# Cloud Object Storage
# Best Practice

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

# Contents

# Best Practice Overview

Last updated: 2023-09-12 17:33:07

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 | Note |
| --- | --- |
| Access Control and Permission Management | Access control and permission management is one of the most practical features of Tencent Cloud COS. The practices in this section will help you understand the following:<br>• ACL Access Control<br>• CAM Access Management<br>• Granting Sub-account Access to COS<br>• Permission Settings<br>• COS API Authorization Policy<br>• Security Guide for Using Temporary Keys<br>• Generating and Using Temporary Keys<br>• Authorize Sub-account to Retrieve Tag List<br>• Condition Key Description and Usage<br>• Authorize sub-accounts under other primary accounts to operate buckets |
| Performance | Tencent Cloud COS supports performance scaling to achieve higher request rates. For steps on obtaining higher request rates, please refer to Request Rate and Performance Optimization. |
| Accessing COS Using the AWS S3 SDK | COS offers APIs compatible with AWS S3. You can access files in COS using the AWS S3 SDK with simple configurations. |
| Backup and Disaster Recovery | This guide outlines three backup scenarios and provides suitable backup solutions for each migration scenario.<br>• Disaster Recovery and High Availability Architecture Based on Cross-Region Replication<br>• Cloud Data Backup<br>• Local Data Backup |
| Endpoint Management Practice | • Learn how to configure a custom HTTPS domain name for accessing Tencent Cloud COS. For more information, see Configuring Custom Domain Name for HTTPS Access.<br>• Learn how to configure cross-origin access rules in Tencent Cloud COS. For more information, see Setting Cross-Origin Access.<br>• Learn how to host a static website in Tencent Cloud COS. For more information, please refer to Hosting a Static Website.<br>• Learn how to set up a frontend single-page application in Tencent Cloud COS. For more information, please refer to Building a Frontend Single-Page Application with COS Static Website Feature. |
| Direct data upload | This practice covers the following data direct upload scenarios:<br>• Web Direct Upload Practice<br>• Direct Upload Practice for WeChat Mini Programs<br>• Direct Upload from Mobile Applications<br>• Direct Upload with uni-app |
| Data Security | • Introducing data security solutions for users from three aspects: proactive protection, real-time monitoring, and post-event tracing. For more information, please refer to Data Security Overview.<br>• Learn how to configure hotlink protection in Tencent Cloud COS to control access sources. For more information, please refer to Hotlink Protection Practices. |
| Data verification | • Learn how to ensure the integrity of uploaded data in Tencent Cloud COS using MD5 checksums. For more information, see MD5 Checksum. |

| | |
|---|---|
| | • Learn how to perform data validation using CRC64 checksums. For more information, see CRC64 Validation. |
| Big Data Practices | • Introducing the steps to use Tencent Cloud COS as the Deep Storage for Druid. For more information, please refer to Using COS as Druid's Deep Storage.<br>• Learn how to use Terraform to manage Tencent Cloud COS. For more information, please refer to Managing COS with Terraform.<br>• Learn how to use DataX to import or export data from Tencent Cloud COS. For more information, see Using DataX to Import or Export COS.<br>• Learn how to configure COSN using CDH. For more information, see CDH Configuration Guide for COSN.<br>• Introducing Tencent Cloud COS Ranger, a permission system solution. For more information, please refer to COS Ranger Permission System Solution.<br>• Learn how to use Stream Compute Service (Oceanus) to connect to Tencent Cloud COS. For more information, please refer to Connecting COS with Stream Compute Service (Oceanus). |
| Using COS in Third-Party Application | • Learn how to use the general configuration of COS in S3-compatible third-party applications to store data on Tencent Cloud COS. For more information, see Using COS General Configuration in S3-Compatible Third-Party Applications.<br>• This practice describes how to save forum attachments to COS by configuring the remote attachment feature. For more information, please see Storing Discuz! Forum Remote Attachments to COS.<br>• Learn how to store multimedia content from WordPress on Tencent Cloud COS using third-party plugins. For more information, please refer to Storing WordPress Multimedia Content on COS. |
| Using APIs to zip files | You can package multiple files through an API. |
| COS Cost Optimization Solutions | This document primarily introduces cost optimization solutions for migrating data to the cloud. |

# Access Control and Permission Management
# ACL Practices

Last updated: 2023-09-20 10:59:55

## 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 access policies, ACL management has the following limitations:**

- Supports granting permissions only to Tencent Cloud accounts.
- Supports only five action groups: read object, write object, read ACL, write ACL, and full permissions.
- No support for granting effective conditions.
- Does not support explicit deny effect.

**Control granularities supported by ACLs**

- Bucket
- Object Key Prefix (Prefix)
- Object

## Control Elements of ACLs

When creating a bucket or object, the resource's root account will have full permissions to the resource, which cannot be modified or deleted. You can use ACLs to grant access permissions to other Tencent Cloud accounts.

Below is an example of a bucket ACL. In this example, 100000000001 represents the root account, 100000000011 represents a sub-account under the root account, and 100000000002 represents another root account. The ACL includes an Owner element that identifies the bucket owner, who has full permissions to the bucket. The Grant element grants anonymous read access, expressed as `http://cam.qcloud.com/groups/global/AllUsers` READ permission.

```
<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="RootAccount">
        <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 by using the definition of the principal in CAM for authorization. The description is as follows:

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

**Sub-account**
You can authorize your sub-accounts under your root account (e.g., 100000000011) or sub-accounts under other root accounts by using the definition of the principal in CAM. The description is as follows:

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

**Anonymous Users**
You can grant access permissions to anonymous users by using the definition of the principal in CAM for authorization. The description is as follows:

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

### Action 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 | Unavailable |
| READ_ACP | Reads the ACL for the bucket | Reads the ACL under the directory | Reads object ACLs |
| WRITE_ACP | Modifies the ACL of bucket | Modifies the ACL in the directory | Modifies the ACL of object |
| FULL_CONTROL | Performs any operation on buckets and objects | Performs any operation on objects in the directory | Performs any operation on objects |

### Standard ACL description

COS supports a range of predefined authorizations, called standard ACLs. The following table lists the meaning of each standard ACL.

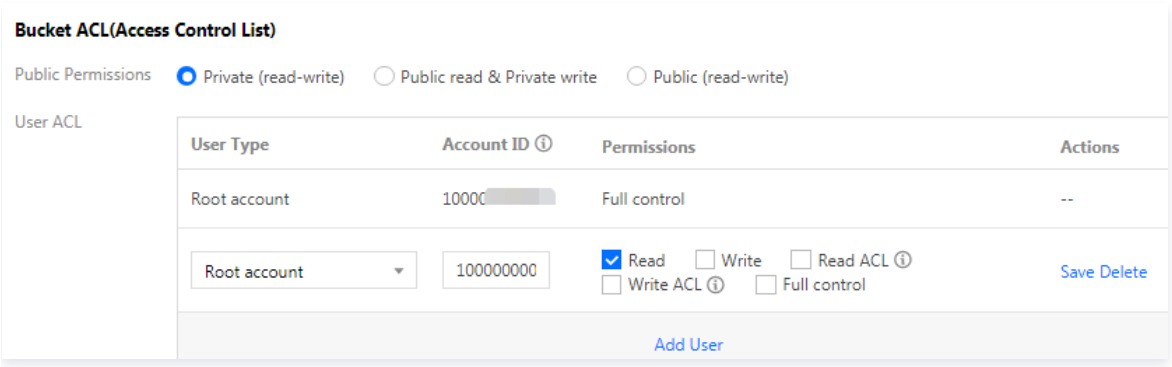
> ⚠ **Note**
> A root account always has the FULL_CONTROL permission, which is not described here.

| 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. |

## How to Use

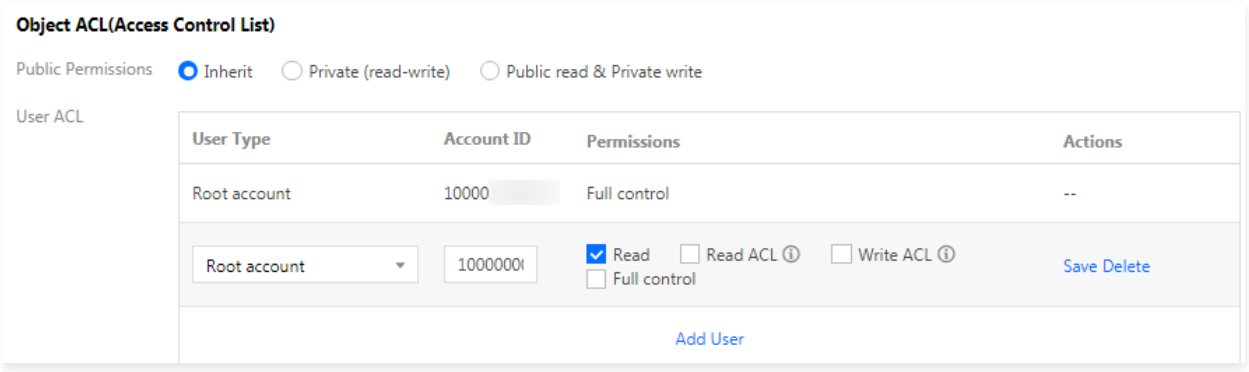
### Operating ACLs via the console

**Setting ACL for a bucket**
The following example allows another root account to have read access to a specific **bucket**:




**Set ACL for an object**

The following example allows another root account to have read access to a specific **object**:




> **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 API Operations for ACL

**Bucket ACL**

| API | Operation | Description |
|---|---|---|
| PUT Bucket acl | Setting bucket ACL | Sets an ACL for a bucket |
| GET Bucket acl | Querying bucket ACL | Queries the ACL of a bucket |

**Object ACL**

| API | Operation | Description |
|---|---|---|
| PUT Object acl | Setting object ACL | Sets the ACL of a specified object in a bucket |
| GET Object acl | Querying object ACL | Queries the ACL of an object |

# CAM Practices

Last updated: 2023-09-20 11:04:14

## Overview

Cloud Access Management (CAM) is a Tencent Cloud authentication and authorization service, which 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.

## SDK

### Granting access to resources under the root account

You can grant the access to the 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.

### Data integrity

CAM currently supports data synchronization 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. Additionally, CAM uses cache (currently valid for one minute) to improve performance, and any policy update does not take effect until cache expires.

## Scenarios

**Enterprise sub-account access permission management**
Employees in different positions within an organization need to have the least privilege access to the company's cloud resources. For example, a company may have numerous cloud resources, including CVM, VPC instances, CDN instances, COS buckets, and objects. The company has various employees, such as developers, testers, and operations staff.
Some developers require read/write access to the development machine cloud resources related to their projects, while testers need read/write access to the test machine cloud resources for their projects. Operations staff are responsible for purchasing machines and managing daily operations. When an employee's role or project involvement changes, the corresponding permissions will be revoked.
**Cross-enterprise access permission management**
There are cases where enterprises may need to share their cloud resources. For example, a company which has many cloud resources wants to focus on product R&D and outsource the operation of its cloud resources to another company. It also need to revoke all permissions that have been granted as soon as the outsourcing service contracts are terminated.

## Policy Syntax

A CAM policy consists of several elements and is used to describe specific information about authorization. Core elements include principal, action, resource, condition, and effect. For more information, please see Access Policy Language Overview.

**Note**
- There is no particular sequence in the description of policy syntax. However, please note that the `action` element is case-sensitive.
- If there are no particular conditions required, the `condition` element is optional.
- You cannot define the `principal` element in the console, but only through the policy management APIs or policy syntax parameters.

### Core elements

| Core elements | Description | Required |
| --- | --- | --- |
| version | Specifies the version of the policy syntax. Valid value: 2.0. | Required |

| | | |
|---|---|---|
| principal | Describes the entity to be authorized by the policy, including users (developers, sub-accounts, anonymous users), and user groups. | This element can only be used in policy management APIs and policy syntax-related parameters |
| statement | Describes the details on a permission or a permission set defined by other elements including effect, action, resource, and condition. One policy has only one statement. | Required |
| action | Describes the action to be allowed or denied. It can be an API operation or a set of API operations. This element is case-sensitive, e.g. `name/cos:GetService`. | Required |
| 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 for the product whose resources you are writing a statement for. | Required |
| condition | Describes the condition for the policy to take effect. A condition consists of 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. | Not required |
| effect | Describes whether the statement result is "allow" or "deny". | Required |

### Policy limits

| Limit | Limit Value |
|---|---|
| The number of user groups within a primary account | 300 |
| The number of sub-accounts within a root account | 1000 |
| The number of roles within a primary account | 1000 |
| The number of user groups a sub-account can join | 10 |
| The number of root accounts a collaborator can work with | 10 |
| The number of sub-accounts within a user group | 100 |
| Number of custom policies that can be created under a root account | 1500 |
| Number of policies that can be directly associated with a user, user group, or role | 200 |
| Maximum character count for a policy syntax | 4096 |

### Policy Example

The following policy example allows a sub-account with ID 100000000011 under the root account with ID 100000000001 (APPID 1250000000) to have **upload** and **download** permissions for the object "exampleobject" in the "examplebucket-bj" bucket in the Beijing region and the "examplebucket-gz" bucket in the Guangzhou region when accessing from 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-1250000000/*",
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-gz-1250000000/exampleobject"
            ],
            "condition": {
```

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

# Granting Sub-Accounts Access to COS

Last updated: 2025-06-23 10:46:58

## 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 users (sub-accounts) with similar functions. Based on business requirements, you can create various user groups and associate them with appropriate policies to assign different permissions.

## Instructions

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.

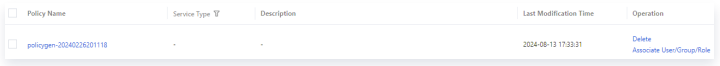
### Step 1: Create a Sub-account

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. Navigate to **Users > User List > Create User** to access the Create User 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 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**: Choose both programmatic access and Tencent Cloud console access. Other configurations can be selected 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.

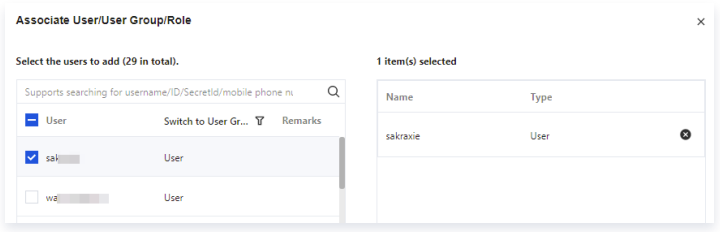
## Step 2. Grant permissions to the sub-account

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

1. Log in to the CAM console.
2. Select **Policies > Create Custom Policy > Create by Policy Syntax** to enter 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 a memorable policy name. If you choose **Blank Template**, you need to input your policy syntax, as detailed in Policy Example. Copy and paste the policy content into the **Policy Content** input box. After confirming the input is correct, click **Finish** to complete the process.
5. After creation, associate the newly created policy with the sub-account.




6. After selecting the sub-account and clicking **OK** to grant permissions, you can use the sub-account to access the specified COS resources.




## 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. Select **User List** to enter the user list page.
3. Click the name of the sub-account to go to the **User Details** page of the sub-account.
4. Click the **API Keys** tab, and then click **Create Key** to generate a 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 APPID>", "<Sub-account SecretId>", "<Sub-account
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, adhere to the principle of least
privilege to reduce usage risks. To obtain a sub-account key, refer to
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 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 APPID> -a <Sub-account SecretId> -s <Sub-account SecretKey> -b <Root account
bucketname> -r <Root account bucket region>
```

Example:

```
coscmd config -u 1250000000 -a ********************************** -s e8Sdeasdfas2238Vi**** -b examplebucket
-r ap-beijing
```

(2) Tencent Cloud console access
After being granted permissions, sub-users can log in to the console at the Sub-user login page by entering the root account ID, sub-user name, and sub-user password. Then, they can access the storage resources under the root account by selecting and clicking **Object Storage** in the **Cloud Products** section.

## Policy Example

Here are some sample policies for typical scenarios. When configuring custom policies, you can copy and paste the reference policies below into the **Edit Policy Content** input box and modify them according to your actual configuration. For more COS common scenario policy syntax, please refer to the Access Policy Language Overview or the **Commercial Use Cases** section in the CAM Product Documentation.

### 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

In this example, only IP ranges `192.168.1.0/24` and `192.168.2.0/24` have read and write permissions, as shown below:
For more information on how to set up effective conditions, please refer to Effective 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: 2023-09-12 17:35:25

## Granting Permission via Bucket Policy

### Prerequisites

1. Create a Bucket
   Granting permission via bucket policy (Policy) is specific to a particular bucket. Therefore, you need to create a bucket first. If you need to grant permission at the account level, please refer to Granting Permission via Access Management (CAM) Use Cases in this document.
2. Prepare the UIN of the Account to be Granted Permission
   In this example, the root account owning the target bucket has a UIN of 100000000001, and its sub-account has a UIN of 100000000011. The sub-account needs to be granted permission to access the target 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
   Navigate to the **Permission Management** of the target bucket, select **Policy Permission Settings > Graphic Settings**, and click to open the **Add Policy** dialog. Then, refer to the authorization use cases in this document for configuration. For detailed instructions on adding a policy, please refer to the Add Bucket Policy document.

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 items | Description |
| --- | --- |
| Effect | Required |
| User | Select **Sub-account** and enter a sub-account UIN, which must be a sub-account under the current root account, such as `100000000011`. |
| Resources | Select a specific resource path, such as `folder/sub-folder/*`. |
| Action | Select **All Actions**. |

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

The configuration information is as follows:

| Configuration items | Description |
| --- | --- |
| Effect | Required |
| User | Select **Sub-account** and enter a sub-account UIN, which must be a sub-account under the current root account, such as `100000000011`. |
| Resources | Select a specific resource path, such as `folder/sub-folder/*`. |
| Action | Read operations (including listing the object list). |

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

The configuration information is as follows:

| Configuration items | Description |
| --- | --- |
| Effect | Required |
| User | Select **Sub-account** and enter a sub-account UIN, which must be a sub-account under the current root account, such as `100000000011`. |
| Resources | Select a specific object key, such as `folder/sub-folder/example.jpg`. |
| Action | All operations. |

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

For this case, we need to add two policies: an **Allow** policy and a **Deny** policy.
1. First, add the **allow** policy. The configuration information is as follows:

| Configuration items | Description |
| --- | --- |
| Effect | Required |
| User | Select **Sub-account** and enter a sub-account UIN, which must be a sub-account under the current root account, such as `100000000011`. |
| Resources | Specify a directory prefix, such as `folder/sub-folder/*`. |
| Action | All operations. |

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

| Configuration items | Description |
| --- | --- |
| Effect | Deny |
| User | Select **Sub-account** and enter a sub-account UIN, which must be a sub-account under the current root account, such as `100000000011`. |
| Resources | Specify the object key to be denied access, such as `folder/sub-folder/privateobject`. |
| Action | All operations. |

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

The configuration information is as follows:

| Configuration items | Description |
| --- | --- |
| Effect | Required |
| User | Select **Sub-account** and enter a sub-account UIN, which must be a sub-account under the current root account, such as `100000000011`. |
| Resources | Specify a prefix, such as `folder/sub-folder/prefix`. |
| Action | All operations. |

## 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: 2024-11-20 20:08:07

> **Note**
> When granting API operation permissions to sub-users or collaborators, please ensure that you follow the principle of least privilege and grant permissions based on business needs. If you grant sub-users or collaborators access to all resources `(resource:*)` or all actions `(action:*)`, there is a risk of data security breaches due to overly broad permissions.

## 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": [
                // Simple upload operation
                "name/cos:PutObject",
                //Upload objects using a form
                "name/cos:PostObject",
                // Multipart upload: Initialize the multipart operation
                "name/cos:InitiateMultipartUpload",
                // Multipart upload: List ongoing multipart uploads
                "name/cos:ListMultipartUploads",
                // Multipart upload: List uploaded parts operation
                "name/cos:ListParts",
                // Multipart upload: Upload part operation
                "name/cos:UploadPart",
                // Multipart upload: Complete all multipart upload operations
                "name/cos:CompleteMultipartUpload",
                // Cancel multipart upload operation
                "name/cos:AbortMultipartUpload"
            ],
            "effect": "allow",
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
            ]
        },
        {
            "action": [
                // Download operation
                "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 | The specific data for authorization can be any resource, a resource with a specified path prefix, a resource with an absolute path, or a combination of these.<br>**Note:** If the path contains Chinese characters, keep them as is. For example, `examplebucket-1250000000/文件夹/文件名.txt`. |
| action | Here, the COS API refers to specifying a single action or a combination of actions based on your requirements, or all actions ( `*` ), such as action being `name/cos:GetService`. **Please note the distinction between uppercase and lowercase letters in English.** |
| condition | Optional condition. For more information, see Element Reference. |

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

## Service API

### Querying the bucket list

For the API "GET Service", if granted operation permission, the policy's action would be `name/cos:GetService`, and the resource would be `*`.

#### Sample

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

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

## Bucket API

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

- To operate on buckets in all regions, set the policy's resource to `*`. **Please note that this policy has a broad scope, which may pose data security risks. Configure with caution.**
- To allow operations only on buckets in a specific region, for example, only allowing operations on buckets with APPID `1250000000` in the Beijing region (ap-beijing), the policy's resource should be `qcs::cos:ap-beijing:uid/1250000000:*`.
- To allow operations only on a specific bucket with a specified name and region, for example, a bucket named `examplebucket-1250000000` in the region "ap-beijing" under the APPID `1250000000`, the policy's resource should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*`.

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

### Create a bucket

To grant permission to access this 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 the Beijing region:

```
{
```

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

> **Note**
> The bucket name must comply with the naming conventions. For more information, please refer to Bucket Naming Conventions.

## Checking a bucket and its permission

To grant permission to access this 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 the object list

To grant permission to access this 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 a bucket

To grant permission to access this 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 permission to access this 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 permission to access this 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 a cross-origin access configuration

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

### Sample

The following policy grants the permission to set a cross-origin access 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 a Cross-Origin Configuration

To grant permission to access this 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 a Cross-Origin Configuration

To grant permission to access this 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 a Lifecycle Configuration

To grant permission to access this 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

To grant permission to access this 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

To grant permission to access this 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 API

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

- To operate on any object, the policy's resource should be set to `*`.
- To restrict operations to any object within a specific bucket, such as the bucket named `examplebucket-1250000000` in the region "ap-beijing" under the APPID `1250000000`, the policy's resource should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*`.
- To restrict operations to a specific bucket and objects with a specified path prefix, such as allowing operations only on objects with the path prefix `doc` in the bucket `examplebucket-1250000000` in the region "ap-beijing" under the APPID `1250000000`, the policy's resource should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*`.
- To operate on an object with a specified absolute path, such as an object with the absolute path `doc/audio.mp3` in the bucket `examplebucket-1250000000` in the region "ap-beijing" under the APPID `1250000000`, the policy's resource should be `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/audio.mp3`.

The `action` field for Object API access policies varies by operation. All Object API access policies are listed below.

### Uploading an object using simple upload

To grant permission to access this 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 includes Initiate Multipart Upload, List Multipart Uploads, List Parts, Upload Part, Complete Multipart Upload, and Abort Multipart Upload. To grant permissions for these operations, the policy's action should be: `"name/cos:InitiateMultipartUpload", "name/cos:ListMultipartUploads", "name/cos:ListParts", "name/cos:UploadPart", "name/cos:CompleteMultipartUpload", "name/cos:AbortMultipartUpload"` `UploadPart","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload"` collection.

### 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 the permission to access 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 an object using a form

To grant permission to access this 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 permission to access this API (Append Object), set the `action` field in the policy to `name/cos:AppendObject`.

### Sample

The following policy grants permission to use append upload for objects with the path prefix `doc` in the bucket `examplebucket-1250000000` residing in the region `ap-beijing` under the account whose `APPID` is `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 permission to access this 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 permission to access this 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/*"
      ]
    }
  ]
}
```

## Replicating Objects

To grant permission to access this 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 a part

For the API "Upload Part - Copy", if granted operation permissions, the target object's action should be a set of `"name/cos:InitiateMultipartUpload", "name/cos:ListMultipartUploads", "name/cos:ListParts", "name/cos:PutObject", "name/cos:CompleteMultipartUpload", "name/cos:AbortMultipartUpload"`, and the source object's action should be "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 permission to access this 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 permission to access this 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/*"
      ]
    }
  ]
}
```

## Configuring a Preflight Request for Cross-origin Access

To grant permission to access this 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 objects

To grant permission to access this 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 One Object

To grant permission to access this 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 permission to access this 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"
      ]
    }
  ]
}
```

## 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 under the path with 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: 2023-09-12 17:35:55

## Feature 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.

## Preparations

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 using temporary keys to authorize access, please ensure that you grant permissions based on your business needs and follow the principle of least privilege. If you grant permissions to all resources ( `resource:*` ) or all actions ( `action:*` ), there is a risk of data security due to excessive permissions.

## Bad Examples and Security Regulations

### Bad example 1. Excessive `resource`

Application A uses COS for registered user profile pictures. Each user's profile picture has a fixed object key `app/avatar/<Username>.jpg`, and also includes different sizes of the profile picture with corresponding object keys `app/avatar/<Username>_m.jpg` and `app/avatar/<Username>_s.jpg`. To simplify usage, the backend sets the resource to `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/*` when generating the temporary key. In this case, a malicious user can obtain the generated temporary key through network packet capture or other means, and then overwrite the profile pictures of any user, resulting in unauthorized access and loss of legitimate user profile picture data.

