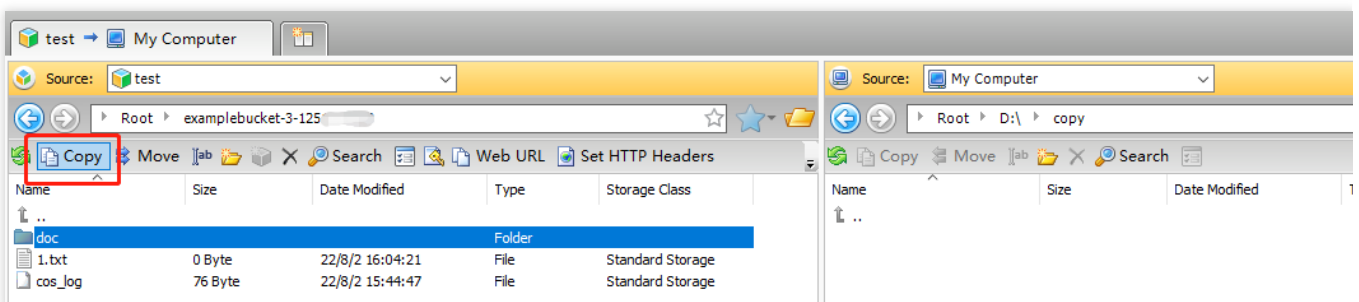
## Downloading an object

Select the target object in the window on the left and drag and drop it to a folder on the local computer on the right.



## Copying an object

Select the destination path in the right window of the tool, right-click the target object in the left window, select **Copy**, and confirm in the pop-up window.



## Renaming an object

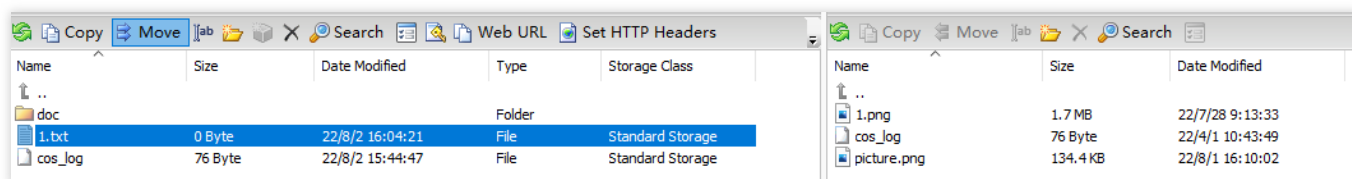
Right-click the target object in the bucket, select **Rename**, and enter a new name.

## Deleting an object

Right-click the target object in the bucket and select **Delete**.

## Moving an object

Select the destination path in the right window of the tool, right-click the target object in the left window, select **Move**, and confirm in the pop-up window.



## Other features

In addition to the above features, CloudBerry Explorer also allows you to set object ACLs, view object metadata, customize headers, and get object URLs.