#### Security regulation

The resource represents the resource path that the temporary key is allowed to access. At this point, you need to fully consider the end-users covered by this path. In principle, the resource specified should only be used by a single user. In this case, the specified `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/*` will obviously cover all users, thus posing a security risk.
In this case, you can consider modifying the user's avatar path to `app/avatar/<Username>/<size>.jpg`. You can then set the resource to `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>/*` to meet the regulatory requirements. Additionally, the resource field supports multiple values in the form of an array. Therefore, you can also explicitly specify multiple resource values to fully restrict the final resource paths that the user has permission to access, such as:

```
"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, with all photos stored under `app/photos/*`. The client needs to list objects (GET Bucket) and download objects (GET Object). To simplify the process, the backend generates a temporary key with the action set to `name/cos:*`. In this case, if a malicious user obtains the temporary key through network packet capture or other means, they can

perform any object operation (such as upload and delete) on any object under the specified resource path, leading to unauthorized access, data loss, and impact on online business.

### Security regulation

The action represents the operations allowed by the temporary key. In principle, you should not use a temporary key that allows all operations, such as `name/cos:*`, to be issued to the frontend. Instead, you must explicitly list all required operations. If the resource paths required for each operation are different, you need to match the **operation** and **resource** paths separately, rather than combining them.
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 users to list and download files of all users (`app/files/*`), but only upload and delete files in their personal directory (`app/files/<Username>/*`). For convenience, the backend mixes the two permissions and four actions (action) when generating temporary keys. The resource paths corresponding to the two permissions are also mixed. In this case, the temporary key will have the greater permissions specified in the resource path, allowing users to list, download, upload, and delete files of all users. Malicious users can exploit this to tamper with or delete other users' files, leading to unauthorized access and exposing legitimate 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 case, you should use:

```
"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/<Username>/*"
    }
]
```

## Bad example 4. Unauthorized access to temporary key

Application D provides a forum application where post attachments are stored in COS. The forum is divided into different user levels, and only active users who have reached a certain threshold of posts can view posts and attachments in specific sections. For some public sections, all users can view the posts and attachments.
When using COS, the backend generates a temporary key based on the section ID passed from the frontend, allowing the download of attachments in the corresponding section. However, during the implementation, the backend does not verify whether the user making the request has permission to access the specified section ID. As a result, anyone can request the interface to obtain a temporary key that grants access to private section attachments, leading to unauthorized access and unintended data leakage due to the inability to effectively restrict access to resources that should be limited.

### 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 follow the principle of least privilege. In practical applications, you can enumerate all possible permissions based on `action` and `resource`. For example, with 3 actions and 2 resources, you can calculate 3 x 2 = 6 allowed resources and corresponding operations. You can then evaluate whether each situation meets your expectations. If the permissions exceed the expected scope, you should consider splitting the permissions by enumerating multiple `statement`.
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: 2023-09-12 17:36:07

> **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.
> - When applying for a temporary key, if you specify a permission scope, the temporary key will only be able to perform operations within that scope. For example, if you apply for a temporary key with the permission to upload files to the bucket examplebucket-1-1250000000, the obtained key **cannot** upload files to examplebucket-2-1250000000, nor **can it** download files from examplebucket-1-1250000000.

## Temporary Key

Temporary Key (Temporary Access Credential) is a restricted permission key obtained through the CAM Cloud API interface.
COS API can use temporary keys to calculate signatures for initiating COS API requests.
When calculating signatures for COS API requests using temporary keys, three fields from the temporary key retrieval interface response are required, as follows:

- TmpSecretId
- TmpSecretKey
- Token

## Benefits to Use a Temporary Key

When using COS on Web, iOS, and Android platforms, calculating signatures with fixed keys cannot effectively control permissions, and placing permanent keys in client code poses a significant risk of leakage. Using temporary keys, however, can conveniently and effectively address permission control issues.
For example, during the temporary key application process, you can set the policy field to restrict operations and resources, limiting permissions within a specified scope.
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**
> For example, when using the Java SDK, you need to obtain the SDK code (version number) on GitHub. If you are prompted that the corresponding SDK version number cannot be found, please confirm whether you have obtained the corresponding version of the SDK 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 | Download | View Sample |
| .NET | Download | View Sample |
| Go | Download | View Sample |
| NodeJS | Download | View Sample |
| PHP | Download | View Sample |

| Python | Download | View Sample |
| --- | --- | --- |

**Note**
To shield the differences between STS interface versions, the STS SDK may return a parameter structure that is not entirely consistent with the STS interface. 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 GitHub, with v3.1.0 or later required.
public class Demo {
    public static void main(String[] args) {
        TreeMap<String, Object> config = new TreeMap<String, Object>();

        try {
            // The SecretId and SecretKey here represent the permanent identity (main account, sub-account, etc.) used to apply for temporary keys. The sub-account must have the necessary permissions to operate the storage bucket.
            String secretId = System.getenv("secretId"); // User's SecretId, it is recommended to use a sub-account key, following the principle of least privilege to minimize usage risks. For obtaining a sub-account key, please refer to https://cloud.tencent.com/document/product/598/37140
             String secretKey = System.getenv("secretKey");//User's SecretKey, it is recommended to use a sub-account key, following the least privilege principle to reduce usage risks. For obtaining a sub-account key, refer to https://cloud.tencent.com/document/product/598/37140
            // Replace with your TencentCloud API key SecretId
            config.put("secretId", secretId);
            // Replace with your TencentCloud API key SecretKey
            config.put("secretKey", secretKey);

            // Setting the domain name:
            // If you are using Tencent Cloud CVM, you can set an internal domain name
            //config.put("host", "sts.internal.tencentcloudapi.com");

            // The validity period of the temporary key, in seconds. The default is 1800 seconds. Currently, the maximum duration for a primary account is 2 hours (i.e., 7200 seconds), and for a sub-account, it is 36 hours (i.e., 129600 seconds).
            config.put("durationSeconds", 1800);

            // Replace with your bucket
            config.put("bucket", "examplebucket-1250000000");
            // Replace with the bucket's region
            config.put("region", "ap-guangzhou");

            // Change this to the allowed path prefix, which can be determined based on the user's login status on your website to allow specific upload paths.
            // Here are a few 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 specified, the user will be allowed to access all resources; unless required by the business, please grant users the appropriate access permissions following the principle of least privilege.
            config.put("allowPrefixes", new String[] {
                    "exampleobject",
                    "exampleobject2"
            });

            // The permission list for the key. The required permissions for this temporary key must be specified here.
```

```
            // Simple upload, form upload, and multipart upload require the following permissions. For other
permission lists, please refer to https://cloud.tencent.com/document/product/436/31923.
            String[] allowActions = new String[] {
                    // Simple Upload
                    "name/cos:PutObject",
                    // Form upload, mini-program upload
                    "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)
             //# Temporary key activation conditions. For detailed rules on setting conditions and the types
of conditions supported by COS, refer to https://cloud.tencent.com/document/product/436/71307.
             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-1250000000/*\"\n" +
             "      ],\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" +
             "  ]\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

### NoSuchMethodError due to JSONObject package conflict

Use 3.1.0 or a later version.

## Accessing COS using a temporary key

When using temporary keys to access COS services via COS API, the temporary sessionToken is passed through the x-cos-security-token field, and the signature is calculated 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 running the following example, please visit the Github project to obtain the Java SDK installation package.

```
// Import cos xml java sdk using the integration method with Maven as described on GitHub.
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 {

        // User Basic Information
        String tmpSecretId = "COS_SECRETID";    // Replace with the temporary SecretId returned by the STS
interface
        String tmpSecretKey = "COS_SECRETKEY";  // Replace with the temporary SecretKey returned by the STS
interface
        String sessionToken = "Token";  // Replace with the temporary Token returned by the STS interface

        // 1. Initialize user authentication information (secretId, secretKey)
        COSCredentials cred = new BasicCOSCredentials(tmpSecretId, tmpSecretKey);
        // 2 Set the bucket region. For more information, please refer to COS regions at
https://cloud.tencent.com/document/product/436/6224
        ClientConfig clientConfig = new ClientConfig(new Region("ap-guangzhou"));
        // 3. Create a COS client
        COSClient cosclient = new COSClient(cred, clientConfig);
        // The bucket name must include the appid
        String bucketName = "examplebucket-1250000000";

        String key = "exampleobject";
        // Upload object, it is recommended to use this interface for files below 20MB.
        File localFile = new File("src/test/resources/text.txt");
        PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);

        // 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 will return the file's etag
            String etag = putObjectResult.getETag();
        } catch (CosServiceException e) {
            // Failure, throw CosServiceException
            e.printStackTrace();
        } catch (CosClientException e) {
```

```
            //Failure, throw CosClientException
            e.printStackTrace();
        }

        // Close the client
        cosclient.shutdown();

    }
}
```

# Authorizing Sub-Account to Get Buckets by Tag

Last updated: 2023-09-12 17:36:20

## Feature 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 as the Owner to the Access Management console and navigate to 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

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.
   - **Action**: Select **Read Operation > GetService (Retrieve 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 the tag in the `key&val` format, replacing `key` with the tag key and `value` with the tag value.
4. Click **Next** and enter the policy name.
5. Click **Done** to complete the creation.

### Policy syntax

1. Go to the CAM policy configuration page.
2. Click **Create Custom Policy > Create by Policy Syntax**.
3. Select to create from a 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 **Done** to complete the creation.

3. Associate the policy with sub-account `SubUser`. On the policy page, find the policy created in step 2 and click **Associate User/Group/Role** on the right side.
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 as the sub-account `SubUser`.
2. On the bucket list page, the **list of accessible buckets for the sub-account will be displayed automatically**.

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. Invoke the GetService API, passing in the tag filter parameters, such as (key, value). The request example is as follows, and more details can be found in 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: 2025-07-10 10:14:11

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 to write policies, adhere to the principle of least privilege by only adding corresponding condition keys for applicable requests (actions) and avoiding the use of wildcard "*" for specified actions, which may result in request failures. For more information on condition keys, refer to the Access Policy Language > Overview document.
- 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 be lowercase.

## 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 following policy example allows a sub-account with ID 100000000002 under the main account ID 100000000001 (APPID 1250000000) to upload and download the object `exampleobject` in the Beijing region's bucket `examplebucket-bj` and the Guangzhou region's bucket `examplebucket-gz`. Access is granted when the IP address is within the `192.168.1.0/24` subnet 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-beijing:uid/1250000000:examplebucket-bj-1250000000/*",
                "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"
                    ]
                }
            }
        }
    ]
}
```

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

## Request parameter versionid

The request parameter `versionid` represents the object's version number. For more information on version control, refer to Versioning Overview. You can use the request parameter `versionid` to specify the object version you want to operate on when downloading (GetObject) or deleting (DeleteObject) an object.

- If `versionid` is not carried, requests will apply to the latest version of the object by default.
- When the `versionid` request parameter is an empty string, it is equivalent to not having the `versionid` request parameter.
- `versionid` request parameter is the string `"null"`. For objects uploaded to a bucket before versioning is enabled, their version IDs will be uniformly set to the string `"null"` 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.
Since `GetObject` retrieves the latest version of an object by default when the request parameter `versionid` is not provided or `versionid` is an empty string, we can use `string_equal_if_exists` 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 the request parameter `versionid` is empty, i.e., `""` , it will also match the "allow" policy, granting access only to requests for the latest version of the object.

```
"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 following two examples, assume that the primary account (uin:100000000001) owns the bucket examplebucket-1250000000. You can use the `cos:content-length` condition key to limit the size of the Content-Length header in the upload request for the sub-user (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

HTTP request content types (MIME) defined in RFC 2616, such as `application/xml` or `image/jpeg`. For more details, please refer to the Request Header List.

### Condition key cos:content-type

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

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

Assuming the primary account (uin:100000000001) owns the bucket examplebucket-1250000000, you can use the `cos:content-type` condition key to restrict the specific content of the Content-Type header in the sub-user's (uin:100000000002) upload request.
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`.
It is important to note that `string_equal` requires the request to carry the Content-Type header, and the value of Content-Type must exactly match the specified value. In practice, you need to **explicitly specify the Content-Type header in your request**. Otherwise, if your request does not carry the Content-Type header, the request will fail. Additionally, when using certain tools to initiate requests without explicitly specifying the Content-Type, the tool may automatically add an unexpected Content-Type header, which could also result in request failure.
In addition, it is recommended to use case-insensitive conditional operators `string_equal_ignore_case` and `string_not_equal_ignore_case`. The reason is: if you use `string_equal` and `string_not_equal`, when the target is to forbid file uploads of type `text/html`, it cannot strictly forbid Content-Type settings such as `text/Html` or `tExt/html`. Using case-insensitive operators ensures strict prohibition. For more information about conditional operators, see Conditional Operators.

```
{
    "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_ignore_case":{
                    "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_ignore_case_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 interface supports adding the request parameter `response-content-type` to set 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: restricting the GetObject request parameter response-content-type to be "image/jpeg"

Assuming the primary account (uin:100000000001) owns the bucket `examplebucket-1250000000`, the following bucket policy restricts the sub-user (uin:100000000002) to only perform `Get Object` requests with the `response-content-type` parameter set to "image/jpeg". Since `response-content-type` is a request parameter, it must be URL-encoded when making the request, i.e., `response-content-type=image%2Fjpeg`. Therefore, when setting the policy, "image/jpeg" should also be URL-encoded as "image%2Fjpeg".

```
{
    "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: requiring PutObject requests to set the storage class to STANDARD

Assuming the primary account (uin:100000000001) owns the bucket examplebucket-1250000000, a bucket policy can be used to restrict the sub-account (uin:100000000002) to only allow PutObject requests with the x-cos-storage-class header and a value of `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

By using the request header `x-cos-acl`, you can specify Access Control Lists (ACLs) when uploading objects or creating buckets, as well as modify object or bucket ACLs. For more information on ACLs, refer to the ACL Overview.

- 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: The object ACL must be set to private in a PutObject request

Assuming the primary account (uin:100000000001) owns the bucket examplebucket-1250000000 and needs to restrict the sub-user (uin:100000000002) to only upload private objects. The following policy requires the PutObject request to carry the x-cos-acl header, and the header value must be `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 condition key `cos:prefix` to restrict the request parameter prefix.

> **Note:**
> If the value of `prefix` contains special characters (such as Chinese characters or `/`), it must be URL-encoded before being added to the bucket policy.

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

Assume that the primary account (uin:100000000001) owns the storage bucket examplebucket-1250000000 and needs to restrict the sub-user (uin:100000000002) to only listing objects within the folder1 directory of the bucket. The following bucket policy stipulates that when the sub-user initiates a GetBucket request, it must include the prefix parameter with the value `folder1/`. Since the prefix value contains the special character `/`, it must be URL-encoded before being written into the bucket policy. Consequently, the policy syntax is described as `folder1%2F`.

```
{
    "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%2F"
                }
            }
        },
        {
            "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_not_equal_if_exist":{
                    "cos:prefix":"folder1%2F"
                }
            }
        }
    ],
    "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 condition key `request_tag` is only applicable to `PutBucket` and `PutBucketTagging` operations. It is not supported for operations such as `GetService`, `PutObject`, and `PutObjectTagging`.

### Condition key qcs:request_tag

You can use the condition key `qcs:request_tag` to require users to include specified bucket tags when initiating PutBucket and PutBucketTagging requests.

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

Many users manage buckets using bucket tags. The following policy example restricts users to only be granted authorization when creating a bucket with the specified bucket tags `<a,b>` and `<c,d>`.
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 `x-cos-tagging: a=b&c=d` | 200, successful |
| PutBucket, request header `x-cos-tagging: a=b` | 200, successful |
| PutBucket, request header `x-cos-tagging: a=b&c=d&e=f` | 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 `x-cos-tagging: a=b&c=d` | 200, successful |
| PutBucket, request header `x-cos-tagging: a=b` | 200, successful |
| PutBucket, request header `x-cos-tagging: a=b&c=d&e=f` | 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"
                    ]
                }
            }
        }
    ]
}
```

## Forcing the Request Header (cos:x-cos-forbid-overwrite) to Prevent File Overwrite When Uploading Files

### Condition Key cos:x-cos-forbid-overwrite

The condition key cos:x-cos-forbid-overwrite can limit upload requests (PutObject, PutObject-Copy, InitiateMultipartUpload, CompleteMultipartUpload) to must carry the request header x-cos-forbid-overwrite, furthermore strictly forbidding users from triggering requests that may overwrite original objects.

### Upload File Request Must Specify x-cos-forbid-overwrite As true

Assuming the root account (uin:100000000001) owns the bucket examplebucket-1250000000 and needs to limit the Sub-user (uin:100000000002) from overwriting existing objects with the same name during object upload. The following policy restricts the Sub-user to must carry the `x-cos-forbid-overwrite` header with the value `true` when initiating upload requests (PutObject, PutObject-Copy, InitiateMultipartUpload, CompleteMultipartUpload).

```
{
    "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-forbid-overwrite": "true"
                }
            }
        },
        {
            "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-forbid-overwrite": "true"
                }
            }
        }
    ]
}
```

## Restricting Request Access Domain (cos:host)

### Condition Key Cos:Host

You can use the condition key cos:host to limit the Host header in user requests, thereby restricting user access to domain names.

### Example 1: Forbid Users From Accessing COS through Specified Domains

The following policy means users are forbidden from accessing COS through the default domain name examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com, but they can access COS using other domain names.

```
{
    "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": {
                "string_equal": {
                    "cos:host": "examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com"
                }
            }
        },
        {
            "principal": {
                "qcs": [
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect": "allow",
            "action": [
                "*"
            ],
            "resource": [
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition": {
                "string_not_equal": {
                    "cos:host": "examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com"
                }
            }
        }
    ]
}
```

### Example 2: Restricting Users to Download Objects Only Via Custom Domain Names

The following policy means users are only able to download objects (GetObject) from the bucket directory folder1 via the custom domain name mydomain1.com.

```
{
    "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/folder1/*"
            ],
            "condition": {
                "string_equal": {
                    "cos:host": "mydomain1.com"
                }
            }
        },
        {
            "principal": {
                "qcs": [
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect": "deny",
            "action": [
                "name/cos:GetObject"
            ],
            "resource": [
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/folder1/*"
            ],
            "condition": {
                "string_not_equal": {
                    "cos:host": "mydomain1.com"
                }
            }
        }
    ]
}
```

## Limit Object Lock Mode (cos:object-lock-mode)

### Condition Key cos:object-lock-mode

You can use the condition key `cos:object-lock-mode` to limit user uploads to objects that must use object lock with a fixed mode.

### Example: Authorizing Users to Set COMPLIANCE Mode Only

```
{
  "statement": [
    {
      "action": [
        "name/cos:PutObject",
        "name/cos:InitiateMultipartUpload",
        "name/cos:PutObjectRetention"
      ],
      "effect": "allow",
      "principal": {
        "qcs": [
          "qcs::cam::uin/1250000000:uin/1250000001"
        ]
      },
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:bjtest-1250000000/*"
      ],
      "condition": {
        "string_equal": {
          "cos:object-lock-mode": "COMPLIANCE"
        }
```

```
      }
    }
  ],
  "version": "2.0"
}
```

## Limiting Object Lock Retention Days (cos:object-lock-remaining-retention-days)

### Condition Key cos:object-lock-remaining-retention-days

You can use the condition key `cos:object-lock-remaining-retention-days` to limit user uploads to objects that must use object lock with the number of days set.

#### Retention Days Verification Principle

The value passed in for this condition key must be an integer (assumed as A). Let the timestamp of retain-until-date in the actual request be ts1 (in seconds), and the current system time timestamp be ts2 (in seconds). The conversion to retention days is (assumed as B).

```
Retention days (B) = round down[(ts1 - ts2)/(3600*24)]
```

Example 1: If retain-until-date is 2022-11-17T10:10:11 and the current time is 2022-11-15T09:00:00, the retention days (B) is 2.
Example 2: If retain-until-date is 2022-11-17T10:10:11 and the current time is 2022-11-15T12:00:00, the retention days (B) is 1.
Whether the conditions are met depends on comparing the size of values A and B.

### Example: PutObject Lock Retention Days Must Be Greater Than N Days

For example, the remaining days of the lock must be greater than 3 (excluding 3).

> **Note:**
> The remaining days specified by the condition key must be more than 3 days, or at least 4 days. If the request time is 16:00:00 on October 1, 2022, the RetainUntilDate set in the user request must be after 16:00:00 on October 5, 2022.

```
{
  "statement": [
    {
      "action": [
        "name/cos:PutObject",
        "name/cos:InitiateMultipartUpload",
        "name/cos:PutObjectRetention"
      ],
      "effect": "allow",
      "principal": {
        "qcs": [
          "qcs::cam::uin/1250000000:uin/1250000001"
        ]
      },
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:bjtest-1250000000/*"
      ],
      "condition": {
        "numeric_greater_than": {
          "cos:object-lock-remaining-retention-days": 3
        }
      }
    }
  ],
  "version": "2.0"
}
```

Use conditional operators. A valid RetainUntilDate is as follows:

| Condition Key | Description | Request Current Time | Input Parameter: Valid Time for Retain-Until-Date | Remarks |
| --- | --- | --- | --- | --- |
| "numeric_equal": { "cos: x-cos-object-lock-remaining-retention-days": 3 } | equal to 3 days | 2022-11-01T12:00:00Z | [ 2022-11-04T12:00:00Z, 2022-11-05T11:59:59Z ] | closed interval |
| "numeric_greater_than": { "cos: x-cos-object-lock-remaining-retention-days": 3 } | more than 3 days (excluding 3 days) | 2022-11-01T12:00:00Z | [ 2022-11-05T12:00:00Z, later ] | closed interval |
| "numeric_less_than": { "cos: x-cos-object-lock-remaining-retention-days": 3 } | less than 3 days (excluding 3 days) | 2022-11-01T12:00:00Z | [ 2022-11-01T12:00:01Z, 2022-11-04T11:59:59Z ] | closed interval |

## Restrict Access Based on Object Lock Retain until Date (cos:object-lock-retain-until-date)

### Condition Key cos:object-lock-retain-until-date

You can use the condition key `cos:object-lock-retain-until-date` to limit user uploads to objects that must use object lock with a specified date, supporting a minimum setting precision of whole seconds.

### Example: Restricting the Specified Lock Date for PutObject Uploads

The request indicates that RetainUntilDate must be after `2022-11-11T12:00:00Z`.

```
{
  "statement": [
    {
      "action": [
        "name/cos:PutObject",
        "name/cos:InitiateMultipartUpload",
        "name/cos:PutObjectRetention"
      ],
      "effect": "allow",
      "principal": {
        "qcs": [
          "qcs::cam::uin/1250000000:uin/1250000001"
        ]
      },
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:bjtest-1250000000/*"
      ],
      "condition": {
        "date_greater_than": {
          "cos:object-lock-retain-until-date": "2022-11-11T12:00:00Z"
        }
      }
    }
  ],
  "version": "2.0"
}
```

## Restricting ACL Authorization Header in Request

### Condition Key

Request headers x-cos-grant-full-control, x-cos-grant-read, x-cos-grant-write, x-cos-grant-read-acp, and x-cos-grant-write-acp are used in requests for object upload, object replication, and object ACL modification to specify ACL permission information. For details, see PutObject request headers. As shown in the table below, corresponding condition keys can respectively limit whether to allow carrying these headers or restrict the content of corresponding headers.

| Condition Key | Corresponding Request Header |
| --- | --- |
| cos:x-cos-grant-full-control | x-cos-grant-full-control |
| cos:x-cos-grant-read | x-cos-grant-read |
| cos:x-cos-grant-write | x-cos-grant-write |
| cos:x-cos-grant-read-acp | x-cos-grant-read-acp |
| cos:x-cos-grant-write-acp | x-cos-grant-write-acp |

The following example uses cos:x-cos-grant-full-control to demonstrate how to limit the usage of ACL authorization headers. Other condition keys follow a similar method. These condition keys have two main usage scenarios:

- Limit this header to account authorization only. See example 1.
- Restrict the use of this header for authorization to prevent users from tampering with the object's ACL permissions by leveraging PutObject permissions. See example 2.

### Example 1: Limiting Accounts Authorized by X-Cos-Grant-Full-Control

The effect of the following policy is: Grant the subaccount permission to upload objects, but the user-uploaded objects must carry the x-cos-grant-full-control header, and the authorized account must be the root account 100000000001. The x-cos-grant-full-control header contains the `"` symbol, which should be passed as a string literal in the policy. Note that it needs to be escaped as `\"`.

```
{
    "statement": [{
            "action": [
                "name/cos:PutObject",
                "name/cos:PostObject",
                "name/cos:AppendObject",
                "name/cos:InitiateMultipartUpload"
            ],
            "effect": "allow",
            "principal": {
                "qcs": [
                    "qcs::cam::uin/1250000000:uin/1250000001"
                ]
            },
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:bjtest-1250000000/*"
            ],
            "condition": {
                "string_equal": {
                    "cos:x-cos-grant-full-control": "id=\"100000000001\""
                }
            }
        },
        {
            "action": [
                "name/cos:PutObject",
                "name/cos:PostObject",
                "name/cos:AppendObject",
                "name/cos:InitiateMultipartUpload"
            ],
            "effect": "deny",
            "principal": {
```

```
                "qcs": [
                    "qcs::cam::uin/1250000000:uin/1250000001"
                ]
            },
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:bjtest-1250000000/*"
            ],
            "condition": {
                "string_not_equal_if_exist": {
                    "cos:x-cos-grant-full-control": "id=\"100000000001\""
                }
            }
        }
    ],
    "version": "2.0"
}
```

### Example 2: Disallow Authorization Via x-cos-grant-full-control

The effect of the following policy is: Grant the subaccount permission to upload objects, but forbid users from carrying the x-cos-grant-full-control header or only allow this header to be set as empty.

```
{
    "statement": [{
            "action": [
                "name/cos:PutObject",
                "name/cos:PostObject",
                "name/cos:AppendObject",
                "name/cos:InitiateMultipartUpload"
            ],
            "effect": "allow",
            "principal": {
                "qcs": [
                    "qcs::cam::uin/1250000000:uin/1250000001"
                ]
            },
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:bjtest-1250000000/*"
            ],
            "condition": {
                "string_equal_if_exist": {
                    "cos:x-cos-grant-full-control": ""
                }
            }
        },
        {
            "action": [
                "name/cos:PutObject",
                "name/cos:PostObject",
                "name/cos:AppendObject",
                "name/cos:InitiateMultipartUpload"
            ],
            "effect": "deny",
            "principal": {
                "qcs": [
                    "qcs::cam::uin/1250000000:uin/1250000001"
                ]
            },
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:bjtest-1250000000/*"
            ],
            "condition": {
```

```
                "string_not_equal": {
                    "cos:x-cos-grant-full-control": ""
                }
            }
        }
    ],
    "version": "2.0"
}
```

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

Last updated: 2024-06-03 11:28:55

## Scenario

The current primary account A (APPID 1250000000) has two storage buckets: `examplebucket1-1250000000` and `examplebucket2-1250000000`. There is another primary account B and its sub-account B0. Due to business requirements, B0 wishes to operate the two storage buckets under account A. The following will introduce how to perform the relevant authorization operations.

## Instructions

### 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. Choose **Create Custom Policy > Create by Policy Syntax**, then select a 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**: Define a unique and meaningful policy name, 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** to finish creating the policy.
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: 2025-04-23 14:11:28

> **Note:**
> COS has already achieved high QPS through its underlying index dispersion mechanism. If you require higher QPS performance, please contact us. In daily file organization, we still recommend that you follow the guidelines in this document to avoid overly concentrated index storage methods.

## Feature 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:**
> Request load refers to the number of requests initiated per second, not the number of concurrent connections. This means that you can still send hundreds of new connection requests within 1 second while maintaining thousands of 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/GET requests per second, contact us to prepare for the workload and avoid exceeding the request limit.

> **Note:**
> If your mixed request workload only occasionally reaches 30,000 requests per second and does not exceed 30,000 requests per second during bursts, you may not need to follow this guide.

## Practical Steps

### 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.gz
logs/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.

# Data Migration
# Migrating Local Data to COS

Last updated: 2023-09-12 17:38:35

## Practice Scenario

For users with local IDCs, Cloud Object Storage (COS) supports the following migration methods across various migration types, assisting users in quickly transferring massive amounts of data from their local IDCs to COS.

| Migration method | Note |
| --- | --- |
| COS Migration (Online Migration) | COS Migration is an all-in-one tool integrating COS data migration features. You can migrate data to COS quickly after simple configurations. |
| Cloud Data Migration (CDM) (Offline Migration) | Cloud Data Migration (CDM) is a migration method that utilizes Tencent Cloud's dedicated offline migration devices to help users transfer their local data to the cloud. This approach addresses the issues of lengthy transfer times, high costs, and low security associated with transferring data from local data centers to the cloud via network transmission. |

Users can consider how to choose a migration method based on factors such as data migration volume, IDC outbound bandwidth, IDC available space resources, and acceptable migration completion time. The following figure shows the estimated time consumption when using online migration. As can be seen, if the migration cycle exceeds 10 days or the migration data volume is more than 50 TB, we recommend you choose CDM for offline migration. Otherwise, please opt for online migration.

| Bandwidth / Data volume | 10Mbps | 100Mbps | 1Gbps | 10Gbps |
| --- | --- | --- | --- | --- |
| 10GB | 3 hours | 17 minutes | 2 minutes | 10 seconds |
| 100GB | 30 hours | 3 hours | 17 minutes | 2 minutes |
| 10TB | 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 |
| 10PB | 35 years | 3.5 years | 125 days | 12.5 days |
| 10PB | 350 years | 35 years | 3.5 years | 125 days |

**Note**
A large number of small files under 1MB and insufficient disk I/O performance can also affect the data migration progress.

## Migration Practices

### COS Migration

The steps are as follows:
1. Install the Java environment.
2. Install the COS Migration Tool.
3. Modify the configuration file.
4. Launch the tool.

For more information, please see COS Migration Tool.

#### Tips

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

1. Adjust the threshold for distinguishing between large and small files and the migration concurrency according to your network environment, achieving the optimal migration method with large file segmentation and small file concurrent transmission. Modify the tool execution time and set bandwidth limits to ensure that your business operations are not affected by the bandwidth occupied by migration data. The above adjustments can be made in the `[common]` section of the config.ini configuration file by modifying the following parameters:

| Parameter name | Description |
| --- | --- |
| smallFileThreshold | 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. |
| 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 machines and performing separate migration tasks for different sources.

## Cloud Data Migration (CDM)

The steps are as follows:

1. Go to the CDM Console to submit an application.
2. After the application is approved, you can wait to receive the device.
3. After receiving the device, copy the data to it according to the migration device manual.
4. After completing the data copy, submit a return request through the console and wait for Tencent Cloud to migrate the data to COS.

For more information, please see Cloud Data Migration product documentation.

### Tips

The following section explains how to efficiently and securely migrate data to Tencent Cloud COS using offline migration.

1. Configure a 10 Gbps network connection for your IDC. To remove potential constraints of the local data environment on data transfer, you can use a high-performance server as a mount point to maximize the replication speed.
2. Parallel transfer is the fastest way to transfer data in CDM. The CPU and memory usage of the device are monitored. If the current migration speed is lower than expected, you can select the following methods:
   - Multiple devices connect to the same CDM device through different network interfaces.
   - Multiple devices connect to several CDM devices through different network interfaces.

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

Last updated: 2023-09-12 17:38:49

## 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 | Concurrence | HTTPS Secure Transfer |
| --- | --- | --- | --- | --- |
| Migration Service Platform (MSP) | 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
- Huawei Cloud OBS Migration
- Migrating from Qiniu KODO
- Migrating from UCLOUD UFile
- Migrating from Kingsoft Cloud KS3
- Migrating from Baidu Cloud BOS
- AWS S3 Migration Tutorial

#### Tips

During the data migration process, the read speed of the data source may vary due to different network environments. However, selecting a higher QPS concurrency when creating a new file migration task, based on the actual situation, can help improve the migration speed.

# Migrating Data from URL to COS

Last updated: 2023-09-12 17:39:05

## Background

For users who want to use a URL list as the data source for data migration, Cloud Object Storage (COS) supports the following migration methods:

| Migration method | Note |
| --- | --- |
| Migration Service Platform (MSP) | Migration Service Platform (MSP) is an integrated platform with various migration tools and a visual interface, which helps users easily monitor and manage large-scale data migration tasks. The "File Migration Tool" within MSP assists users in migrating data from various public clouds and data source sites to COS. |
| COS Origin Pull | COS origin-pull automatically migrates data with read and write access requests from the data source to Tencent Cloud COS. This migration method not only helps users quickly perform data tiering based on access frequency but also accelerates the read and write access speed of hot data in the business system. |
| COS Migration | COS Migration is an all-in-one tool integrating COS data migration features. You can migrate data to COS quickly after 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.

> ⚠ **Note**
> Currently, COS does not support migrating data from URLs containing authentication information.

## Migration Practices

### MSP

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, please see Migrating from a URL List.

#### Tips

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.

### COS origin-pull

The steps are as follows:

1. Log in to the COS console and enable origin-pull settings in the target bucket for the data to be migrated.
2. Configure the bucket origin-pull address and save the configuration.
3. Redirect read and write requests from the business system to Tencent Cloud COS.

For more information, please see Setting Origin-Pull.

#### Tips

The following steps enable the separation of hot and cold source data, seamlessly migrating hot data to Tencent Cloud COS to accelerate the read request speed of hot data.

1. Switch the read and write requests of the business system to COS, and enable the origin-pull setting in the COS console, with the origin address set to the data source site. The system structure is shown in the following diagram:


2. After a period of time, the cold data remains at the source, while the hot data has been migrated to Tencent Cloud COS. The migration process does not affect the business system.

## COS Migration

The steps are as follows:

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

For more information, please see COS Migration Tool.

### Tips

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

1. Adjust the threshold for distinguishing between large and small files and the migration concurrency according to your network environment, achieving the optimal migration method with large file segmentation and small file concurrent transmission. Modify the tool execution time and set bandwidth limits to ensure that your business operations are not affected by the bandwidth occupied by migration data. The above adjustments can be made in the `[common]` section of the config.ini configuration file by modifying the following parameters:

| Parameter name | Description |
| --- | --- |
| smallFileThreshold | 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. |
| 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 machines and performing separate migration tasks for different sources.

# Migrating Data Within COS

Last updated: 2023-09-12 17:40:26

## Background

For users utilizing Tencent Cloud Object Storage (COS), if you need to migrate data from one bucket to another within the same COS service, we recommend using the following migration methods:

- Online Migration: 1. Migration Service Platform (MSP)
- Online migration: Cross-bucket replication

## Migration Practices

### Migration Service Platform

MSP is a platform that integrates various migration tools and provides a visual interface, helping users easily monitor and manage large-scale data migration tasks. The "File Migration Tool" within MSP assists users in migrating data from various public clouds and data source sites 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, 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

Bucket replication is a configuration feature provided by Tencent Cloud COS for buckets. By setting up replication rules, you can automatically and asynchronously copy **incremental objects** between buckets in different regions. Once bucket replication is enabled, COS will precisely replicate the object content (such as object metadata and version ID) from the source bucket to the target bucket, ensuring that the replicated object copies have completely consistent attributes.

Additionally, operations performed on objects in the source bucket, such as uploading or deleting objects, will also be replicated to the target bucket. For more information and instructions, please refer to Bucket Replication.

# Migrating Data Between HDFS and COS

Last updated: 2023-09-12 17:40:43

## Feature 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 Hadoop file system semantics, Hadoop DistCp can be conveniently used for bidirectional data migration between Tencent Cloud Object Storage (COS) and other Hadoop file systems. This article takes HDFS as an example and demonstrates how to use Hadoop DistCp to perform data migration between Hadoop file systems and COS.

## Preparations

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 file list in 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 permissions to the destination path in 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
> - To use custom monitoring capabilities, you need to grant Tencent Cloud Observability Platform the permissions to report and read metrics. Please grant QcloudMonitorFullAccess cautiously or grant Tencent Cloud Observability Platform Interface permissions as needed.

## Practical Steps

### Copy data from HDFS to COS bucket

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

```
[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 launches a MapReduce job to execute the file copy task, and upon completion, it outputs a simple report, as shown below:

```
                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.CopyCommitter: Cleaning up temporary work folder: file:/tmp/hadoop-iainyu/mapred/staging/iainyu1750342510/.staging/_distcp-1385927222
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. Execute the command `hadoop fs -ls -R cosn://hdfs-test-1250000000/` to list the directories and files that have been migrated to the storage 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-1
-rw-rw-rw-   1 iainyu iainyu     167225 2019-12-13 21:03 cosn://hdfs-test-125       /test/test-2
-rw-rw-rw-   1 iainyu iainyu     167225 2019-12-13 21:03 cosn://hdfs-test-125       /test/test-3
```

## Copy files from buckets in COS 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**
> This command-line configuration supports bidirectional operations, allowing data migration from HDFS to COS and vice versa.

Run the following command:

```
hadoop distcp -Dfs.cosn.impl=org.apache.hadoop.fs.CosFileSystem -Dfs.cosn.bucket.region=ap-XXX  -
Dfs.cosn.userinfo.secretId=AK**XXX  -Dfs.cosn.userinfo.secretKey=XXXX  -libjars /home/hadoop/hadoop-cos-
2.6.5-shaded.jar  cosn://bucketname-appid/test/ hdfs:///test/
```

The parameters are described as follows:

- 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; this value can be accessed from the API Key Management page.
- Dfs.cosn.userinfo.secretKey: Enter the secretKey of the bucket owner’s account; this value can be accessed from the API Key Management page.
- libjars: Specifies the location of the Hadoop-COS JAR package. The Hadoop-COS JAR package can be downloaded from the dep directory in the Github repository.

> **Note**
> For other parameters, please refer to the Hadoop Tools documentation.

## 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 14:27:03

## Feature Overview

COS offers AWS S3-compatible APIs. Therefore, after your data is migrated from S3 to COS, you can easily make your client application 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.

### Prerequisites

- 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.

### Initialize

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() {
         // Request STS here to obtain temporary key information
        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 AWSCredentialsProvider protocol.

```
-(AWSTask<AWSCredentials *> *)credentials{
    // Request STS here to obtain temporary key information
    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:@"http://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 defaultServiceManager] setDefaultServiceConfiguration:configuration];
```

## Node.js

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

### Initialize

When initializing an instance, set the Tencent Cloud SecretKey and Endpoint. Taking the bucket region `ap-guangzhou` as an example, the code snippet 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 AWS Configuration and Certificate Files

> **Note**
> The following example demonstrates how to modify the AWS configuration and certificate files on a Linux system.

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 AWS Configuration and Certificate Files

> **Note**
> The following example demonstrates how to modify the AWS configuration and certificate files on a Linux system.

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 AWS Configuration and Certificate Files

> **Note**
> The following example demonstrates how to modify the AWS configuration and certificate files on a Linux system.

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.

### Initialize

When initializing an instance, set the Tencent Cloud SecretKey and Endpoint. For example, if the bucket is located in the `ap-guangzhou` region:

```
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. Initiate a server 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.Getenv("SECRETKEY"), "")
// Key
        customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string, options
...interface{}) (aws.Endpoint, error) {
                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 AWS Configuration and Certificate Files

> **Note**
> The following example demonstrates how to modify the AWS configuration and certificate files on a Linux system.

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);
```

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

Last updated: 2023-09-12 17:43:54

## Feature Overview

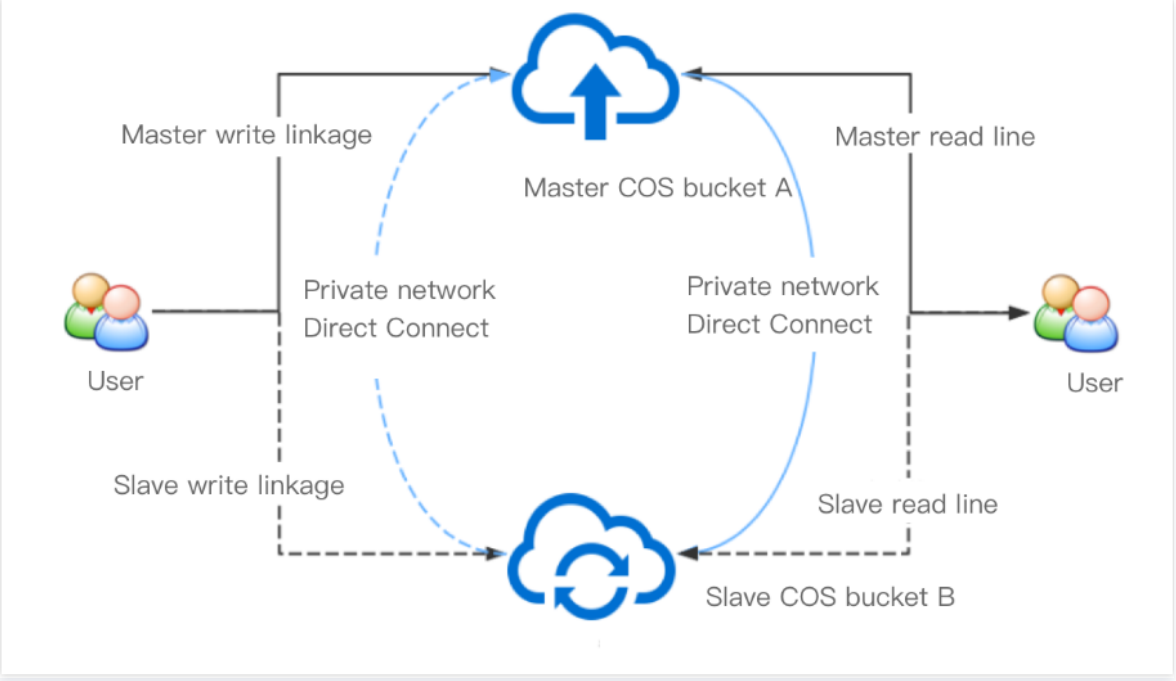
Tencent Cloud Object Storage (COS) offers customers a 99.95% availability and 99.999999999% reliability. However, due to uncontrollable factors such as natural disasters and fiber optic failures, the availability and reliability of cloud data cannot reach 100%. Additionally, certain industries, such as the financial sector, require high availability and reliability due to the unique nature of their business operations.
To ensure business continuity and stability, and to meet the demands for high availability and reliability, Tencent Cloud COS offers a data disaster recovery and high availability solution based on bucket replication. We recommend that enterprises implement disaster recovery and backup measures for their cloud data according to their business needs, ensuring stable and continuous operation.
This document primarily covers two aspects: firstly, it introduces a disaster recovery solution for cloud-based business master-slave switching based on bucket replication; secondly, it further elaborates on a high availability solution also based on bucket replication. The high availability solution is achieved through a combination of products and features such as bucket replication, origin-pull, Serverless Cloud Function (SCF), and Content Delivery Network (CDN).

## 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 disaster recovery backup architecture based on cross-bucket replication is as follows:




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.

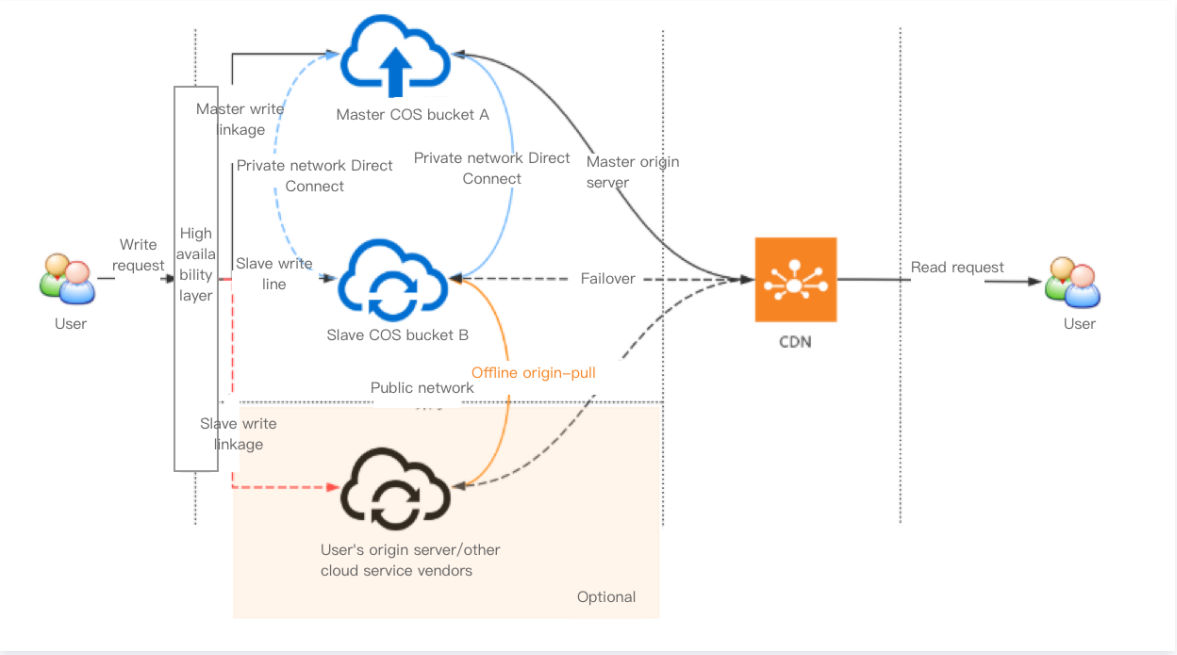
> **Note**
> After the incremental data in Bucket A is replicated to Bucket B, although it becomes incremental data in Bucket B, it will not be replicated back to Bucket A.

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 detection can be implemented using Tencent Cloud SCF, as demonstrated in Scheduled Detection and Email Alerting. By modifying the SCF code, you can change the detection address to the domain names of the primary and secondary buckets. Additionally, you can modify the alert code snippet according to your business requirements to implement the necessary measures for other services.

## 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 bucket replication is as follows:




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**: Generally composed of COS buckets in different regions; users can also ensure data consistency by setting origin-pull policies and **incorporating their own origin servers or buckets from other cloud providers**.
- **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 whiling protecting the security of the data on the origin servers.

## Reference

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

- Versioning Overview
- Cross-Bucket Replication Overview
- Scheduled monitoring and alarm notifications via email
- Setting Origin-Pull
- Setting CDN Acceleration
- Origin Server Configuration
- Connected Domain

# Cloud Data Backup

Last updated: 2023-09-12 17:44:11

## Feature Overview

This article provides an overview of the various methods available for backing up data stored in the cloud. Tencent Cloud Object Storage (COS) offers three convenient options for users to back up their data stored in COS:

- Backup and Disaster Recovery Solution Based on Cross-region Replication
- Batch Data Copy Solution (COS Batch Processing Feature)
- Migration Backup Solution Based on Migration Service Platform (MSP) Tool

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

Cross-region replication is a configuration feature for storage buckets. By setting up cross-region replication rules, you can automatically and asynchronously copy **incremental objects** between storage buckets in different regions. Once cross-region replication is enabled, COS will precisely replicate the object content (such as object metadata and version ID) from the source bucket to the target bucket, ensuring that the replicated object copies have completely consistent attribute information.
For more details, please refer to: Disaster Recovery and High Availability Architecture Based on Cross-region Replication.

### Scenarios

- **Remote disaster recovery:** COS boasts 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.

### How to Use

Please refer to the Setting Cross-region Replication console documentation.

## Batch Data Copy Solution

COS Batch Processing feature offers users large-scale, object-level batch copy operations.

### Scenarios

Due to certain business reasons, you may need to back up all data from an existing storage bucket to another storage bucket to ensure data availability and reliability.

### How to Use

Please refer to the Batch Processing console documentation.

### Notes

Batch object copy operations require the use of the inventory feature. For more information about the inventory, see Inventory Feature Overview. You can batch copy specified objects from the source bucket to the target bucket, which can be in the same or different regions. Batch object copy operations support custom configurations related to PUT Object - copy parameters, which can affect the metadata or storage type of the replica. For more information about PUT Object-copy, please refer to PUT Object - copy.

> ⚠ Note

- All objects to be copied must be in the same bucket.
- Only one destination bucket can be configured for a batch copy 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 with the same name as an object to be copied, COS will overwrite the object.

## Data Migration Backup Solution

Tencent Cloud Migration Service Platform (MSP) offers users a convenient and efficient resource migration solution. After creating a migration task, users can view the migration progress, reports, and other information through the migration service console. For detailed instructions, please refer to: Tencent Cloud COS Inter-Migration.

# Local Data Backup

Last updated: 2023-09-12 17:44:25

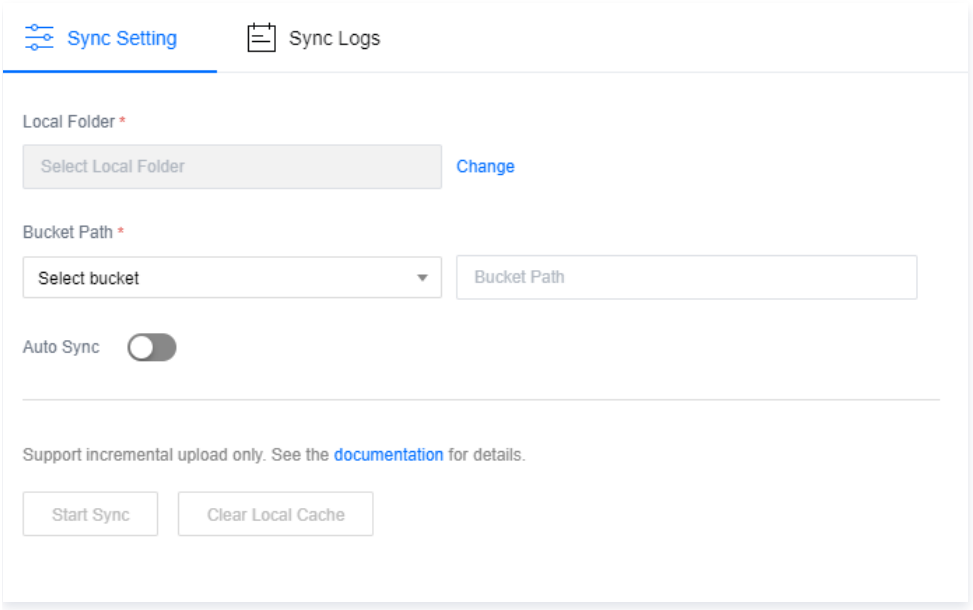
## Feature Overview

This article provides an overview of the various methods available for backing up local data to the cloud. Tencent Cloud Object Storage (COS) offers three backup options for users to conveniently store their local data in COS buckets:

- COSBrowser File Synchronization Backup
- COS Migration Online Migration Backup
- CDM Offline Migration Backup

## File Synchronization Backup

COSBrowser is a visual interface tool introduced by Tencent Cloud Object Storage for managing cloud-based files, allowing you to easily view, transfer, and manage COS resources with a user-friendly interaction. COSBrowser is available in both desktop and mobile versions. For more information, see COSBrowser Overview.
COSBrowser desktop integrates a file synchronization feature that allows you to associate local folders with COS buckets, enabling real-time synchronization of local files to the cloud.




### How to Use

Please refer to File Synchronization Usage Guide.

## Online Migration Solutions

COS Migration is an all-in-one tool integrating COS data migration features. You can migrate data to COS quickly after simple configurations.

### Scenarios

For users with on-premises IDCs, COS Migration helps to quickly transfer massive amounts of data from local IDCs to Tencent Cloud Object Storage (COS).

### How to Use

For more information, see COS Migration.

## Offline Migration Solution

CDM uses the dedicated offline migration device provided by Tencent Cloud to help you migrate local data to the cloud. It can solve the problems that the local IDC migrates to cloud through network: long time, high cost and low security.

Users can consider the migration method based on factors such as data migration volume, IDC outbound bandwidth, available IDC resources, and acceptable migration completion time. The following chart shows the estimated time consumption when using online migration. As can be seen, if the migration cycle exceeds 10 days or the migration data volume is more than 50 TB, we recommend choosing Cloud Data Migration (CDM) for offline migration.

| Data Volume \ Bandwidth | 10Mbps | 100Mbps | 1Gbps | 10Gbps |
|---|---|---|---|---|
| 10GB | 3 hours | 17 minutes | 2 minutes | 10 seconds |
| 100GB | 30 hous | 3 hours | 17 minutes | 2 minutes |
| 1TB | 12.5 days | 30 hous | 3 hours | 17 minutes |
| 10TB | 125 days | 12.5 days | 30 hous | 3 hours |
| 100TB | 3.5 years | 125 days | 12.5 days | 30 hous |
| 1PB | 35 years | 3.5 years | 125 days | 12.5 days |
| 10PB | 350 years | 35 years | 3.5 years | 125 days |

## How to Use

Please see Cloud Data Migration (CDM).

# Domain Name Management Practice Supporting HTTPS for Custom Endpoints

Last updated: 2024-07-16 16:41:02

## 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

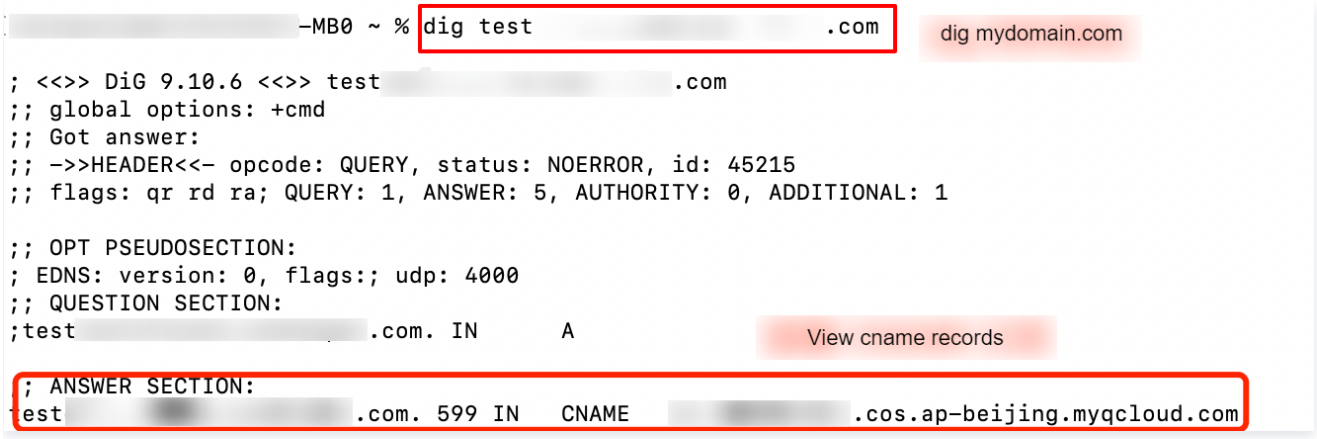
### Tencent Cloud Domain

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 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's custom domain.
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=token, Domain=domain,
Scheme=scheme)
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 语言 SDK
- 小程序SDK

# Supporting HTTPS for Custom Endpoints

Last updated: 2023-09-12 17:45:07

## Feature Overview

Users can access objects in a bucket using their own custom domain (e.g., `example.com`). Follow the steps below for detailed instructions:

- Supporting HTTPS for a custom endpoint with CDN acceleration enabled
- Supporting HTTPS for a custom endpoint with CDN acceleration disabled

## Instructions

### Enable 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 provides a step-by-step example of configuring a custom domain with HTTPS access through reverse proxy in Cloud Object Storage (COS) with CDN acceleration disabled. In this example, we will access the Guangzhou-based bucket named testhttps-1250000000 directly via the custom domain `https://example.com` without enabling CDN acceleration. Follow the steps below:

#### 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.
HTTPS certificate hosting currently is not supported in other regions. If you need to use HTTPS certificates, see Method 2.

- Method 1: Bind a custom origin domain name via the COS console
  Bind the bucket testhttps-1250000000 to the domain `https://example.com` with CDN acceleration disabled. For detailed instructions, please refer to the Enable Custom Origin Domain Name documentation.
- Method 2: Configure reverse proxy for the domain
  Set up a reverse proxy for the domain `https://example.com` on the server. Refer to the following configuration (the Nginx configuration provided is for reference only):

```
server {
  listen        443;
  server_name  example.com ;

  ssl on;
  ssl_certificate /usr/local/nginx/conf/server.crt;
  ssl_certificate_key /usr/local/nginx/conf/server.key;

  error_log logs/example.com.error_log;
  access_log logs/example.com.access_log;
  location / {
      root /data/www/;
      proxy_pass  http://testhttps-1250000000.cos.ap-guangzhou.myqcloud.com; // Configure the default
download domain for the bucket
  }
}
```

The `server.crt;` and `server.key` are the HTTPS certificates for your custom domain. If your domain does not have an HTTPS certificate, please visit the Tencent Cloud SSL Certificate page to apply for one.

If you do not have a certificate temporarily, you can remove the following configuration information. However, a warning will appear when accessing the site. Click "Continue" to proceed:

```
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

Configure your domain's DNS resolution at your DNS provider. If you are using Tencent Cloud DNS, go to the DNS Resolution Console and resolve the domain `example.com` to the server IP from step 1. For detailed instructions, refer to Quickly Add Domain Resolution.

### Step 3. Perform advanced configurations

- **Accessing web pages directly through a browser**
  After configuring your custom domain to support HTTPS, you can use your domain to download objects from the bucket. If you need to access web pages, images, etc., directly in the browser, you can use the static website feature. For detailed instructions, please refer to Setting up a Static Website.
  Once the configuration is complete, add a line of information to the Nginx configuration, restart Nginx, and refresh the browser cache.

```
proxy_set_header Host $http_host;
```

- **Configure Hotlink Protection**
  If the bucket is public, there is a risk of hotlinking. Users can prevent malicious hotlinking by setting up hotlink protection and enabling a Referer whitelist. Follow the steps below:

1. Log in to the COS console, enable the hotlink protection feature, and configure an allowlist. For detailed directions, please see Setting Hotlink Protection.
2. Add the following code to the Nginx configuration file, restart Nginx, and refresh the browser cache.

```
proxy_set_header   Referer www.test.com;
```

3. After completing the setup, opening the file directly will display an error message: `Error code: -46616`. Error description: Not in the referer allowlist. However, accessing the custom domain through a proxy will allow the webpage to load normally.

```
{
errorcode: -46616,
errormsg: "not hit white referer, retcode:-46616"
}
```

# Setting Cross-Origin Resource Sharing (CORS)

Last updated: 2023-09-20 11:34:01

## One-Origin Policy

The one-origin policy is a key security mechanism for isolating potentially malicious files. The policy restricts the way files/scripts loaded from one origin interacts with the resources from another origin. Resources with the same protocol, domain name (or IP), and port are considered to belong to the same origin. The scripts in one origin only have permissions to read/write resources in its origin, and cannot access resources from 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 |
| --- | --- | --- |
| `http://www.example.com/dir2/other.html` | Yes | Same protocol, domain name, and port |
| `http://www.example.com/dir/inner/another.html` | Yes | Same protocol, domain name, and port |
| `https://www.example.com/secure.html` | No | Different protocols (HTTPS) |
| `http://www.example.com:81/dir/etc.html` | No | Different ports (81) |
| `http://news.example.com/dir/other.html` | No | Different domain names |

### Cross-origin Access

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. 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 some header information. In some cases, an additional request is made, but the user will not perceive it.
Therefore, the key to CORS communication lies in the server. As long as the server implements the CORS APIs, cross-origin communication can be implemented.

## CORS Use Cases

CORS is used when you're 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 concern about cross-origin.
With CORS, you can use AJAX on browsers to directly access, upload, and download COS data without using your app server for data transfer. If your websites use both COS and AJAX, you are advised to 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. In other words, CORS rules of COS are only used to decide whether to add CORS-related headers. It's up to the browser whether to block the request.
All object and multipart APIs of COS support CORS authentication.

> **Note**
> When two webpages from `www.a.com` and `www.b.com` on the same browser request the same cross-origin resource, if the request from `www.a.com` reaches the server first, the server will attach the Access-Control-Allow-Origin header to the resource and return it to the user of `www.a.com`. When `www.b.com` initiates a request, the browser will return the cached response from the previous request to the user. If the header content does not match the CORS requirements, it will result in a failed request for `www.b.com`.

## CORS Configuration Example

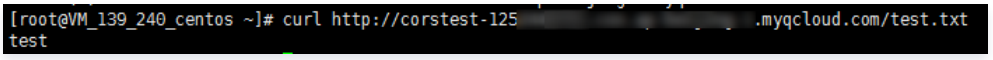
The following simple example demonstrates the configuration steps for using AJAX to fetch data from COS. The bucket permissions in the example are set to public, and for accessing private buckets, just attach a signature to the request, with the rest of the configuration remaining the same. In the example below, the bucket name is "corstest" and the access permissions are set to public read and private write.

### Prerequisites

1. **Verify that the file is accessible**
   Upload a text document named test.txt to corstest. The access address for test.txt is `http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt`.
   Use curl to directly access the text document, replacing the following address with your file address:

```
curl http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt
```

   The content of the test.txt file is returned: test. This indicates that the document can be accessed normally.

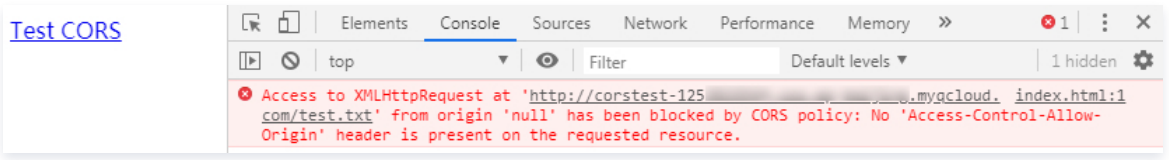



2. **Accessing files using AJAX**
   Let's attempt to access the test.txt file directly using AJAX technology.
   2.1 Create a simple HTML file by copying the following code and saving it as a local HTML file. Open the file in a browser. Since no custom headers are set, this request does not require preflight.

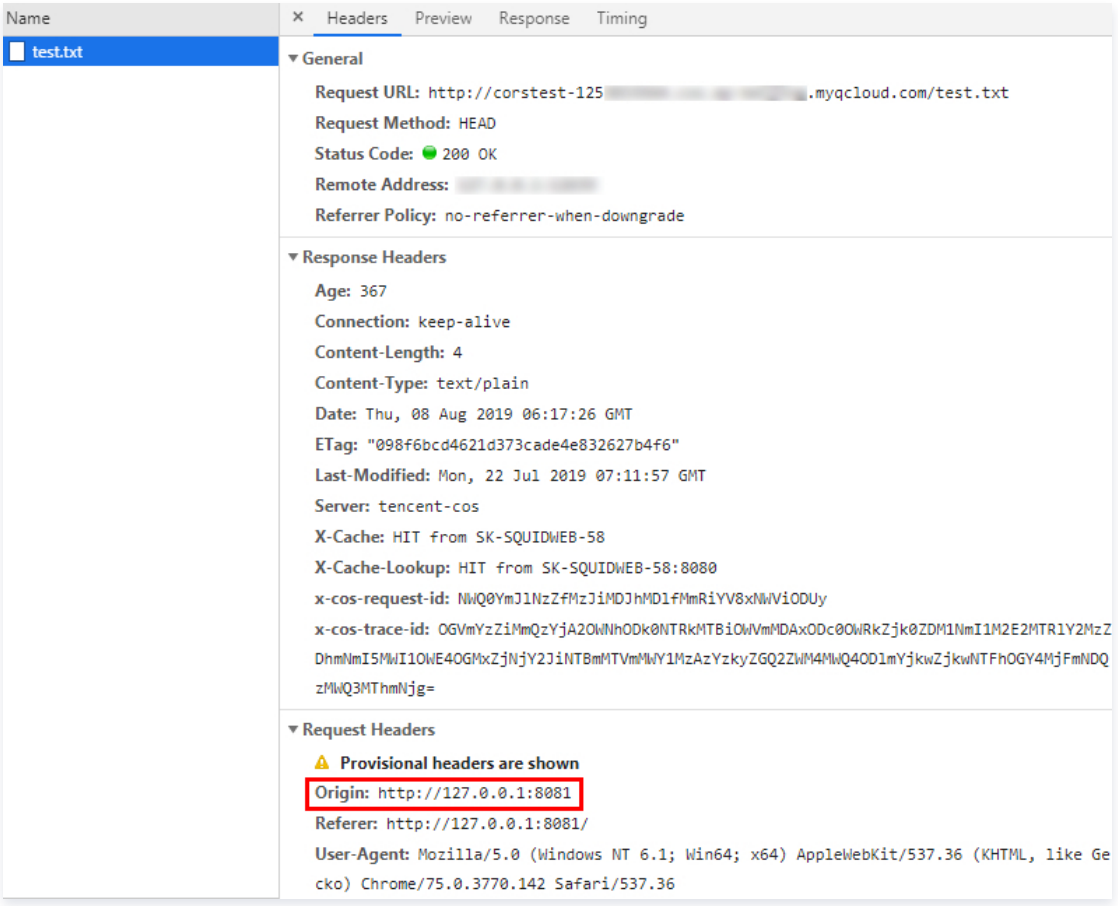
```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
</head>
<body>
<a href="javascript:test()">Test CORS</a>
<script>
function test() {
    var url = 'http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt';
    var xhr = new XMLHttpRequest();
    xhr.open('HEAD', url);
    xhr.onload = 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.');
    };
    xhr.send();
}
</script>
</body>
</html>
```

   2.2 Open the HTML file in a browser, and click on **Test CORS** to send a request. The following error occurs, indicating that access is denied because the Access-Control-Allow-Origin header is not found. Clearly, this is because the server has not configured

CORS.




2.3 Access failed. Upon further inspection of the header, it is evident that the browser sent a request with an Origin, indicating a cross-origin request.



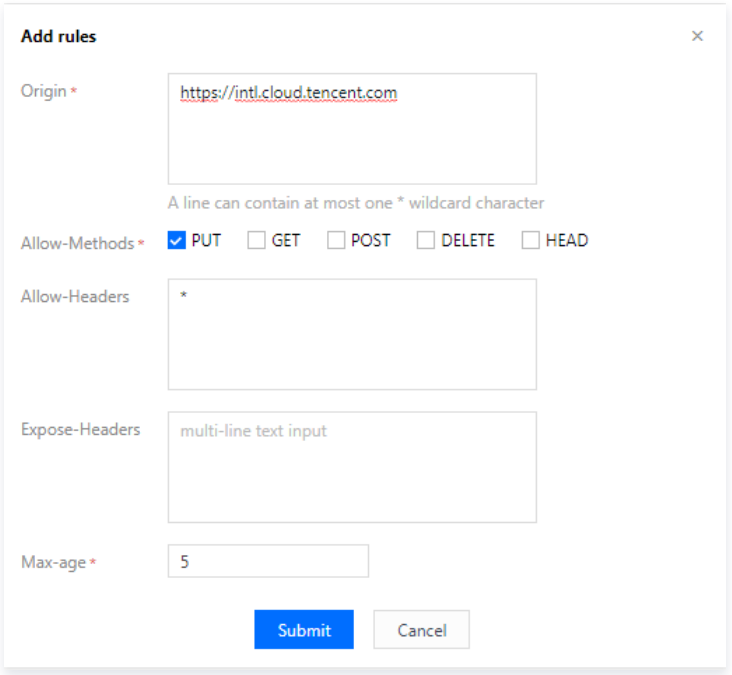

**Note**
You can set up the webpage on the server with the address `http://127.0.0.1:8081`, so the Origin is `http://127.0.0.1:8081`.

## Configuring CORS

Now that you have identified the cause of the access failure, you can solve the problem by configuring the bucket-related CORS. This example configures CORS in the COS console, which is recommended for uncomplicated configurations. You can configure as follows:

1. Log in to the COS Console, click on **Bucket List**, enter the relevant bucket, click on the **Security Management** tab, and scroll down to find the "Cross-Origin Resource Sharing (CORS) Settings."

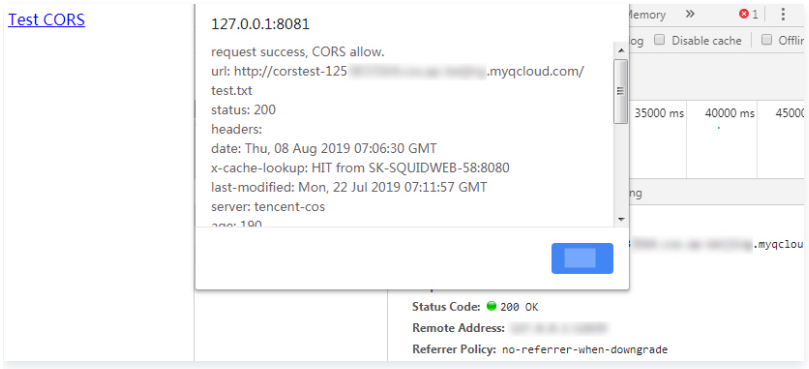
2. Click **Add Rule** to create the first rule with the most relaxed configuration, as shown below:




> **Note**
> CORS settings consist of a series of rules, which are matched one by one, starting from the first rule. The earliest matched rule takes precedence.

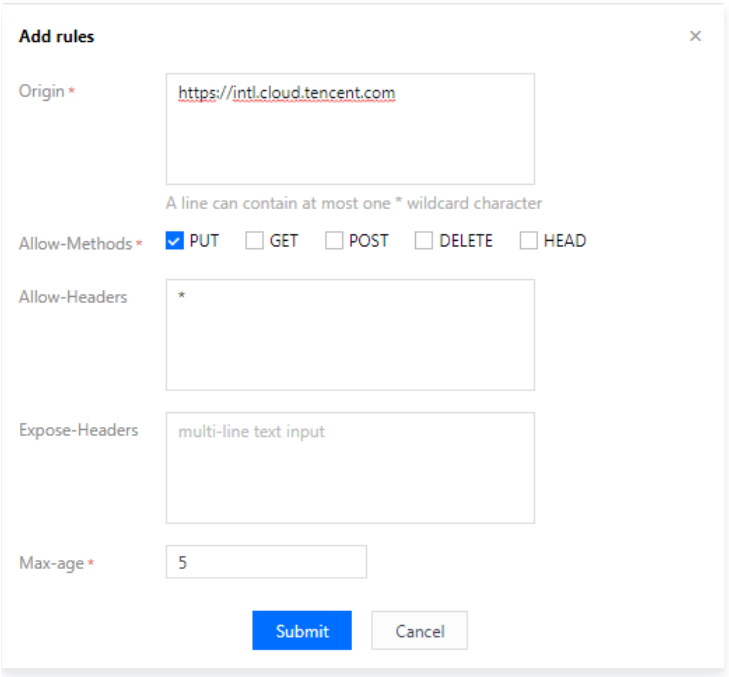
### Verifying the result

After completing the configuration, try accessing the test.txt file again. The result is as follows, and the file can be accessed normally.




## 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 occurs even under this configuration, the root cause may lie in other factors rather than CORS.
In addition to the most lenient configuration, you can also set up more fine-grained control mechanisms for targeted control. For example, the following minimal configuration can be used for a successful match in this case:




Therefore, for most scenarios, it is recommended to use the minimal configuration based on your specific use case to ensure security.

## CORS Configuration Items

CORS configuration items are as follows:

### Origin

This refers to the origin allowing cross-origin requests.

- More than one domain name can be specified, with one domain name per line.
- Wildcard `*` is supported, which means all domain names are allowed. Not recommended.
- A single specific domain name is supported, for example, `http://www.abc.com`.
- Second-level wildcard domain names such as `http://*.abc.com` are supported. Note that each line can contain only one `*`.
- Do not omit protocol name HTTP or HTTPS, and specify the port if the port is not default 80.

### Request methods

Enumerate the allowed cross-origin request methods (one or more).
For example: GET, PUT, POST, DELETE, HEAD.

### Allow-Header

Allowed cross-origin request header.

- More than one domain name can be specified, with one domain name per line.
- Header is easy to be omitted. Therefore, if there is no special requirement, you are advised to set this field to `*`, meaning that all headers are allowed.
- The values are case-insensitive.
- Each header specified in Access-Control-Request-Headers must also be provided in Allowed-Header.

### Expose-Header

This is a list of headers exposed to the browser, that is, the response headers that the user accesses from the app (for example, Javascript's XMLHttpRequest object).

- The configuration should be specific to the requirements of the app. ETag is recommended by default.
- Wildcard is not allowed. The headers are case-insensitive, with one header per line.

### Timeout Max-Age

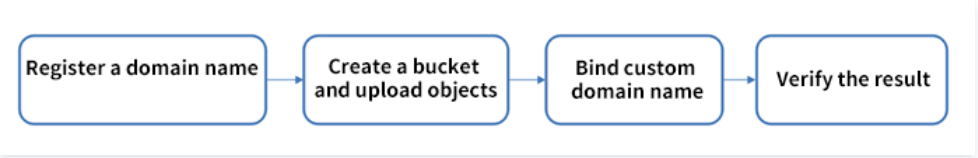
This is the time (in seconds) the browser can cache the results of a preflight request (OPTIONS request) for specific resources. In general cases, you can set it to a bigger value, for example, 60 seconds. Note that this configuration item is optional.

# Hosting a Static Website

Last updated: 2025-03-28 14:32:21

## Feature Overview

IIn this practice, users can host a static website on Tencent Cloud object storage (Cloud Object Storage, COS). Visitors can access the hosted static website through the static website domain provided by COS or a bound custom domain (for example, `www.example.com`). Whether you want to host an existing static website on COS or build one from scratch, this practice can help you host a static website on COS. Here are the specific steps:




**Note:**

- For buckets created after January 1, 2024, when users use the default domain name (including the static website domain) to access files, preview is not supported. For details, please see COS Bucket Domain Usage Security Management Notification. It is recommended that you enable the static website and configure a custom domain name (set the type of the origin server to static website), and then use the custom domain name to access.
- Use COS default domain to access. No static website effect.

## Preparations

The following are related services that may be used during the practice:

- Domain Registration (optional): Before hosting a static website, you need to have a domain name, such as `www.example.com`. You can apply for a domain name through Tencent Cloud Domain Registration. If you are using the static website domain provided by COS, you do not need this service.
- COS: You need to create a bucket in COS for storing the uploaded webpage contents.
- DNSPod: Utilize DNS resolution to achieve the purpose of accessing static websites using custom domain names.

**Note**

All steps in this guide use `www.example.com` as an example domain. In actual operations, please replace this domain with your own domain name.

## Step 1. Register a domain name and obtain ICP filing (optional)

Domain registration is a prerequisite for any service built on the Internet. After the domain name is registered, it needs to go through the MIIT filing procedure before your website can be accessed.

- If you have already registered a domain name and obtained ICP filing for it, skip this step and proceed to step 2.
- If you have already registered a domain name but have not obtained ICP filing for it, please apply for ICP filing.
- If you have not registered a domain name, please register a domain name first, and then proceed with domain name filing.

## Step 2. Create a bucket and upload content

After completing the domain name registration and filing, you need to perform the following tasks in the COS Console to create and configure the website content:

### 1. Creating Bucket

Please log in to the COS Console with your Tencent Cloud account and create a bucket for your website. The bucket is used to store data, so you can store your website contents in it.
If you are using COS for the first time, you can create a bucket directly by clicking **Create Bucket** on the overview page of the console, or by clicking the **Bucket List** navigation bar on the left side. For detailed instructions, please refer to the Create Bucket documentation.

### 2. Configure the bucket and upload content

1. To enable the **Static Website** setting for a bucket, follow these steps:
   1.1 Log in to the COS Console. In the left sidebar, click **Bucket List**. Locate the bucket you just created and click **Configuration Management** on the right.
   1.2 In the left sidebar, select **Basic Configuration > Static Website**, click **Edit**, set the current status to Enabled, the index document to index.html, leave the rest unconfigured, and then click **Save**.
2. Upload your website contents to the bucket. For detailed directions, see Uploading Objects.
   You can use the bucket to store any content you want to host, including text files, photos, and videos. If you haven't built your website yet, just create a file as described in this document.
   For example, you can create a file with the following HTML code and upload it to the bucket. The filename of the website homepage is usually index.html. In subsequent steps, you will need to provide this file as an index document for your website.

```
<!DOCTYPE html>
<html>
 <head>
     <title>Hello COS!</title>
     <meta charset="utf-8">
 </head>
 <body>
     <p>Welcome to the static website feature of COS.</p>
     <p>This is the homepage!</p>
 </body>
</html>
```

> **Note:**
> After the COS static website feature is enabled, if a user accesses any first-level directory that does not point to any files, COS will first match `index.html` in the bucket directory and then `index.htm` by default. If this file does not exist, a 404 error will be returned.

## Step 3. Bind a custom domain name

### 1. Adding a Domain

1. Log in to the COS Console, go to the **Bucket List** page on the left sidebar, and click the bucket that hosts your website.
2. In the left menu bar, click **Domain and Transfer** > **Custom Endpoint** to enter the custom origin server domain name management page.
3. In the "Custom Endpoint" column, click **Add Domain**. The main configuration items are described as follows.
   - **Domain Name**: Enter your purchased custom domain name (for example, www.example.com). For more information, please see Enable Custom Origin Site Domain.
   - **Origin Server Type**: Select **Static Website Endpoint**.
4. Click **Create**. Once configured, wait a few minutes and wait for the domain name deployment to complete. Then copy the corresponding CNAME record and proceed with the domain name resolution procedure.

### 2. Domain Name Resolution

You need to add a CNAME record for your custom domain name at your service provider and point it to the corresponding CNAME record in the above domain addition steps. For more information about configuring domain name resolution, please refer to Appendix: Configuring Domain Name Resolution.

## Step 4. Verify the result

After completing the above steps, verify the result by entering the domain name, e.g. `www.example.com`, in your browser:

- `http://www.example.com` —— Returns the index page (index.html) in the storage bucket named example.
- `http://www.example.com/folder/` —— Returns the index page (folder/index.html) in the folder directory of the storage bucket named example.
- `http://www.example.com/test.html` (non-existent file) —— returns a 404 error. If you need to customize the error document, you can add the **Error Document** setting in Step 2 Configure Static Website. When accessing a non-existent file, the specified error document will be displayed.

**Note**

- In some cases, you may have to clear your browser's cache to see the expected result.
- For each bucket, you can configure only one error document, which can be placed in a sub-directory, such as pages/404.html.

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

Last updated: 2023-09-12 17:57:21

## 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.

## Prerequisites

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, you can directly proceed to the Enable Bucket Static Website Configuration step.

### 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  (Vue2.x)
```

Or:

```
npm install vue-router@4 -S  (Vue3.x)
```

4. Modify the main.js and App.vue files in the project.
The main.js file is 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: '/:w+', 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>
    <img alt="Vue logo" src="./assets/logo.png">
    <router-view></router-view>
  </div>
</template>
```

**Note**
Due to space limitations, only a selection of key code snippets is provided here. To download the complete code, click here.

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. Quickly generate a React project using create-react-app, refer to the official documentation.
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**
> Due to space limitations, only a selection of key code snippets is provided here. To download the complete code, click here.

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
```

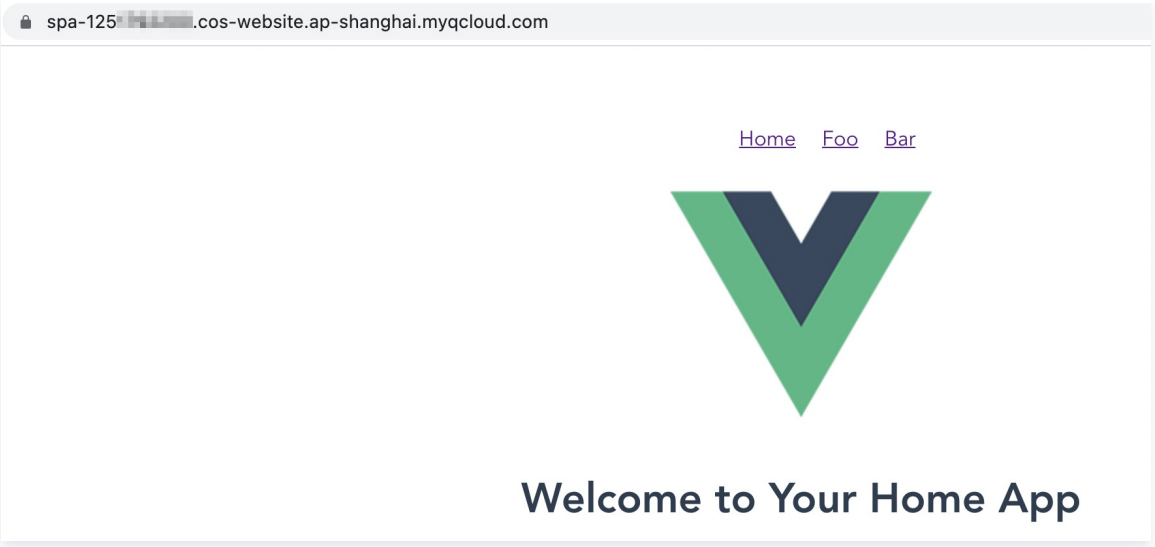
The `build` directory is created in the root directory of the project, and the React program code is ready.

## Configuring the Bucket Static Website

1. Go to the details page of the created bucket and choose **Basic Configurations** > **Static Website**.
2. Configure On the Static Website Management page. For detailed instructions, please refer to 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 from the compilation directory (default is "dist" for Vue and "build" for React) to the root directory of the bucket. For detailed operations, please refer to Uploading Objects.
3. Access the static website domain of the storage bucket.
   You can now see the deployed application homepage, using a Vue application as an 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?

The issue may be due to missing or incorrect configuration of the **Error Document** when setting up the static website. Please review the standard configuration screenshot provided at the beginning of this document, where both the Error Document and Index Document are 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 issue is caused by the lack of **Error Document Response Code** configuration when setting up the static website. To resolve this, refer to the configuration screenshot at the beginning and set the response code to 200.

## 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

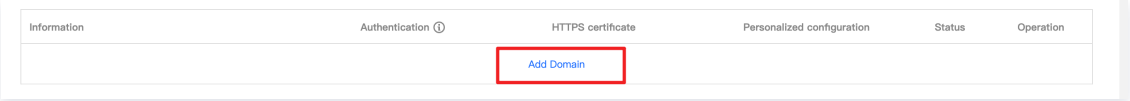
Last updated: 2024-12-17 09:27:14

## 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.

# Direct Data Upload
# Upload Security Restriction

Last updated: 2025-09-19 10:10:03

Documentation introduction to common issues with client upload and response plan. The following example is suitable for direct upload and client SDK upload scenarios.
The server-side code implementation uses Node.js as an example. For more languages, see Server Signature Practice.

## 1.Upload Size Limitation

### Scenario 1: PutObject Upload, Achieved by Setting Numeric_less_than_equal Condition in Temporary Key Policy

```
condition: {
  // Uploaded files must be less than 5MB
  'numeric_less_than_equal': {
    'cos:content-length': 5 * 1024 * 1024
  },
 }
```

### Scenario 2: PostObject Upload, Content-Length-Range Condition Can Be Achieved by Setting Signature Policy Conditions

```
  var policy = JSON.stringify({
    ...
    conditions: [
      ['content-length-range', 1, 5 * 1024 * 1024], // Limit file size range such as 1 - 5MB
    ],
  });
```

The example will return an error 403 for file uploads exceeding 5MB.

## 2.Restrict Upload File Types

### Scenario 1: PutObject Upload, Can Be Achieved by Setting String_like Condition in Temporary Key Policy Via STS

```
condition: {
  // Uploaded files must be image type
  'string_like': {
    'cos:content-type': 'image/*'
  }
 }
```

### Scenario 2: PostObject Upload, Set $Content-Type Condition Via Signature Policy Conditions

```
  var policy = JSON.stringify({
    ...
    conditions: [
      // Limit uploading file content-type must be image type
      ['starts-with', '$Content-Type', 'image/*'],
    ],
  });
```

The example will return an error 403 for non-image file uploads.

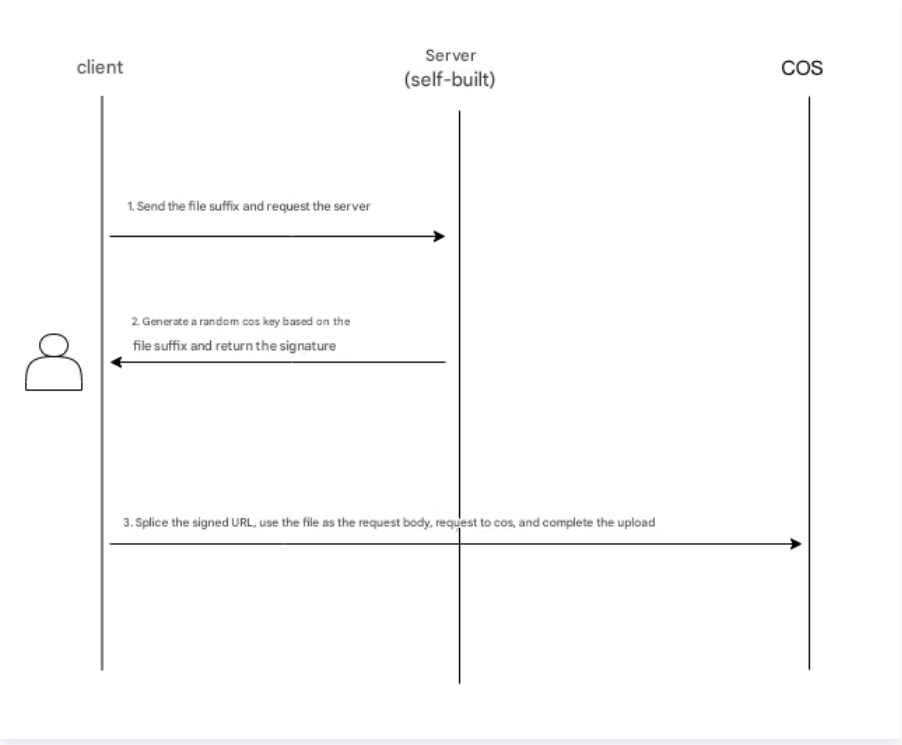
## 3.Prevent File Upload Overwrite

In Web or client upload scenarios, if the file name is specified by the client, there may exist risks of file overwrite upload.
The key measure to prevent file overwrite is **server-side determination of the upload path.**

```
/** Server-side generation upload path example nodejs **/

// Get the file extension ext passed from the frontend
const ext = req.query.ext;
const cosKey = generateCosKey(ext);

function generateCosKey(ext) {
  const date = new Date();
  const m = date.getMonth() + 1;
  const ymd = `${date.getFullYear()}${m < 10 ? `0${m}` : m}${date.getDate()}`;
  const r = ('000000' + Math.random() * 1000000).slice(-6);
  const cosKey = `file/${ymd}/${ymd}_${r}${ext ? `.${ext}` : ''}`;
  return cosKey;
};
```

### Implementation Process

1. Select a file on the client, and the client will send the suffix to the server.
2. The server generates a random COS file path with time based on the suffix, calculates the corresponding signature, and returns the URL and signature information to the client.
3. The client uses a PUT or POST request to directly upload the file to COS.




## Documentation

- Security guide for temporary key used for frontend direct upload to COS
- Web direct upload practice
- Mini Program direct upload practice
- Direct Upload for Mobile Apps practice
- HarmonyOS direct upload practice
- Flutter direct upload practice

- uni-app direct upload practice

# Server Signature Practice

Last updated: 2025-09-19 10:10:33

Documentation introduction on how to set up a secure temporary signature service.

## Solution Strength

- Permission security: It can effectively limit the scope of security permissions and 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.
- Transmission security: Generate the signature on the server to avoid the risk of temporary key leakage during transmission.

## Upload Process

1. Select a file on the client, and the client will send the original file name to the server.
2. The server generates a random COS file path with time based on the file extension, applies for corresponding permissions, and returns the signature information and COS key to the front end.

## Setting Up a Temporary Signature Service

Temporary Access Credentials are restricted-permission keys obtained through the TencentCloud API provided by CAM. When initiating a COS API request, the three fields TmpSecretId, TmpSecretKey, and Token in the returned information from the temporary key API are required to calculate the signature.
When requesting a server signature, first obtain a temporary key, then use it to generate the signature and return it to the client.
The following example code in languages:

Node.js

Complete code can be found in Sample Code.

```
// Temporary key service example
var express = require('express');
var crypto = require('crypto');
var moment = require('moment');
var STS = require('qcloud-cos-sts');
const url = require('url');
const { log } = require('console');

// config
var config = {
  // Get Tencent Cloud Key, recommend using a Sub-user key with limited permissions
https://console.cloud.tencent.com/cam/capi
  secretId: process.env.COS_SECRET_ID,
  secretKey: process.env.COS_SECRET_KEY,
  // key validity period
  durationSeconds: 1800,
  // fill in the bucket and region here, for example: test-1250000000, ap-guangzhou
  bucket: process.env.PERSIST_BUCKET,
  region: process.env.PERSIST_BUCKET_REGION
};

// get temporary key
var getTempCredential = async function(cosKey){
  var shortBucketName = config.bucket.substr(0 , config.bucket.lastIndexOf('-'));
  var appId = config.bucket.substr(1 + config.bucket.lastIndexOf('-'));
  // Start getting temporary key
  var policy = {
    "version": "2.0",
    "statement": [
      {
        "action": [
          // Upload operation required
          "name/cos:PutObject"
        ],
        "effect": "allow",
        "resource": [
```

```
            // Only cosKey resource
            'qcs::cos:' + config.region + ':uid/' + appId + ':prefix//' + appId + '/' + shortBucketName +
'/' + cosKey,
          ]
        }
      ]
    };
    let tempKeys = null;
    try{
      tempKeys = await STS.getCredential({
        secretId: config.secretId,
        secretKey: config.secretKey,
        durationSeconds: config.durationSeconds,
        policy: policy,
      });
      console.log(tempKeys);
      return tempKeys;
    } catch(err){
      console.log(err);
      res.send(JSON.stringify(err));
      return null;
    }
};

// Calculate the signature.
var getSignature = function(tempCredential, httpMethod, cosHost, pathname) {
    const signAlgorithm = 'sha1';
    const credentials = tempCredential.credentials;
    const keyTime = `${tempCredential.startTime};${tempCredential.expiredTime}`;

    // step 1: generate SignKey
    var signKey = crypto.createHmac(signAlgorithm, credentials.tmpSecretKey).update(keyTime).digest('hex');
    console.log("signKey:"+signKey);

    // step two: generate StringToSign
    const httpString = `${httpMethod.toLowerCase()}\n/${pathname}\n\nhost=${cosHost}\n`;
    console.log("httpString:"+httpString);
    const httpStringHash = crypto.createHash(signAlgorithm).update(httpString).digest('hex');
    const stringToSign = `${signAlgorithm}\n${keyTime}\n${httpStringHash}\n`;
    console.log("stringToSign:"+stringToSign);

    // step three: generate Signature
    var signature = crypto.createHmac(signAlgorithm, signKey).update(stringToSign).digest('hex');
    console.log("signature:"+signature);

    // step four: generate authorization
    let authorization = `q-sign-algorithm=${signAlgorithm}&
q-ak=${credentials.tmpSecretId}&
q-sign-time=${keyTime}&
q-key-time=${keyTime}&
q-header-list=host&q-url-param-list=&q-signature=${signature}`;

    // Remove \n caused by line breaks above
    authorization = authorization.replace(/\n/g, '');
    console.log("authorization:"+authorization);

    return authorization;
}

// Create a temporary key service and a static service for debug
var app = express();
// Signature interface for direct upload
```

```
app.all('/sts-server-sign', async function (req, res, next) {
  // Get the field to be signed
  var httpMethod = req.query.httpMethod;
  var cosHost = req.query.host;
  var cosKey = req.query.cosKey;

  console.log(httpMethod + " " + cosHost + " " + cosKey);

  cosHost = decodeURIComponent(cosHost);
  cosKey = decodeURIComponent(cosKey);

  console.log(httpMethod + " " + cosHost + " " + cosKey);

  // Check for anomalies
  if (!config.secretId || !config.secretKey) return res.send({ code: '-1', message: 'secretId or
secretKey not ready' });
  if (!config.bucket || !config.region) return res.send({ code: '-1', message: 'bucket or regions not
ready' });
  if (!httpMethod || !cosHost || !cosKey) return res.send({ code: '-1', message: 'httpMethod or host or
coskey is not empty' });

  // Start getting temporary key
  var tempCredential = await getTempCredential(cosKey);
  if(!tempCredential){
    res.send({ code: -1, message: 'get temp credentials fail' });
    return;
  }

  // Calculate the signature with temporary key
  let authorization = getSignature(tempCredential, httpMethod, cosHost, cosKey);

  // Return domain name, file path, signature, credentials
  res.send({
    code: 0,
    data: {
      cosHost: cosHost,
      cosKey: cosKey,
      authorization: authorization,
      securityToken: tempCredential.credentials.sessionToken
    },
  });
});

app.all('*', function (req, res, next) {
  res.send({ code: -1, message: '404 Not Found' });
});

// Start the signature service
app.listen(3000);
console.log('app is listening at http://127.0.0.1:3000');
```

Go

Complete code can be found in Sample Code.

```
package main

import (
```

```
    "context"
    "encoding/json"
    "errors"
    "fmt"
    "github.com/tencentyun/cos-go-sdk-v5"
    sts "github.com/tencentyun/qcloud-cos-sts-sdk/go"
    "math/rand"
    "net/http"
    "net/url"
    "os"
    "reflect"
    "strings"
    "time"
    "unicode"
)

type Config struct {
    filename        string
    appId           string
    SecretId        string
    SecretKey       string
    Proxy           string
    DurationSeconds int
    Bucket          string
    Region          string
    AllowActions    []string
}

type Permission struct {
    LimitExt           bool     `json:"limitExt"`
    ExtWhiteList       []string `json:"extWhiteList"`
    LimitContentType   bool     `json:"limitContentType"`
    LimitContentLength bool     `json:"limitContentLength"`
}

func generateCosKey(ext string) string {
    date := time.Now()
    m := int(date.Month()) + 1
    ymd := fmt.Sprintf("%d%02d%d", date.Year(), m, date.Day())
    r := fmt.Sprintf("%06d", rand.Intn(1000000))
    cosKey := fmt.Sprintf("file/%s/%s_%s.%s", ymd, ymd, r, ext)
    return cosKey
}

func getPermission() Permission {
    permission := Permission{
        LimitExt:           true,
        ExtWhiteList:       []string{"jpg", "jpeg", "png", "gif", "bmp"},
        LimitContentType:   false,
        LimitContentLength: false,
    }
    return permission
}

func getConfig() Config {
    config := Config{
        filename:        "test.jpg",
        appId:           "12500000000",
        SecretId:        os.Getenv("SECRETID"),  // User's SecretId, recommend using sub-account keys,
adhere to the principle of least privilege to reduce use risk. To obtain sub-account key, see
https://cloud.tencent.com/document/product/598/37140
```

```
        SecretKey:       os.Getenv("SECRETKEY"), // User's SecretKey, recommend using sub-account keys,
adhere to the principle of least privilege to reduce use risk. To obtain sub-account key, see
https://cloud.tencent.com/document/product/598/37140
        Proxy:           os.Getenv("Proxy"),
        DurationSeconds: 1800,
        Bucket:          "0-1253960454",
        Region:          "ap-guangzhou",
        AllowActions: []string{
            "name/cos:PutObject",
        },
    }
    return config
}

func stringInSlice(str string, list []string) bool {
    for _, v := range list {
        if v == str {
            return true
        }
    }
    return false
}

func StructToCamelMap(input interface{}) map[string]interface{} {
    v := reflect.ValueOf(input)
    if v.Kind() == reflect.Ptr {
        v = v.Elem()
    }

    result := make(map[string]interface{})
    typ := v.Type()

    for i := 0; i < v.NumField(); i++ {
        field := typ.Field(i)
        fieldValue := v.Field(i)

        // Convert field name to lower camel case
        key := toLowerCamel(field.Name)

        // Handle nested structure
        if fieldValue.Kind() == reflect.Struct ||
            (fieldValue.Kind() == reflect.Ptr && fieldValue.Elem().Kind() == reflect.Struct) {

            if fieldValue.IsNil() && fieldValue.Kind() == reflect.Ptr {
                result[key] = nil
                continue
            }

            result[key] = StructToCamelMap(fieldValue.Interface())
        } else {
            // Handle basic type
            result[key] = fieldValue.Interface()
        }
    }

    return result
}

// Convert to lower camel case (first letter lowercase)
func toLowerCamel(s string) string {
    if s == "" {
        return s
```

```
    }

    // Handle all caps words (for example ID)
    if strings.ToUpper(s) == s {
        return strings.ToLower(s)
    }

    // Switch to lower camel case
    runes := []rune(s)
    runes[0] = unicode.ToLower(runes[0])
    return string(runes)
}

func getStsCredential() (map[string]interface{}, error) {
    config := getConfig()

    permission := getPermission()

    c := sts.NewClient(
        // Get key through environment variables, os.Getenv method means get environment variables
        config.SecretId,  //os.Getenv("SECRETID"),  // User's SecretId, recommend using sub-account keys,
follow the principle of least privilege to reduce use risk. To obtain sub-account key, see
https://cloud.tencent.com/document/product/598/37140
        config.SecretKey, //os.Getenv("SECRETKEY"),                 // User's SecretKey, recommend using
sub-account keys, adhere to the principle of least privilege to reduce use risk. To obtain sub-account
key, see https://cloud.tencent.com/document/product/598/37140
        nil,
        // sts.Host("sts.internal.tencentcloudapi.com"), // Set domain name, default domain
sts.tencentcloudapi.com
        // sts.Scheme("http"),      // Set protocol. Default is https. Public cloud STS cannot use http.
Set http only in special scenarios.
    )

    condition := make(map[string]map[string]interface{})

    segments := strings.Split(config.filename, ".")
    if len(segments) == 0 {
        //ext := ""
    }
    ext := segments[len(segments)-1]

    if permission.LimitExt {
        extInvalid := ext == "" || !stringInSlice(ext, permission.ExtWhiteList)
        if extInvalid {
            return nil, errors.New("unauthorized file, upload prohibited")
        }
    }

    if permission.LimitContentType {
        condition["string_like_if_exist"] = map[string]interface{}{
            // Only upload allowed with content-type as image/*
            "cos:content-type": "image/*",
        }
    }

    // 3. Limit upload file size
    if permission.LimitContentLength {
        condition["numeric_less_than_equal"] = map[string]interface{}{
            // Upload size limit cannot exceed 5MB (effective for simple upload only)
            "cos:content-length": 5 * 1024 * 1024,
        }
    }
```

```
    key := generateCosKey(ext)
    // Policy overview https://cloud.tencent.com/document/product/436/18023
    opt := &sts.CredentialOptions{
        DurationSeconds: int64(config.DurationSeconds),
        Region:          config.Region,
        Policy: &sts.CredentialPolicy{
            Version: "2.0",
            Statement: []sts.CredentialPolicyStatement{
                {
                    // Permission list for keys. Simple upload and sharding require the following
permissions. For other permission lists, see https://cloud.tencent.com/document/product/436/31923
                    Action: config.AllowActions,
                    Effect: "allow",
                    Resource: []string{
                        // Change to the allowed path prefix. The specific path for upload can be
selected based on your website's user login status, for example: a.jpg or a/* or * (using wildcard *
poses significant security risks, please be cautious when considering using)
                        // The bucket naming format is BucketName-APPID, the bucket filled here must be
in this format
                        "qcs::cos:ap-guangzhou:uid/" + config.appId + ":" + config.Bucket + "/" + key,
                    },
                    // Start building the effective condition condition
                    // For setting rules about condition and COS-supported condition types, see
https://cloud.tencent.com/document/product/436/71306
                    Condition: condition,
                },
            },
        },
    }

    // case 1 request temporary key
    res, err := c.GetCredential(opt)
    if err != nil {
        return nil, err
    }
    // Convert to lower camel case map
    resultMap := StructToCamelMap(res)
    resultMap["bucket"] = config.Bucket
    resultMap["region"] = config.Region
    resultMap["key"] = key

    return resultMap, nil
}

func main() {
    result, err := getStsCredential()
    if err != nil {
        fmt.Printf("request temporary key fail: %v\n", err)
        return // determine whether to exit based on context
    }

    // Type assertion for credentials as map[string]interface{}
    credentials, ok := result["credentials"].(map[string]interface{})
    if !ok {
        fmt.Println("Invalid credential format")
        return
    }
    // Field acquisition with type checking
    tak, tok := credentials["tmpSecretID"].(string)
    tsk, tskok := credentials["tmpSecretKey"].(string)
    token, tokenok := credentials["sessionToken"].(string)
```

```
    if !tok || !tskok || !tokenok {
        fmt.Println("Missing fields or type error in temporary credential")
        return
    }
    host := "https://" + result["bucket"].(string) + ".cos." + result["region"].(string) +
".myqcloud.com"
    u, _ := url.Parse(host)
    b := &cos.BaseURL{BucketURL: u}
    c := cos.NewClient(b, &http.Client{})
    name := result["key"].(string)
    ctx := context.Background()
    opt := &cos.PresignedURLOptions{
        Query:  &url.Values{},
        Header: &http.Header{},
    }
    opt.Query.Add("x-cos-security-token", token)
    signature := c.Object.GetSignature(ctx, http.MethodPut, name, "tak", "tsk", time.Hour, opt, true)
    fmt.Printf("%s\n%s\n%s\n%s\n", tak, tsk, token, signature)
    data := make(map[string]string)

    // Add token and sign
    data["securityToken"] = token
    data["authorization"] = signature
    data["cosHost"] = host
    data["cosKey"] = name
    // Convert to JSON
    jsonData, err := json.MarshalIndent(data, "", "  ")
    if err != nil {
        fmt.Println("JSON encoding failure:", err)
        return
    }
    fmt.Println(string(jsonData))
}
```

### PHP

Complete code can be found in Sample Code.

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

use QCloud\COSSTS\Sts;

// Generate the COS file path Filename to upload
function generateCosKey($ext) {
  $ymd = date('Ymd');
  $r = substr('000000' . rand(), -6);
  $cosKey = 'file/' . $ymd. '/' . $ymd . '_' . $r;
  if ($ext) {
    $cosKey = $cosKey . '.' . $ext;
  }
  return $cosKey;
};

// Get temporary key for single file upload permission
function getKeyAndCredentials($filename) {
  // Business implements user login status verification, such as token validation
```

```
  // $canUpload = checkUserRole($userToken);
  // if (!$canUpload) {
  //   return 'Current user has no upload permission';
  // }

  // Control file upload types and size, enable on demand
  $permission = array(
    'limitExt' => false, // Restrict file extensions
    'extWhiteList' => ['jpg', 'jpeg', 'png', 'gif', 'bmp'], // Restricted file extensions
    'limitContentType' => false, // Restrict content types
    'limitContentLength' => false, // Limit upload file size
  );
  $condition = array();

  // The client passes the original file name, generate a random Key based on the file suffix
  $ext = pathinfo($filename, PATHINFO_EXTENSION);

  // 1. Limit file upload extensions
  if ($permission['limitExt']) {
    if ($ext === '' || array_key_exists($ext, $permission['extWhiteList'])) {
      return 'unauthorized file, upload prohibited';
    }
  }

  // 2. Limit file upload content-type
  if ($permission['limitContentType']) {
    // Only upload allowed with content-type as image/*
    $condition['string_like_if_exist'] = array('cos:content-type' => 'image/*');
  }

  // 3. Limit upload file size
  if ($permission['limitContentLength']) {
    // Upload size limit cannot exceed 5MB (effective for simple upload only)
    $condition['numeric_less_than_equal'] = array('cos:content-length' => 5 * 1024 * 1024);
  }

  $cosKey = generateCosKey($ext);
  $bucket = 'test-125000000'; // Replace with your bucket
  $region = 'ap-guangzhou'; // Replace with the park where the bucket resides

  $config = array(
    'url' => 'https://sts.tencentcloudapi.com/', // URL and domain should be consistent
    'domain' => 'sts.tencentcloudapi.com', // Domain name, optional, defaults to sts.tencentcloudapi.com
    'proxy' => '',
    'secretId' => "",//getenv('GROUP_SECRET_ID'), // Fixed key. For plaintext key, fill in directly as
'xxx', do not use getenv() function
    'secretKey' => "",//getenv('GROUP_SECRET_KEY'), // Fixed key. For plaintext key, fill in directly as
'xxx', do not use getenv() function
    'bucket' => $bucket, // Replace with your bucket
    'region' => $region, // Replace with the park where the bucket resides
    'durationSeconds' => 1800, // key validity period
    'allowPrefix' => array($cosKey), // Only allocate path permission for the current key
    // Permission list for keys. Simple upload and sharding require the following permissions. For other
permission lists, see https://cloud.tencent.com/document/product/436/31923
    'allowActions' => array (
        // Simple upload
        'name/cos:PutObject'
    ),
  );

  if (!empty($condition)) {
    $config['condition'] = $condition;
```

```
  }

  $sts = new Sts();
  $tempKeys = $sts->getTempKeys($config);
  $resTemp = array_merge(
    $tempKeys,
    [
      'startTime' => time(),
      'bucket' => $bucket,
      'region' => $region,
      'key' => $cosKey,
    ]
);
  return $resTemp;
}

function getSign()
{
    $filename = "test.jpg";
    $method = "putObject";
    $result = getKeyAndCredentials($filename);
    $credentials = $result["credentials"];
    $sessionToken = $credentials["sessionToken"];
    $tmpSecretId = $credentials["tmpSecretId"];
    $tmpSecretKey = $credentials["tmpSecretKey"];
    $expiredTime = $result["expiredTime"];
    $startTime = $result["startTime"];
    $bucket = $result["bucket"];
    $region = $result["region"];
    $key = $result["key"];

    $cosClient = new Qcloud\Cos\Client(
        array(
            'region' => $region,
            'scheme' => 'https', // protocol header, defaults to http
            'signHost' => true, //By default, sign the Header Host. You can also choose not to sign the
Header Host, but this may cause request failure or security vulnerability. Set to false if not signing
the host.
            'credentials'=> array(
                'secretId'  => $tmpSecretId,
                'secretKey' => $tmpSecretKey,
                'token' => $sessionToken)));
    ### Upload pre-signiture by using the simple upload.
    try {
        $url = $cosClient->getPresignedUrl($method,array(
            'Bucket' => $bucket,
            'Key' => $key,
            'Body' => "",
            'Params'=> array('x-cos-security-token' => $sessionToken),
            'Headers'=> array(),
        ), $expiredTime - $startTime); //Signature validity time

        $parsedUrl = parse_url($url);
        $host = 'https://' . $parsedUrl['host']; // automatically include port (if any)
        $queryString = isset($parsedUrl['query']) ? $parsedUrl['query'] : '';
        $queryParts = explode('&', $queryString);
        $signParts = array_filter($queryParts, function($part) {
            return strpos($part, 'x-cos-security-token=') !== 0;
        });
        $sign = implode('&', $signParts);
        $result = [
            'cosHost' => $host,
```

```
            'cosKey' => $key,
            'authorization' => $sign,
            'securityToken' => $sessionToken
        ];

        echo json_encode($result, JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES);

    } catch (\Exception $e) {
        //Request failed
        echo($e);
    }
}
getSign();
```

### Python

Complete code can be found in Sample Code.

```
#!/usr/bin/env python
# coding=utf-8
import json
import os
import datetime
import random
from urllib.parse import urlencode

from qcloud_cos import CosConfig, CosS3Client

from sts.sts import Sts


if __name__ == '__main__':

    # config
    config = {
        "filename":"test.jpg",
        "appId": "1250000000",
        "secretId": os.getenv("SecretId"),
        "secretKey": os.getenv("SecretKey"),
        "proxy": os.getenv("Proxy"),
        "durationSeconds": 1800,
        "bucket": "0-1253960454",
        "region": "ap-guangzhou",
        # key upload permission list
        "allowActions": [
            # Simple upload
            "name/cos:PutObject"
        ],
    }

    permission = {
        "limitExt": True,  # Restrict file extensions
        "extWhiteList": ["jpg", "jpeg", "png", "gif", "bmp"],  # Restricted file extensions
        "limitContentType": False,  # Limit uploading contentType
        "limitContentLength": False,  # Limit upload file size
    }


    # Generate the COS file path Filename to upload
```

```
    def generate_cos_key(ext=None):
        date = datetime.datetime.now()
        ymd = date.strftime('%Y%m%d')
        r = str(int(random.random() * 1000000)).zfill(6)
        cos_key = f"file/{ymd}/{ymd}_{r}.{ext if ext else ''}"
        return cos_key


    segments = config['filename'].split(".")
    ext = segments[-1] if segments else ""
    key = generate_cos_key(ext)
    resource = f"qcs::cos:{config['region']}:uid/{config['appId']}:{config['bucket']}/{key}"

    condition = {}

    # 1. Limit file upload extensions
    if permission["limitExt"]:
        ext_invalid = not ext or ext not in permission["extWhiteList"]
        if ext_invalid:
            print('Unauthorized file, upload prohibited')

    # 2. Limit file upload content-type
    if permission["limitContentType"]:
        condition.update({
            "string_like_if_exist": {
                # Only upload allowed with content-type as image/*
                "cos:content-type": "image/*"
            }
        })

    # 3. Limit upload file size
    if permission["limitContentLength"]:
        condition.update({
            "numeric_less_than_equal": {
                # Upload size limit cannot exceed 5MB (effective for simple upload only)
                "cos:content-length": 5 * 1024 * 1024
            }
        })


    def get_credential_demo():
        credentialOption = {
            # Temporary key valid duration in seconds
            'duration_seconds': config.get('durationSeconds'),
            'secret_id': config.get("secretId"),
            # fixed key
            'secret_key': config.get("secretKey"),
            # Replace with your bucket
            'bucket': config.get("bucket"),
            'proxy': config.get("proxy"),
            # Replace with the bucket's located region
            'region': config.get("region"),
            "policy": {
                "version": '2.0',
                "statement": [
                    {
                        "action": config.get("allowActions"),
                        "effect": "allow",
                        "resource": [
                            resource
                        ],
                        "condition": condition
```

```
                    }
                ],
            },
        }

        try:

            sts = Sts(credentialOption)
            response = sts.get_credential()
            credential_dic = dict(response)
            credential_info = credential_dic.get("credentials")
            credential = {
                "bucket": config.get("bucket"),
                "region": config.get("region"),
                "key": key,
                "startTime": credential_dic.get("startTime"),
                "expiredTime": credential_dic.get("expiredTime"),
                "requestId": credential_dic.get("requestId"),
                "expiration": credential_dic.get("expiration"),
                "credentials": {
                    "tmpSecretId": credential_info.get("tmpSecretId"),
                    "tmpSecretKey": credential_info.get("tmpSecretKey"),
                    "sessionToken": credential_info.get("sessionToken"),
                },
            }
            return credential
        except Exception as e:
            print(e)


    result = get_credential_demo()
    credentials = result["credentials"]
    secret_id = credentials["tmpSecretId"]
    secret_key = credentials["tmpSecretKey"]
    token = credentials["sessionToken"]
    bucket = result["bucket"]
    region = result["region"]
    key = result["key"]
    expired = result["expiredTime"]

    config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token)
    client = CosS3Client(config)
    sign = client.get_auth(Method='put',Bucket=bucket, Key=key, Expired=expired, Params={
        'x-cos-security-token': token
    },SignHost=True)
    sign = urlencode(dict([item.split('=', 1) for item in sign.split('&')]))
    host = "https://" + result["bucket"] + ".cos." + result["region"] + ".myqcloud.com"
    response = {
        "cosHost":host,
        "cosKey":key,
        "authorization":sign,
        "securityToken":token
    }
    print('get data : ' + json.dumps(response, indent=4))
```

Java

Complete code can be found in Sample Code.

```
package com.tencent.cloud;

import com.qcloud.cos.COSClient;
import com.qcloud.cos.ClientConfig;
import com.qcloud.cos.auth.BasicSessionCredentials;
import com.qcloud.cos.auth.COSCredentials;
import com.qcloud.cos.http.HttpMethodName;
import com.qcloud.cos.region.Region;
import com.tencent.cloud.cos.util.Jackson;
import org.junit.Test;

import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.*;

public class ServerSignTest {

    public static String generateCosKey(String ext) {
        Date date = new Date();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");
        String ymd = dateFormat.format(date);

        Random random = new Random();
        int r = random.nextInt(1000000);
        String rStr = String.format("%06d", r);

        String cosKey = String.format("file/%s/%s_%s.%s", ymd, ymd, rStr, ext != null ? ext : "");
        return cosKey;
    }

    // obtain configuration information
    public TreeMap<String,Object> getConfig(){

        String bucket = "0-1250000000";
        String appId = "1250000000";
        String filename = "test.jpg";
        String region = "ap-guangzhou";
        String secretId = "";
        String secretKey = "";
        String proxy  = "";
        int durationSeconds = 1800;

        String[] segments = filename.split("\\.");
        String ext = segments.length > 0 ? segments[segments.length - 1] : "";

        // temporary key limit
        Boolean limitExt = false; // Restrict file extensions
        List extWhiteList = Arrays.asList("jpg", "jpeg", "png", "gif", "bmp"); // Restricted file
extensions
        Boolean limitContentType = false; // Limit uploading contentType
        Boolean limitContentLength = false; // Limit upload file size


        Map<String, Object> condition = new HashMap();

        // 1. Limit file upload extensions
        if (limitExt) {
            boolean extInvalid = ext == null || !extWhiteList.contains(ext);
            if (extInvalid) {
                System.out.println("Unauthorized file, upload prohibited");
                return null;
            }
```

```
        }

        // 2. Limit file upload content-type
        if (limitContentType) {
            condition.put("string_like_if_exist", new HashMap<String, String>() {{
                put("cos:content-type", "image/*");
            }});
        }

        // 3. Limit upload file size (only takes effect for simple upload)
        if (limitContentLength) {
            condition.put("numeric_less_than_equal", new HashMap<String, Long>() {{
                put("cos:content-length", 5L * 1024 * 1024);
            }});
        }
        String key = generateCosKey(ext);
        String resource = "qcs::cos:" + region + ":uid/" + appId + ':' + bucket + '/' + key;
        List allowActions = Arrays.asList(
                // Simple upload
                "name/cos:PutObject"
        );

        // Build policy
        Map<String, Object> policy = new HashMap();
        policy.put("version", "2.0");
        Map<String, Object> statement = new HashMap();
        statement.put("action", allowActions);
        statement.put("effect", "allow");
        List<String> resources = Arrays.asList(
                resource
        );
        statement.put("resource", resources);
        statement.put("condition", condition);
        policy.put("statement", Arrays.asList(statement));


        // Build config
        TreeMap <String,Object> config = new TreeMap<String, Object>();
        config.put("secretId",secretId);
        config.put("secretKey",secretKey);
        config.put("proxy",proxy);
        config.put("duration",durationSeconds);
        config.put("bucket",bucket);
        config.put("region",region);
        config.put("key",key);
        config.put("policy",Jackson.toJsonPrettyString(policy));
        return config;
    }

    public TreeMap <String,Object> getKeyAndCredentials() {
        TreeMap config = this.getConfig();
        try {
            Response response = CosStsClient.getCredential(config);
            TreeMap <String,Object> credential = new TreeMap<String, Object>();
            TreeMap <String,Object> credentials = new TreeMap<String, Object>();
            credentials.put("tmpSecretId",response.credentials.tmpSecretId);
            credentials.put("tmpSecretKey",response.credentials.tmpSecretKey);
            credentials.put("sessionToken",response.credentials.sessionToken);
            credential.put("startTime",response.startTime);
            credential.put("expiredTime",response.expiredTime);
            credential.put("requestId",response.requestId);
            credential.put("expiration",response.expiration);
```

```
            credential.put("credentials",credentials);
            credential.put("bucket",config.get("bucket"));
            credential.put("region",config.get("region"));
            credential.put("key",config.get("key"));
            return credential;
        } catch (Exception e) {
            e.printStackTrace();
            throw new IllegalArgumentException("no valid secret !");
        }
    }
    /**
     Basic temporary key application example, suitable for granting a set of operation permissions to
multiple object paths in a bucket
     */
    @Test
    public void testGetKeyAndCredentials() {
        TreeMap <String,Object> credential = this.getKeyAndCredentials();
        TreeMap <String,Object> credentials = (TreeMap<String, Object>) credential.get("credentials");
        try {


            String tmpSecretId = (String) credentials.get("tmpSecretId");
            String tmpSecretKey = (String) credentials.get("tmpSecretKey");
            String sessionToken = (String) credentials.get("sessionToken");
            Date expiredTime = new Date((Long) credential.get("expiredTime"));
            String key = (String) credential.get("key");
            String bucket = (String) credential.get("bucket");
            String region = (String) credential.get("region");

            COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sessionToken);
            ClientConfig clientConfig = new ClientConfig();
            clientConfig.setRegion(new Region(region));

            COSClient cosClient = new COSClient(cred, clientConfig);


            Map<String, String> headers = new HashMap<String,String>();
            Map<String, String> params = new HashMap<String,String>();
            params.put("x-cos-security-token",sessionToken);
            URL url = cosClient.generatePresignedUrl(bucket, key, expiredTime, HttpMethodName.PUT,
headers, params);
            String host = "https://" + url.getHost();
            String query = url.toString().split("\\?")[1];
            String sign = query.split("&x-cos-security-token")[0];
            TreeMap <String,Object> result = new TreeMap<String, Object>();
            result.put("cosHost",host);
            result.put("cosKey",key);
            result.put("authorization",sign);
            result.put("securityToken",sessionToken);
            System.out.println(Jackson.toJsonPrettyString(result));
        } catch (Exception e) {
                e.printStackTrace();
            throw new IllegalArgumentException("no valid sign !");
        }
    }
}
```

### .NET(C#)

Complete code can be found in Sample Code.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Net.Mail;
using COSSTS;
using COSXML;
using COSXML.Auth;
using COSXML.Model.Tag;
using Newtonsoft.Json;
using Formatting = System.Xml.Formatting;


namespace COSSnippet
{
    public class ServerSign
    {
        //permanent key
        string secretId = "";
        string secretKey = "";

        string bucket = "bucket-1250000000";
        string appId = "1250000000";
        string region = "ap-guangzhou";
        string filename = "test.jpg";
        string method = "put";
        int time = 1800;

        // limit
        Boolean limitExt = false; // Restrict file extensions
        List<string> extWhiteList = new List<String> { "jpg", "jpeg", "png", "gif", "bmp" }; //
Restricted file extensions
        Boolean limitContentType = false; // Limit uploading contentType
        Boolean limitContentLength = false; // Limit upload file size


        public string generateCosKey(string ext)
        {
            DateTime date = DateTime.Now;
            int m = date.Month;
            string ymd = $"{date.Year}{(m < 10 ? $"0{m}" : m.ToString())}{date.Day}";

            Random random = new Random();
            string r = random.Next(0, 1000000).ToString("D6"); // Generate a 6-digit random number with
leading zeros

            string cosKey = $"file/{ymd}/{ymd}_{r}.{(string.IsNullOrEmpty(ext) ? "" : ext)}";
            return cosKey;
        }

        public Dictionary<string, object> getConfig()
        {
            Dictionary<string, object> config = new Dictionary<string, object>();
            string[] allowActions = new string[] {  // Allowed operation scope, using upload as an
example
                "name/cos:PutObject"
            };
```

```
string[] segments = filename.Split(".");
string ext = segments.Length > 0 ? segments[segments.Length - 1] : string.Empty;
string key = generateCosKey(ext);
string resource = $"qcs::cos:{region}:uid/{appId}:{bucket}/{key}";

var condition = new Dictionary<string, object>();

// 1. Limit file upload extensions
if (limitExt)
{
    var extInvalid = string.IsNullOrEmpty(ext) || !extWhiteList.Contains(ext);
    if (extInvalid)
    {
        Console.WriteLine("Unauthorized file, upload prohibited");
        return null;
    }
}

// 2. Limit file upload content-type
if (limitContentType)
{
    condition["string_like_if_exist"] = new Dictionary<string, string>
    {
        { "cos:content-type", "image/*" } // Only upload allowed with content-type as image/*
    };
}

// 3. Limit upload file size (only takes effect for simple upload)
if (limitContentLength)
{
    condition["numeric_less_than_equal"] = new Dictionary<string, long>
    {
        { "cos:content-length", 5 * 1024 * 1024 } // Upload size limit cannot exceed 5MB
    };
}

var policy = new Dictionary<string, object>
{
    { "version", "2.0" },
    { "statement", new List<Dictionary<string, object>>
        {
            new Dictionary<string, object>
            {
                { "action", allowActions },
                { "effect", "allow" },
                { "resource", new List<string>
                    {
                        resource,
                    }
                },
                { "condition", condition }
            }
        }
    }
};

// Serialize to JSON and output
string jsonPolicy = JsonConvert.SerializeObject(policy);

config.Add("bucket", bucket);
config.Add("region", region);
```

```
            config.Add("durationSeconds", time);

            config.Add("secretId", secretId);
            config.Add("secretKey", secretKey);
            config.Add("key", key);
            config.Add("policy", jsonPolicy);
            return config;
        }

        // Obtain temporary access credentials for federated identity
https://cloud.tencent.com/document/product/1312/48195
        public Dictionary<string, object> GetCredential()
        {

            var config = getConfig();
            // get temporary key
            Dictionary<string, object> credential = STSClient.genCredential(config);
            Dictionary<string, object> credentials = JsonConvert.DeserializeObject<Dictionary<string,
object>>(JsonConvert.SerializeObject((object) credential["Credentials"]));
            Dictionary<string, object> credentials1 = new Dictionary<string, object>();
            credentials1.Add("tmpSecretId",credentials["TmpSecretId"]);
            credentials1.Add("tmpSecretKey",credentials["TmpSecretKey"]);
            credentials1.Add("sessionToken",credentials["Token"]);
            Dictionary<string, object> dictionary1 = new Dictionary<string, object>();
            dictionary1.Add("credentials",credentials1);
            dictionary1.Add("startTime",credential["StartTime"]);
            dictionary1.Add("requestId",credential["RequestId"]);
            dictionary1.Add("expiration",credential["Expiration"]);
            dictionary1.Add("expiredTime",credential["ExpiredTime"]);
            dictionary1.Add("bucket",config["bucket"]);
            dictionary1.Add("region",config["region"]);
            dictionary1.Add("key",config["key"]);
            return dictionary1;
        }
        static void Main(string[] args)
        {
            ServerSign m = new ServerSign();
            Dictionary<string, object> result = m.GetCredential();
            Dictionary<string, object> credentials = (Dictionary<string, object>)result["credentials"];
            string tmpSecretId = (string)credentials["tmpSecretId"];
            string tmpSecretKey = (string)credentials["tmpSecretKey"];
            string sessionToken = (string)credentials["sessionToken"];
            string bucket = (string)result["bucket"];
            string region = (string)result["region"];
            string key = (string)result["key"];
            long expiredTime = (long)result["expiredTime"];
            int startTime = (int)result["startTime"];

            QCloudCredentialProvider cosCredentialProvider = new DefaultSessionQCloudCredentialProvider(
                tmpSecretId, tmpSecretKey, expiredTime, sessionToken);

            CosXmlConfig config = new CosXmlConfig.Builder()
                .IsHttps(true)  //set default HTTPS request
                .SetRegion(region) // Set a default bucket region.
                .SetDebugLog(true) // Display log.
                .Build(); // Create CosXmlConfig object.
            CosXml cosXml = new CosXmlServer(config, cosCredentialProvider);

            PreSignatureStruct preSignatureStruct = new PreSignatureStruct();

            preSignatureStruct.appid = m.appId;//"1250000000";
```

```
            preSignatureStruct.region = region;//"COS_REGION";

            preSignatureStruct.bucket = bucket;//"examplebucket-1250000000";
            preSignatureStruct.key = "exampleObject"; //object key
            preSignatureStruct.httpMethod = "PUT"; //HTTP request method
            preSignatureStruct.isHttps = true; //generate HTTPS request URL
            preSignatureStruct.signDurationSecond = 600; //request signature time is 600s
            preSignatureStruct.headers = null; //headers to validate in signature
            preSignatureStruct.queryParameters = null; //request parameters to validate in URL signature
            //Upload pre-signed URL (pre-signed URL calculated using permanent key)
            Dictionary<string, string> queryParameters = new Dictionary<string, string>();
            queryParameters.Add("x-cos-security-token",sessionToken);
            Dictionary<string, string> headers = new Dictionary<string, string>();
            string authorization = cosXml.GenerateSign(m.method,key,queryParameters,headers,expiredTime -
startTime,expiredTime - startTime);

            string host = "https://" + bucket + ".cos." + region + ".myqcloud.com";
            Dictionary<string, object> response = new Dictionary<string, object>();
            response.Add("cosHost",host);
            response.Add("cosKey",key);
            response.Add("authorization",authorization);
            response.Add("securityToken",sessionToken);
            Console.WriteLine($"{JsonConvert.SerializeObject(response)}");
        }
    }
}
```

# Practice of Direct Transfer for Web End

Last updated: 2024-09-29 12:04:47

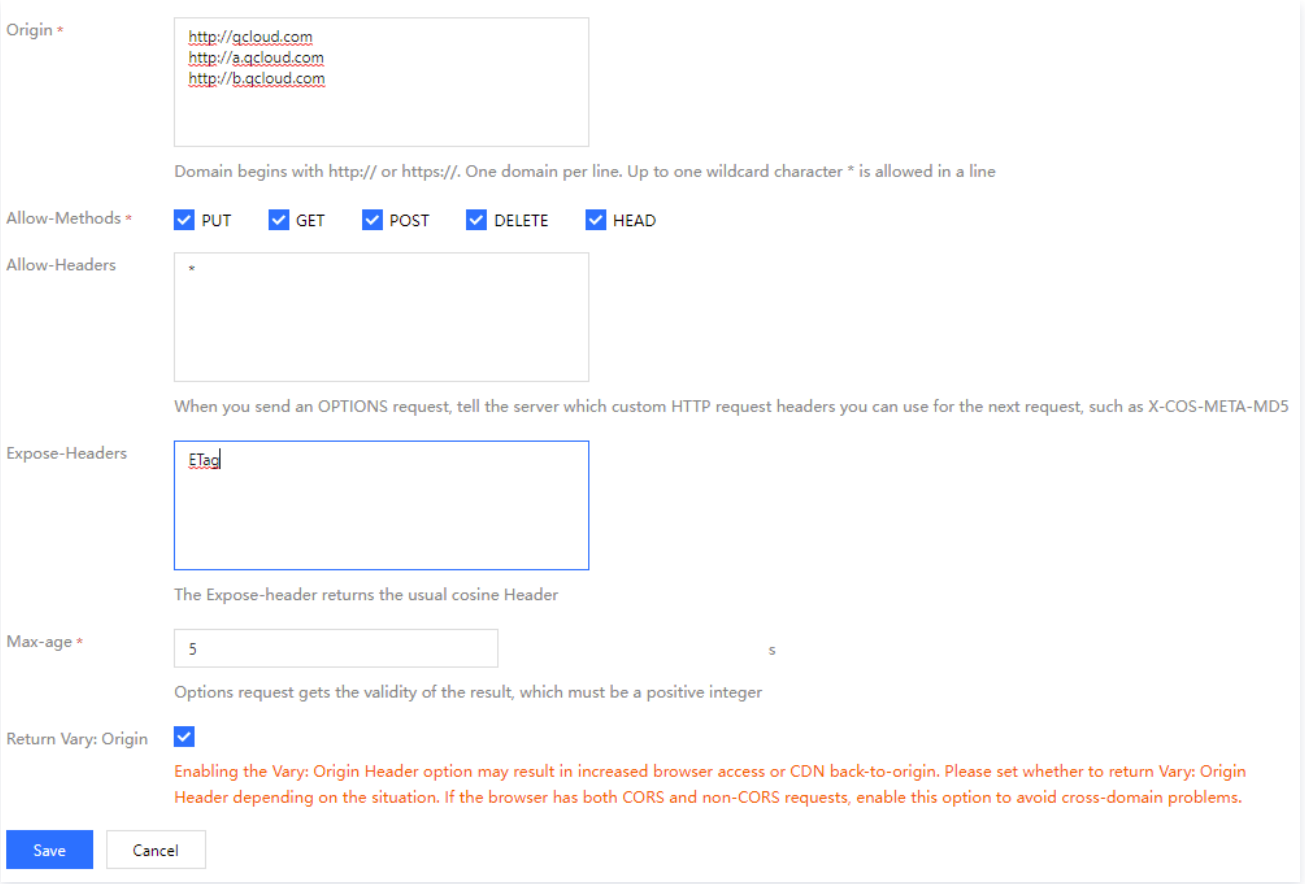
## Feature Overview

This document explains how to upload files directly to a Cloud Object Storage (COS) bucket from a web page without relying on an SDK, using simple code.

> **Note**
> The content of this document is based on the XML version of the API.

## Preparations

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 refer to the Creating a Bucket document.
2. Navigate to the bucket details page and click the **Security Management** tab. Scroll down to find the **Cross-Origin Resource Sharing (CORS) Settings** configuration, and click **Add Rule**. Configure as shown in the example below. For more information, please refer to the Setting up Cross-Origin Access document.




3. Log in to the Access Management Console to obtain your project's SecretId and SecretKey.

## Solution Description

### Implementation Process

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 a timestamp based on the file extension and calculates the corresponding signature. It then returns the URL and signature information to the front-end.
3. The front-end uses PUT or POST requests to directly upload files to COS.

### Solution strengths

- 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:

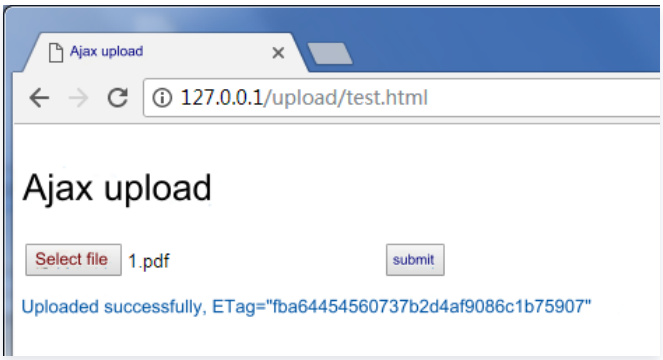
1. Prepare the bucket configuration according to the steps of Prerequisites.
2. Create a `test.html` file and copy the code below into the `test.html` file.
3. Deploy the back-end signature service and modify the signature service address in `test.html`.
4. Place `test.html` under the Web server, access 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>
    <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:
https://github.com/tencentyun/cos-demo/blob/main/server/upload-sign/nodejs/app.js
          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 path.
            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-domain rules are configured
properly'
              );
            };
            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:

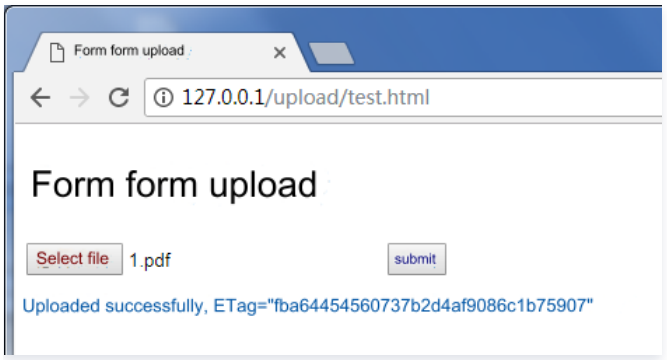
1. Obtain the bucket information by taking the steps in Prerequisites.
2. Create a `test.html` file and copy the code below into the `test.html` file.
3. Deploy the signature service at the backend and modify the signature service address in `test.html`.
4. Place `test.html` on the Web server. Then, 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" />

    <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, demo:
https://github.com/tencentyun/cos-demo/blob/main/server/upload-sign/nodejs/app.js
          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 path.
            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 back after uploading
            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 long file content affecting
the signature judgment and authentication.
            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 rules are configured
properly'
              );
            };
            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.

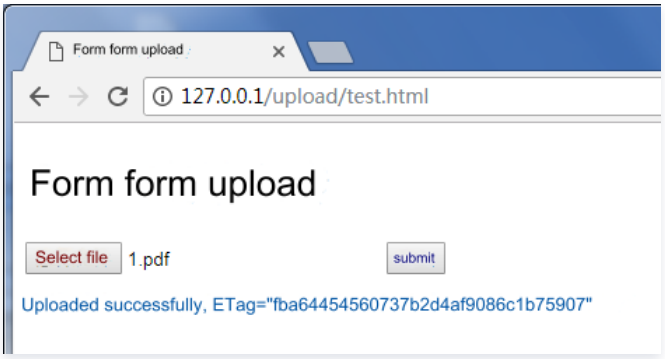
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Simple Form Upload (Compatible with IE8) (Server-side Signature Calculation)</title>
    <style>
      h1,
      h2 {
        font-weight: normal;
      }
      #msg {
        margin-top: 10px;
      }
    </style>
  </head>
  <body>
    <h1>Simple Form Upload (Compatible with IE8) (Server-side Signature Calculation)</h1>
    <div>Minimum compatibility with IE6 for uploading, and onprogress is not supported</div>.

    <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 affecting signature
judgment and authentication -->
      <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, demo:
https://github.com/tencentyun/cos-demo/blob/main/server/upload-sign/nodejs/app.js
          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 path.
            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 back after uploading
            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 png files
  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 5GB per object. For
example, restrict uploaded files to no more than 5MB.
  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: 2023-09-12 18:28:49

## Feature Overview

This document describes how to use simple code to upload files to a COS bucket directly through a Weixin Mini Program without using an SDK.

> **Note**
> The content of this document is based on the XML version of the API.

## Prerequisites

1. Log in to the Object Storage Console and create a bucket. Set the BucketName (bucket name) and Region (region name). For more information, see the Create Bucket document.
2. Log in to the CAM console and obtain your project's SecretId and SecretKey on the API key management page.
3. Configuring the Weixin Mini Program Allowlist

To request COS in your Weixin Mini Program, you need to log in to the Weixin Official Accounts Platform and configure the domain name allowlist in **Development** > **Development Settings**. The SDK uses two APIs: wx.uploadFile and wx.request.

- For the cos.postObject method, requests are sent using wx.uploadFile.
- For other methods, requests are sent using wx.request.

Both need to have the COS domain name configured in their respective whitelists. For example: `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com`.

## Solution Description

### 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 a timestamp based on the file extension and calculates the corresponding signature. It then returns the URL and signature information to the front-end.
3. The front-end uses PUT or POST requests to directly upload files to COS.

### Solution strengths

- Permission Security: Using server-side signatures can effectively limit the secure permission scope, allowing uploads to only 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.

## Practical Steps

### Configure Server-side Implementation for Signature Generation

> **Note**
> During the official deployment, please add a layer of your website's own permission verification to the server-side.

For information on how to calculate signatures, refer to the Request Signature document.
For server-side Node.js signature calculation code, refer to the Node.js Example.

### Weixin Mini Program Upload Example

The following code demonstrates both the PUT Object (recommended) and POST Object interfaces, with operational guidance as follows:

#### Using POST for Uploads

```
var uploadFile = function () {
  // URL encode format for a wider range of characters
  var camSafeUrlEncode = function (str) {
    return encodeURIComponent(str)
      .replace(/!/g, '%21')
      .replace(/'/g, '%27')
      .replace(/\(/g, '%28')
      .replace(/\)/g, '%29')
      .replace(/\*/g, '%2A');
  };

  // Obtain Signature
  var getAuthorization = function (options, callback) {
    wx.request({
      method: 'GET',
      // Replace with your own server address to obtain 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": "Temporary secret retrieval failed",
            content: JSON.stringify(data),
            showCancel: false,
          });
        }
      },
      error: function (err) {
        wx.showModal({
          "title": "Temporary secret retrieval failed",
          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: 'Upload successful', content: url, showCancel: false });
        } 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('Current progress:', res);
    });
  };

  // Upload file
  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; // Allowing the server to determine the filename is more secure
      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 file
  wx.chooseMedia({
    count: 1, // Default 9
    sizeType: ['original'], // Specifies whether to use the original image or a compressed version; the
default is the original image.
    sourceType: ['album', 'camera'], // Specify the source as either album or camera; both are available by
default.
    success: function (res) {
      uploadFile(res.tempFiles[0].tempFilePath);
    },
  });
};
```

### Using PUT for Upload

```
var uploadFile = function () {
  // URL encode format for a wider range of characters
  var camSafeUrlEncode = function (str) {
    return encodeURIComponent(str)
      .replace(/!/g, '%21')
      .replace(/'/g, '%27')
      .replace(/\(/g, '%28')
      .replace(/\)/g, '%29')
      .replace(/\*/g, '%2A');
  };
```

```
// Obtain Signature
var getAuthorization = function (options, callback) {
  wx.request({
    method: 'GET',
    // Replace with your own server address to obtain 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": "Temporary secret retrieval failed",
          content: JSON.stringify(data),
          showCancel: false,
        });
      }
    },
    error: function (err) {
      wx.showModal({
        "title": "Temporary secret retrieval failed",
        content: JSON.stringify(err),
        showCancel: false,
      });
    },
  });
};

var putFile = function ({ prefix, filePath, key, AuthData }) {
  // Put upload requires reading the actual content of the file for uploading
  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": "Upload successful",
              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('Current progress:', res);
        });
      },
    });
  };

  // Upload file
  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 file
  wx.chooseMedia({
    count: 1, // Default 9
    sizeType: ['original'], // Specifies whether to use the original image or a compressed version; the
default is the original image.
    sourceType: ['album', 'camera'], // Specify the source as either album or camera; both are available by
default.
    success: function (res) {
      uploadFile(res.tempFiles[0].tempFilePath);
    },
  });
};
```

## Documentation

If you need to use a Mini Program SDK, see Getting Started with Mini Program SDK.

# Practice of Direct Upload for Mobile Apps

Last updated: 2023-09-12 18:30:06

## Feature Overview

This document explains how to upload files directly to a Cloud Object Storage (COS) bucket from an App without relying on the SDK, using simple code.

> **Note**
> The content of this document is based on the XML version of the API.

## Preparations

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 refer to the Creating Bucket document.
2. Log in to the Access Management Console to obtain your project's SecretId and SecretKey.

## Practical Steps

The overall process is as follows:

1. The client calls the server-side interface and passes the file extension. The server then generates a COS key and direct upload URL based on the extension and timestamp.
2. The server obtains temporary keys through the STS SDK.
3. The server uses the obtained temporary key to sign the direct upload URL and returns the URL, signature, token, and other information.
4. After obtaining the information from step 3, the client initiates a PUT request with the signature, token, and other headers for the upload.

For specific code, refer to the iOS example and Android example.

### Configure Server-side Implementation for Signature Generation

> **Note**
> During the official deployment, please add a layer of your website's own permission verification to the server-side.

For security reasons, the backend obtains a temporary key, generates a direct upload URL, and signs it directly. You can refer to the Server Example for more information. The specific steps are as follows:

1. Obtain temporary keys using the STS SDK.
2. Generate the COS key and direct upload URL based on the file extension.
3. Sign the direct upload URL using a temporary key and return the direct upload URL, signature, token, and other information.

Server-side Configuration Steps:

1. Ensure that the keys, bucket, and region are properly configured.

```
var config = {
  // Obtain Tencent Cloud key, it is recommended to use the key of a sub-user with restricted permissions
https://console.cloud.tencent.com/cam/capi
  secretId: process.env.COS_SECRET_ID,
  secretKey: process.env.COS_SECRET_KEY,
  // Key validity period
  durationSeconds: 1800,
  // Enter the bucket and region here, for example: test-1250000000, ap-guangzhou
  bucket: process.env.PERSIST_BUCKET,
  region: process.env.PERSIST_BUCKET_REGION,
  // Restricted upload extensions
  extWhiteList: ['jpg', 'jpeg', 'png', 'gif', 'bmp'],
};
```

2. Terminal Execution

```
npm install
```

3. Enable the service

```
node app.js
```

At this point, the server-side has started successfully, and the client-side process can begin.

To use other languages or implement it on your own, please take the following steps:

1. Obtain temporary keys from the server. The server first uses the fixed keys SecretId and SecretKey to request temporary keys from the STS service, obtaining tmpSecretId, tmpSecretKey, and sessionToken. For more information, please refer to the Temporary Key Generation and Usage Guide or the cos-sts-sdk documentation.
2. Sign the direct upload URL to generate an authorization.
3. Return the direct upload URL, authorization, sessionToken, and other information. When the client uploads a file, place the obtained signature and sessionToken in the authorization and x-cos-security-token fields of the request header, respectively.

## Client Upload Example

### iOS Client Upload

1. Request direct upload and signature information from the server.

The server address here refers to the Node service started in the previous step.

```
// ext represents the file extension of the file to be uploaded, such as jpg, png, etc.
http://127.0.0.1:3000/sts-direct-sign?ext=jpg
```

iOS Request Code

```
NSURL *stsURL = [NSURL URLWithString:[NSString stringWithFormat:@"http://127.0.0.1:3000/sts-direct-sign?
ext=%@",@"file_extension"]];
NSMutableURLRequest * stsRequest = [NSMutableURLRequest requestWithURL:stsURL];
[stsRequest setHTTPMethod:@"GET"];
NSURLSession * session = [NSURLSession sharedSession];
NSURLSessionDataTask * task = [session dataTaskWithRequest:stsRequest completionHandler:^(NSData * _Nullable
data, NSURLResponse * _Nullable response, NSError * _Nullable error) {
    NSDictionary * dic = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingAllowFragments
error:nil];
    NSDictionary *params = dic[@"data"]; // This contains the signature information.
}];
[task resume];
```

2. Begin uploading files using the obtained signature information.

```
// params is the dic[@"data"] obtained from the previous step request
NSString * cosHost = [params objectForKey:@"cosHost"];
NSString * cosKey = [params objectForKey:@"cosKey"];
NSString * authorization = [params objectForKey:@"authorization"];
NSString * securityToken = [params objectForKey:@"securityToken"];

NSURL * stsURL = [NSURL URLWithString:[NSString stringWithFormat:@"https://%@/%@",cosHost,cosKey]];
NSMutableURLRequest * stsRequest = [NSMutableURLRequest requestWithURL:stsURL];
[stsRequest setHTTPMethod:@"PUT"];
[stsRequest setAllHTTPHeaderFields:@{
    @"Content-Type":@"application/octet-stream",
    @"Content-Length":@(data.length).stringValue,// data is the NSData of the file to be uploaded
    @"Authorization":authorization,
    @"x-cos-security-token":securityToken,
    @"Host":cosHost
}];
```

```
NSURLSession * session = [NSURLSession sessionWithConfiguration: [NSURLSessionConfiguration
defaultSessionConfiguration] delegate:self delegateQueue:nil];

NSURLSessionDataTask * task = [session uploadTaskWithRequest:stsRequest fromData:self.body
completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response, NSError * _Nullable error) {
    if(error){
        NSLog(@"Upload failed");
    } else {
        NSLog(@"Upload successful");
    }
}];
[task resume];
```

### Android Client Upload

1. Request direct upload and signature information from the server.

```
 /**
  * Obtain the direct upload URL and signature, etc.
  *
  * @param ext File extension: The backend will generate a COS key based on the extension after direct
upload.
  * @return Direct upload URL and signature, etc.
  */
 private JSONObject getStsDirectSign(String ext) {
     // Obtain the upload path and signature
     HttpURLConnection getConnection = null;
     try {
         // Direct upload signature service URL (For production environment, please replace with the
official direct upload signature service URL)
         URL url = new URL("http://127.0.0.1:3000/sts-direct-sign?ext=" + ext);
         getConnection = (HttpURLConnection) url.openConnection();
         getConnection.setRequestMethod("GET");

         int responseCode = getConnection.getResponseCode();
         if (responseCode == HttpURLConnection.HTTP_OK) {
             BufferedReader reader = new BufferedReader(new
InputStreamReader(getConnection.getInputStream()));
             StringBuilder stringBuilder = new StringBuilder();
             String line;
             while ((line = reader.readLine()) != null) {
                 stringBuilder.append(line);
             }
             reader.close();
             JSONObject jsonObject;
             try {
                 jsonObject = new JSONObject(stringBuilder.toString());
                 if(jsonObject.has("code") && jsonObject.optInt("code") == 0){
                     return jsonObject.optJSONObject("data");
                 } else {
                     Log.e(TAG, String.format("getStsDirectSign error code: %d, error message: %s",
jsonObject.optInt("code"), jsonObject.optString("message")));
                 }
             } catch (JSONException e) {
                 e.printStackTrace();
             }
         } else {
             Log.e(TAG, "getStsDirectSign HTTP error code: " + responseCode);
         }
     } catch (IOException e) {
         e.printStackTrace();
         Log.e(TAG, "getStsDirectSign Error sending GET request: " + e.getMessage());
```

```
    } finally {
        if (getConnection != null) {
            getConnection.disconnect();
        }
    }
    return null;
}
```

2. Begin uploading files using the obtained direct upload and signature information.

```
/**
 * Uploading files
 * @param filePath Local file path
 * @param listener Progress callback
 */
private void uploadFile(String filePath, final ProgressListener listener) {
    File file = new File(filePath);
    // Obtain direct upload signature and related data
    JSONObject directTransferData = getStsDirectSign(FilePathHelper.getFileExtension(file));
    if (directTransferData == null) {
        toastMessage("getStsDirectSign fail");
        return;
    }
    String cosHost = directTransferData.optString("cosHost");
    String cosKey = directTransferData.optString("cosKey");
    String authorization = directTransferData.optString("authorization");
    String securityToken = directTransferData.optString("securityToken");

    // Generate the upload URL
    URL url;
    try {
        url = new URL(String.format("https://%s/%s", cosHost, cosKey));
    } catch (MalformedURLException e) {
        e.printStackTrace();
        return;
    }

    HttpURLConnection conn = null;
    try {
        conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("PUT");
        conn.setDoOutput(true);

        // Set the request header
        conn.setRequestProperty("Content-Type", "application/octet-stream");
        conn.setRequestProperty("Content-Length", String.valueOf(file.length()));
        conn.setRequestProperty("Authorization", authorization);
        conn.setRequestProperty("x-cos-security-token", securityToken);
        conn.setRequestProperty("Host", cosHost);

        // Obtain the output stream
        OutputStream outputStream = conn.getOutputStream();

        // Read the file and upload it
        FileInputStream inputStream = new FileInputStream(file);
        byte[] buffer = new byte[BUFFER_SIZE];
        int bytesRead;
        long totalBytesRead = 0;
        while ((bytesRead = inputStream.read(buffer)) != -1) {
            outputStream.write(buffer, 0, bytesRead);
            totalBytesRead += bytesRead;
            if (listener != null) {
```

```
                listener.onProgress(totalBytesRead, file.length());
            }
        }
        // Close the stream
        inputStream.close();
        outputStream.flush();
        outputStream.close();

        // Obtain the response code
        int responseCode = conn.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
            // Uploaded successfully
        } else {
            Log.e(TAG, "uploadFile HTTP error code: " + responseCode);
        }
    } catch (IOException e) {
        e.printStackTrace();
        Log.e(TAG, "uploadFile Error sending PUT request: " + e.getMessage());
    } finally {
        if (conn != null) {
            conn.disconnect();
        }
    }
}
```

## Documentation

If you have more extensive API usage requirements, please refer to the following client SDK documentation:

- iOS SDK
- Android SDK

# HarmonyOS Direct Upload Practice

Last updated: 2025-09-19 10:21:11

## Overview

This documentation introduction explains how to directly upload files to a Cloud Object Storage (COS) bucket on the HarmonyOS system using simple code without relying on an SDK.

> **Notes:**
> Note: The content of this documentation is based on the XML version of the API.

## Prerequisites

- Log in to the COS console and create a bucket to get the Bucket (bucket name) and Region (region name). For details, see create a bucket.
- Log in to the Cloud Access Management console to obtain your project's SecretId and SecretKey.

## Practice Steps

Procedure:
1. The client calls the server API to input the file suffix. The server generates a cos key and a direct upload url based on the suffix and timestamp.
2. The server obtains a temporary key using the STS SDK.
3. The server signs the direct upload url with the obtained temporary key pair and returns the url, signature, token, etc.
4. After obtaining the information in procedure 3, the client directly initiates a put request and uploads with headers such as signature and token.

For specific code, see HarmonyOS sample code.

### Configuring the Server to Implement Signatures

> **Notes:**
> Add a layer of authority check on your website itself when the server is officially deployed.

For security reasons, the backend obtains the temporary key, generates a direct upload url, and directly signs it. See Server Signature Practice.
Procedure:
1. Obtain a temporary key using the STS SDK.
2. Generate a cos key and direct upload url based on the extension.
3. Sign the direct upload url with the temporary key and return the direct upload url, signature, token, etc.

**Server configuration procedure:**
1. Configure the key, bucket, and region.

```
var config = {
  // Get Tencent Cloud Key, recommend using a Sub-user key with limited permissions
https://console.cloud.tencent.com/cam/capi
  secretId: process.env.COS_SECRET_ID,
  secretKey: process.env.COS_SECRET_KEY,
  // key validity period
  durationSeconds: 1800,
  // Fill in the bucket and region here, for example: test-1250000000, ap-guangzhou
  bucket: process.env.PERSIST_BUCKET,
  region: process.env.PERSIST_BUCKET_REGION,
  // Upload extension limit
  extWhiteList: ['jpg', 'jpeg', 'png', 'gif', 'bmp'],
};
```

2. Execute in the terminal

```
npm install
```

3. Start the service.

```
node app.js
```

The server has started successfully here, and the client process can begin.
**For other languages or implementing your own, see the following process:**

1. Get a temporary key from the server. The server first uses the fixed key SecretId and SecretKey to obtain a temporary key from the STS service, getting the temporary key tmpSecretId, tmpSecretKey, and sessionToken. For details, see temporary key generation and usage guide or cos-sts-sdk.
2. Sign the direct upload url and generate authorization.
3. Return the direct upload url, authorization, sessionToken, etc. When uploading files, the client places the obtained signature and sessionToken in the authorization and x-cos-security-token fields of the request header.

## HarmonyOS Upload Example

1. Request direct upload and signature information from the server.

```
import http from '@ohos.net.http';

/**
 Get the direct upload url and signature
 *
 * @param ext File suffix. The backend generates a cos key based on the suffix for direct upload.
 * @returns Direct upload url and signature
 */
public static async getStsDirectSign(ext: string): Promise<Object> {
  // Each httpRequest corresponds to an HTTP request task and cannot be reused
  let httpRequest = http.createHttp();
  //Direct upload signature business server url (official environment, replace with official direct upload
signature business url)
  // For server-side code example of direct upload signature business, see:
https://github.com/tencentyun/cos-demo/blob/main/server/direct-sign/nodejs/app.js
  let url = "http://127.0.0.1:3000/sts-direct-sign?ext=" + ext;
  try {
    let httpResponse = await httpRequest.request(url, { method: http.RequestMethod.GET });
    if (httpResponse.responseCode == 200) {
      let result = JSON.parse(httpResponse.result.toString())
      if (result.code == 0) {
        return result.data;
      } else {
        console.info(`getStsDirectSign error code: ${result.code}, error message: ${result.message}`);
      }
    } else {
      console.info("getStsDirectSign HTTP error code: " + httpResponse.responseCode);
    }
  } catch (err) {
    console.info("getStsDirectSign Error sending GET request: " + JSON.stringify(err));
  } finally {
    // When the request is complete, call destroy to terminate.
    httpRequest.destroy();
  }
  return null;
}
```

2. Use the obtained direct upload and signature information to start uploading files.

```
import http from '@ohos.net.http';
```

```
import promptAction from '@ohos.promptAction'
import fs from '@ohos.file.fs';
import request from '@ohos.request';
import common from '@ohos.app.ability.common';
import { MediaBean } from '../bean/MediaBean';
import { MediaHelper } from './MediaHelper';

/**
 * Upload files (implemented via uploadTask)
 * @param context context
 * @param media Media file
 */
public static async uploadFileByTask(context: common.Context, media: MediaBean, progressCallback:
(uploadedSize: number, totalSize: number) => void) {
  // Get the direct upload signature and data
  let ext = MediaHelper.getFileExtension(media.fileName);
  let directTransferData: any = await UploadHelper.getStsDirectSign(ext);
  if (directTransferData == null) {
    promptAction.showToast({ message: 'getStsDirectSign fail' });
    return;
  }

  // Upload information returned by the server
  let cosHost: String = directTransferData.cosHost;
  let cosKey: String = directTransferData.cosKey;
  let authorization: String = directTransferData.authorization;
  let securityToken: String = directTransferData.securityToken;

  // Generate the uploaded url
  let url = `https://${cosHost}/${cosKey}`;
  try {
    // Copy the uri file to cacheDir (because request.uploadFile only accepts internal: paths)
    let file = await fs.open(media.fileUri, fs.OpenMode.READ_ONLY);
    let destPath = context.cacheDir + "/" + media.fileName;
    await fs.copyFile(file.fd, destPath);
    let realuri = "internal://cache/" + destPath.split("cache/")[1];

    let uploadConfig = {
      url: url,
      header: {
        "Content-Type": "application/octet-stream",
        "Authorization": authorization,
        "x-cos-security-token": securityToken,
        "Host": cosHost
      },
      method: "PUT",
      files: [{ filename: media.fileName, name: "file", uri: realuri, type: ext }],
      data: []
    };
    // Start uploading
    let uploadTask = await request.uploadFile(context, uploadConfig)

    uploadTask.on('progress', progressCallback);
    let upCompleteCallback = (taskStates) => {
      for (let i = 0; i < taskStates.length; i++) {
        promptAction.showToast({ message: 'upload succeeded' });
        console.info("upOnComplete taskState:" + JSON.stringify(taskStates[i]));
      }
    };
    uploadTask.on('complete', upCompleteCallback);

    let upFailCallback = (taskStates) => {
```

```
      for (let i = 0; i < taskStates.length; i++) {
        promptAction.showToast({ message: 'upload failed' });
        console.info("upOnFail taskState:" + JSON.stringify(taskStates[i]));
      }
    };
    uploadTask.on('fail', upFailCallback);
  } catch (err) {
    console.info("uploadFile Error sending PUT request: " + JSON.stringify(err));
    promptAction.showToast({ message: "uploadFile Error sending PUT request: " + JSON.stringify(err) });
  }
}
```

# Direct Upload Practice in Flutter

Last updated: 2023-09-12 18:31:58

## Feature Overview

This document explains how to upload files directly to a Cloud Object Storage (COS) bucket in Flutter without relying on an SDK, using simple code.

> **Note:**
> This document is based on the XML APIs.

## Preparations

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 refer to the Creating Bucket document.
2. Log in to the Access Management Console to obtain your project's SecretId and SecretKey.

## Practical Steps

The overall process is as follows:

1. The client calls the server-side interface and passes the file extension. The server then generates a COS key and direct upload URL based on the extension and timestamp.
2. The server obtains temporary keys through the STS SDK.
3. The server uses the obtained temporary key to sign the direct upload URL and returns the URL, signature, token, and other information.
4. After obtaining the information from step 3, the client initiates a PUT request with the signature, token, and other headers for the upload.

   For the specific code, refer to the Flutter Example.

## Server Side

> **Note:**
> In official deployment, add a layer of permission check of your website.

For security reasons, the backend obtains a temporary key, generates a direct upload URL, and signs it directly. You can refer to the Server Example for more information. The specific steps are as follows:

1. Obtain temporary keys using the STS SDK.
2. Generate the COS key and direct upload URL based on the file extension.
3. Sign the direct upload URL using a temporary key and return the direct upload URL, signature, token, and other information.

Server-side Configuration Steps:

1. Ensure that the keys, bucket, and region are properly configured.

```
var config = {
  // Obtain Tencent Cloud key, it is recommended to use the key of a sub-user with restricted permissions
https://console.cloud.tencent.com/cam/capi
  secretId: process.env.COS_SECRET_ID,
  secretKey: process.env.COS_SECRET_KEY,
  // Key validity period
  durationSeconds: 1800,
  // Enter the bucket and region here, for example: test-1250000000, ap-guangzhou
  bucket: process.env.PERSIST_BUCKET,
  region: process.env.PERSIST_BUCKET_REGION,
  // Restricted upload extensions
  extWhiteList: ['jpg', 'jpeg', 'png', 'gif', 'bmp'],
};
```

2. Terminal Execution

```
npm install
```

3. Enable the service

```
node app.js
```

At this point, the server-side has started successfully, and the client-side process can begin.
To use other languages or implement it on your own, please take the following steps:

1. Obtain temporary keys from the server. The server first uses the fixed keys SecretId and SecretKey to request temporary keys from the STS service, obtaining tmpSecretId, tmpSecretKey, and sessionToken. For more information, please refer to the Temporary Key Generation and Usage Guide or the cos-sts-sdk documentation.
2. Sign the direct upload URL to generate an authorization.
3. Return the direct upload URL, authorization, sessionToken, and other information. When the client uploads a file, place the obtained signature and sessionToken in the authorization and x-cos-security-token fields of the request header, respectively.

## Client (Flutter)

For the specific code, refer to the Flutter Example.

### Using Dio network library

1. Request direct upload and signature information from the server.

```
/// Obtain the direct upload URL and signature, etc.
/// @param ext File extension. The backend will generate a COS key based on the extension after direct
upload.
/// @return Direct upload URL and signature, etc.
static Future<Map<String, dynamic>> getStsDirectSign(String ext) async {
  Dio dio = Dio();
  // Direct upload signature service URL (Replace with the official direct upload signature service URL in a
production environment)
  // For an example of direct upload signature server-side code, refer to: https://github.com/tencentyun/cos-
demo/blob/main/server/direct-sign/nodejs/app.js
  // 10.91.22.16 is the address of the direct upload signature application server, such as the aforementioned
Node service. In short, it is the URL to access the direct upload signature application server.
  Response response = await dio.get('http://10.91.22.16:3000/sts-direct-sign',
      queryParameters: {'ext': ext});
  if (response.statusCode == 200) {
    if (kDebugMode) {
      print(response.data);
    }
    if (response.data['code'] == 0) {
      return response.data['data'];
    } else {
      throw Exception(
          'getStsDirectSign error code: ${response.data['code']}, error message:
${response.data['message']}');
    }
  } else {
    throw Exception(
        'getStsDirectSign HTTP error code: ${response.statusCode}');
  }
}
```

2. Begin uploading files using the obtained direct upload and signature information.

```
/// Upload file
/// @param filePath File path
/// @param progressCallback Progress callback
static Future<void> upload(String filePath, ProgressCallback progressCallback) async {
```

```
  String ext = path.extension(filePath).substring(1);
  Map<String, dynamic> directTransferData;
  try {
    directTransferData = await getStsDirectSign(ext);
  } catch (err) {
    if (kDebugMode) {
      print(err);
    }
    throw Exception("getStsDirectSign fail");
  }
  String cosHost = directTransferData['cosHost'];
  String cosKey = directTransferData['cosKey'];
  String authorization = directTransferData['authorization'];
  String securityToken = directTransferData['securityToken'];
  String url = 'https://$cosHost/$cosKey';
  File file = File(filePath);
  Options options = Options(
    method: 'PUT',
    headers: {
      'Content-Length': await file.length(),
      'Content-Type': 'application/octet-stream',
      'Authorization': authorization,
      'x-cos-security-token': securityToken,
      'Host': cosHost,
    },
  );
  try {
    Dio dio = Dio();
    Response response = await dio.put(url,
        data: file.openRead(),
        options: options, onSendProgress: (int sent, int total) {
          double progress = sent / total;
          if (kDebugMode) {
            print('Progress: ${progress.toStringAsFixed(2)}');
          }
          progressCallback(sent, total);
        });
    if (response.statusCode == 200) {
      if (kDebugMode) {
        print('Uploaded successfully');
      }
    } else {
      throw Exception("Upload failed ${response.statusMessage}");
    }
  } catch (error) {
    if (kDebugMode) {
      print('Error: $error');
    }
    throw Exception("Upload failed ${error.toString()}");
  }
}
```

### Using the native HTTP Client library

1. Request direct upload and signature information from the server.

```
/// Obtain the direct upload URL and signature, etc.
/// @param ext File extension. The backend will generate a COS key based on the extension after direct
upload.
/// @return Direct upload URL and signature, etc.
static Future<Map<String, dynamic>> _getStsDirectSign(String ext) async {
  HttpClient httpClient = HttpClient();
```

```
  // Direct upload signature service URL (Replace with the official direct upload signature service URL in a
production environment)
  // For an example of direct upload signature server-side code, refer to: https://github.com/tencentyun/cos-
demo/blob/main/server/direct-sign/nodejs/app.js
  // 10.91.22.16 is the address of the direct upload signature application server, such as the aforementioned
Node service. In short, it is the URL to access the direct upload signature application server.
  HttpClientRequest request = await httpClient
      .getUrl(Uri.parse("http://10.91.22.16:3000/sts-direct-sign?ext=$ext"));
  HttpClientResponse response = await request.close();
  String responseBody = await response.transform(utf8.decoder).join();
  if (response.statusCode == 200) {
    Map<String, dynamic> json = jsonDecode(responseBody);
    if (kDebugMode) {
      print(json);
    }
    httpClient.close();
    if (json['code'] == 0) {
      return json['data'];
    } else {
      throw Exception(
          'getStsDirectSign error code: ${json['code']}, error message: ${json['message']}');
    }
  } else {
    httpClient.close();
    throw Exception(
        'getStsDirectSign HTTP error code: ${response.statusCode}');
  }
}
```

2. Begin uploading files using the obtained direct upload and signature information.

```
/// Upload file
/// @param filePath File path
/// @param progressCallback Progress callback
static Future<void> upload(String filePath, ProgressCallback progressCallback) async {
  // Obtain direct upload signature and related information
  String ext = path.extension(filePath).substring(1);
  Map<String, dynamic> directTransferData;
  try {
    directTransferData = await _getStsDirectSign(ext);
  } catch (err) {
    if (kDebugMode) {
      print(err);
    }
    throw Exception("getStsDirectSign fail");
  }

  String cosHost = directTransferData['cosHost'];
  String cosKey = directTransferData['cosKey'];
  String authorization = directTransferData['authorization'];
  String securityToken = directTransferData['securityToken'];
  String url = 'https://$cosHost/$cosKey';

  File file = File(filePath);
  int fileSize = await file.length();
  HttpClient httpClient = HttpClient();
  HttpClientRequest request = await httpClient.putUrl(Uri.parse(url));
  request.headers.set('Content-Type', 'application/octet-stream');
  request.headers.set('Content-Length', fileSize.toString());
  request.headers.set('Authorization', authorization);
  request.headers.set('x-cos-security-token', securityToken);
  request.headers.set('Host', cosHost);
```

```
  request.contentLength = fileSize;
  Stream<List<int>> stream = file.openRead();
  int bytesSent = 0;
  stream.listen(
        (List<int> chunk) {
      bytesSent += chunk.length;
      double progress = bytesSent / fileSize;
      if (kDebugMode) {
        print('Progress: ${progress.toStringAsFixed(2)}');
      }
      progressCallback(bytesSent, fileSize);
      request.add(chunk);
    },
    onDone: () async {
      HttpClientResponse response = await request.close();
      if (response.statusCode == 200) {
        if (kDebugMode) {
          print('Uploaded successfully');
        }
      } else {
        throw Exception("Upload failed $response");
      }
    },
    onError: (error) {
      if (kDebugMode) {
        print('Error: $error');
      }
      throw Exception("Upload failed ${error.toString()}");
    },
    cancelOnError: true,
  );
}
```

## Documentation

If you have more extensive API usage requirements, please refer to the Flutter SDK.

# Data Security
# Best Practices for Protecting COS SecretKey with KMS White-Box Keys

Last updated: 2023-09-12 18:32:37

## KMS White-Box Key Overview

KMS White-Box Keys are utilized to safeguard sensitive root key information on the client side, such as API SecretKeys, authentication keys or tokens used by internal user systems, and other local sensitive root key data. This ensures a comprehensive, end-to-end protection without exposing sensitive key information in plaintext. By obfuscating and integrating the algorithm with the key, the key information is effectively protected through lookup tables, enabling encryption and decryption without revealing any keys. Additionally, the security of the keys is further ensured through device binding.

This article introduces an example of using Key Management Service (KMS) White-Box Keys to encrypt and decrypt SecretKeys for the COS product. By employing White-Box Keys, the SecretKey is protected and securely utilized. For detailed steps, please refer to:

- Key Management and Distribution
- Accessing COS via Object Storage SDK Tool

## Advantages of Using KMS White-Box Keys

**High Security**
Employing high-strength obfuscation and reinforcement algorithms along with multi-layer security protection technologies, the exposure of API SecretKey in plaintext within the source code is effectively eliminated.
**Dynamic White-Box**
Supports dynamic key rotation with flexibility while maintaining the integrity of the white-box library.
**Device Binding**
Bind device fingerprint information to reinforce the protection of decryption keys.
**Algorithm Support**
Supports international AES and Chinese cryptographic SM4 algorithms, meeting compliance requirements.

## Key management and distribution

### Step 1. Create a white-box key

> **Note**
> - White-Box Keys are a chargeable feature of KMS. For more information, please see KMS Billing Overview and KMS Purchase Methods.
> - Creating a white-box key pair is achieved by invoking the white-box service. Both console and API methods are supported. This example uses the console method.

1. Log in to the Key Management System (Compliant) Console, select **White-box Key Management** > **User Key Management** from the left menu, switch the "Region" according to your business needs, and click **Create**.
2. In the pop-up dialog box, enter the white-box key name, select the encryption algorithm, and provide the description and tags (both optional). Click **Confirm** to complete the creation of the white-box key.

### Step 2. Obtain the API SecretKey from the console

1. Log in to the API Key Management Console with your root account to view your API keys.
2. In the key operation column, click **Show**, complete the identity verification, and obtain and copy the SecretKey.

### Step 3. Perform base64 encoding on the plaintext SecretKey

Encode the SecretKey content obtained in Step 2 using base64. For example, if the plaintext SecretKey is `lY9Ynrabcdj05YH1234LE37xxxx`, use the openssl command to generate the base64-encoded result: `bFk5WW5yYWJjZGowNVlIMTIzNExFMzcwSE9Nxxxx`.

```
echo lY9Ynrabcdj05YH1234LE37xxxx | openssl base64
```

### Step 4. Encrypt the API SecretKey using the white-box key

1. Log in to the Key Management System (Compliant) Console. In the white-box key list, click on the "White-box Key ID/Name" or **Encrypt** in the "Operation" column.
2. In the pop-up dialog, fill in the encoded content obtained in Step 3 into the plaintext (base64) text box, and click **White-box Encryption**.

Upon successful encryption, a randomly generated Initialization Vector (IV) and the encrypted ciphertext will be returned. Click **Download IV** and **Download Ciphertext** to complete the content download.

> **Note**
> Both the initialization vector (abbreviated as IV) and the encrypted ciphertext have been base64 encoded.

3. (Optional Step) White-Box Keys support binding the decryption key to the physical environment where the decryption is executed. The procedure is as follows:
   - Download the fingerprint collection tool for the corresponding environment from the White-Box Key Console, and run the fingerprint collection tool in the specified environment to obtain the fingerprint string for that environment.
   - In the White-Box Key Console, select the specified White-Box Key, click "Add Device Fingerprint," input the fingerprint string for the corresponding environment, and bind the fingerprint with the key.
   - Once the fingerprint environment variable is bound to the key, the decryption key will be updated. Subsequently, data encrypted with this White-Box Key can only be decrypted using the updated decryption key within the bound physical environment.

Download Fingerprint Collection Tool:
Bind the fingerprint string with the specified White-Box Key:

### Step 5. Download the decryption key

1. Log in to the Key Management System (Compliance) Console. In the white-box key list, click "White-box Key ID/Name" to access the key's basic information page.
2. On the Key Basic Information page, click **Download Decryption Key** and name it decrypt_key_sm4.bin.

### Step 6: Download the decryption SDK file

1. Log in to the Key Management System (Compliance) Console. In the White-Box Key list, click on **Download Decryption SDK File** on the right side.
2. In the pop-up window, based on the programming language of each business system, select and download the corresponding decryption SDK, and integrate the decryption SDK into the business system.

### Step 7: Distribute white-box decryption key and API SecretKey ciphertext

Administrators distribute the decryption key, IV, and ciphertext files obtained from the above steps to the developers or operations personnel of each business system. The decryption key is deployed to the corresponding business system's files, while the initialization vector (IV) and ciphertext serve as parameters for the decryption SDK.

> **Note**
> The downloaded decryption key is a binary .bin file, which needs to be placed on the same server as the executable file (already integrated with the decryption SDK). The file path will serve as the decryption parameter for the decryption SDK.

## Accessing COS using the Object Storage SDK Tool

### Step 1: Initialize Identity Information Using Permanent Key

After installing the Object Storage SDK tool, import the downloaded White-Box SDK into your client-side code and follow the steps below to obtain the decrypted Secret Key:

1. Incorporate the pre-downloaded White-Box SDK into your code.
2. Invoke the decryption function of the White-Box Key Decryption SDK to decrypt the encrypted Secret Key. Pass in the following parameters to obtain the decrypted plaintext.

- decrypt_key_bin_dir: The directory where the decryption key is stored in Step 7.
- decrypt_key_sm4.bin: The decryption key downloaded in Step 5, corresponding to the file name.
- InitializationVector: The IV downloaded in Step 4.
- CipherText: The encrypted SecretKey ciphertext from Step 4 using White-Box encryption.
- algorithmType: The algorithm type used when generating the key, with values of 0 or 1. A value of 0 represents AES_256, while 1 represents SM4.

3. Initialize the COS SDK using the decrypted Secret Key and Secret ID (if the Secret ID is also encrypted, it needs to be decrypted as well). For subsequent service calls related to COS, please refer to the relevant documentation of the COS SDK.

The White-Box Key Decryption SDK is implemented in C language and offers adaptations for other high-level languages, such as Golang, Python, and Java. For specific usage methods, please refer to the code examples in the decryption SDK for the designated language.
The following Golang sample code is based on CGO:
Since the underlying functionality of the White-Box Decryption SDK is implemented by the C lib library, the example code uses CGO annotations to import the decryption SDK header file directory `#cgo CFLAGS: -I${header_file_dir_path}` and the lib library file `#cgo LDFlags: -L${lib_dir_path} -l${library_name}` into the code. The header file directory and lib library are typically located in the root directory of the downloaded White-Box Decryption SDK and can be adjusted according to the project structure during application integration.

```
/*
#cgo CFLAGS: -I${SRCDIR}/include
#cgo LDFLAGS: -L${SRCDIR}/lib -lydwbcrypto

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "wrp.h"

static int clt_load_key(WRP_KEY_CTX *key_ctx, char *decrypt_key_path, const char *file_name,
        uint32_t mode, uint32_t algoType) {
    printf("begin to load key in dir: %s\n", decrypt_key_path);
    printf("begin to load key: %s\n", file_name);
    int err;
    const WRP_KEY* keyalg = WRP_KEY_wbaes();
    if (algoType == 1) { keyalg = WRP_KEY_wbsm4(); }

    err = WRP_KEY_init(key_ctx, keyalg, 0);
    if (err) { printf("Err: orig_byte init\n"); goto end; }

    err = WRP_KEY_ctrl(key_ctx, WRP_KEY_CTRL_WB_SET_PATH, decrypt_key_path, strlen(decrypt_key_path));
    if (err) { printf("Err: set path\n"); goto end; }

    err = WRP_KEY_import(key_ctx, file_name, mode);
    if (err) { printf("Err: WRP_KEY_import error: %d, \n", err); goto end; }

end:
    return err;
}


int clt_cbc_dec(char * file_dir, const char * file_name, uint8_t* ciph, uint32_t ciphlen, char* iv,
        uint32_t algoType, uint8_t* out, uint32_t* outlen) {
    WRP_KEY_CTX *mykey;
    WRP_CIPHER_CTX *myaes = NULL;

    uint32_t bits;
    ERRNO err = ERRNO_OK;

    const WRP_CIPHER* alg = WRP_wbaes_cbc();
    if (algoType == 1) { alg = WRP_wbsm4_cbc(); }
```

```
    // IO: load wbkey
    mykey = WRP_KEY_CTX_new();
    err = clt_load_key(mykey, file_dir, file_name, KEYMODE_DECRYPT, algoType);
    if (err) { printf("load whitebox key error: %d\n", err); goto cleanup; }

    printf("load key success\n");
    bits = WRP_KEY_key_len(mykey, KEYLEN_TYPE_BITS);
    printf("key len=%u\n", bits);

    if (bits == 0) { printf("get whitebox key length error\n"); err=-1; goto cleanup; }

    // crypto
    myaes = WRP_CIPHER_CTX_new();

    err = WRP_CIPHER_Decrypt_init(myaes, alg, mykey, (uint8_t *)iv);
    if (err) { printf("Dec: init Err**\n"); goto cleanup; }
    printf("decrypt init success, begin to decrypt cipher\n");

    err = WRP_CIPHER_Decrypt_doCipher(myaes, ciph, ciphlen, out, outlen);
    if (err) { printf("decrypt error: %.8X\n", err); goto cleanup; }
    //printf("decrypt success\n");
    //printf("output: %s\n", deout);
    //printf("\n");

cleanup:
    WRP_KEY_CTX_free(mykey);
    WRP_CIPHER_CTX_free(myaes);
    return err;
}

void set_fingerprint_env_name(char *fingerprint_env_name) {
    WRP_getENV_KEY((uint8_t *)fingerprint_env_name, strlen(fingerprint_env_name));
}

*/
import “C”

// Decrypt ...
func Decrypt(input DecryptionInput) ([]byte, error) {
    if input.KeyDir == "" || input.KeyFileName == "" || input.CipherText == "" || input.InitializationVector
== "" {
        return nil, errors.New("invalid white-box decryption input")
    }
    cipher, err := base64.StdEncoding.DecodeString(input.CipherText)
    if err != nil {
        return nil, fmt.Errorf("failed to decode cipher text: %w", err)
    }
    iv, err := base64.StdEncoding.DecodeString(input.InitializationVector)
    if err != nil {
        return nil, fmt.Errorf("failed to decode initialization vector: %w", err)
    }

    decryptionKeyDir := C.CString(input.KeyDir)
    defer C.free(unsafe.Pointer(decryptionKeyDir))

    decryptionKeyFileName := C.CString(input.KeyFileName)
    defer C.free(unsafe.Pointer(decryptionKeyFileName))

    // malloc a buffer which size is a little greater than your plaintext
    outLen := len(input.CipherText) + 1024
    outBuf := C.malloc(C.size_t(outLen))
```

```
    defer C.free(unsafe.Pointer(outBuf))

    errC := C.clt_cbc_dec(
        decryptionKeyDir, decryptionKeyFileName,
        (*C.uchar)(unsafe.Pointer(&cipher[0])),
        C.uint(len(cipher)),
        (*C.char)(unsafe.Pointer(&iv[0])),
        C.uint(input.AlgorithmType),
        (*C.uchar)(outBuf),
        (*C.uint)(unsafe.Pointer(&outLen)),
    )
    if errC != 0 {
        return nil, fmt.Errorf("whitebox decryption failed with code %d", int(errC))
    }
    plain := C.GoBytes(outBuf, C.int(outLen))
    return plain, nil
}

func main() {
    // Replace <bucket> and <region> with the actual information
    // The naming convention for buckets is {name}-{appid}; the bucket name provided here must follow this
format.
    u, _ := url.Parse("https://<bucket>.cos.<region>.myqcloud.com")
    // Specify White-Box Key related information, as well as the encrypted secret ID and secret key, using
environment variables.
    encryptionAlgorithm, _ := strconv.Atoi(os.Getenv("EncryptionAlgorithm"))
    whiteboxDecryptInput := DecryptionInput{
        KeyDir:               os.Getenv("WhiteboxKeyDir"),      // e.g.: /usr/local/
        KeyFileName:          os.Getenv("WhiteboxKeyFileName"), // e.g.: whitebox_key.bin
        InitializationVector: os.Getenv("WhiteboxDecryptionIV"),
        AlgorithmType:        uint(encryptionAlgorithm),
    }
    // Decrypting Secret ID using White-Box SDK
    whiteboxDecryptInput.CipherText = os.Getenv("COS_Encrypted_Secret_ID")
    secretID, err := Decrypt(whiteboxDecryptInput)
    if err != nil {
        panic(err)
    }
    // Decrypt SecretKey using White-Box SDK
    whiteboxDecryptInput.CipherText = os.Getenv("COS_Encrypted_Secret_Key")
    secretKey, err := Decrypt(whiteboxDecryptInput)
    if err != nil {
        panic(err)
    }

    b := &cos.BaseURL{BucketURL: u}
    c := cos.NewClient(b, &http.Client{
        // Set timeout duration
        Timeout: 100 * time.Second,
        Transport: &cos.AuthorizationTransport{
            // Fill in the account and key information accurately, or set them as environment variables
            SecretID:  string(secretID),
            SecretKey: string(secretKey),
        },
    })

    name := "test/hello.txt"
    resp, err := c.Object.Get(context.Background(), name, nil)
    if err != nil {
        panic(err)
    }
    bs, _ := ioutil.ReadAll(resp.Body)
```

```
    _ = resp.Body.Close()
    fmt.Printf("%s\n", string(bs))
}

type DecryptionInput struct {
    KeyDir               string
    KeyFileName          string
    InitializationVector string // Base64 encoded.
    CipherText           string // Base64 encoded.
    AlgorithmType        uint   // 0: AES_256, 1: SM4.
}
```

### Step 2: Directly use COS SDK to request COS services

After initialization, you can directly use the COS SDK tool for basic operations such as uploading and downloading without manually generating signatures like API requests. This is because the SDK tool generates signatures on your behalf using the key and initiates requests to COS.

# Introduction to COS Data Security Solution

Last updated: 2023-09-20 11:21:48

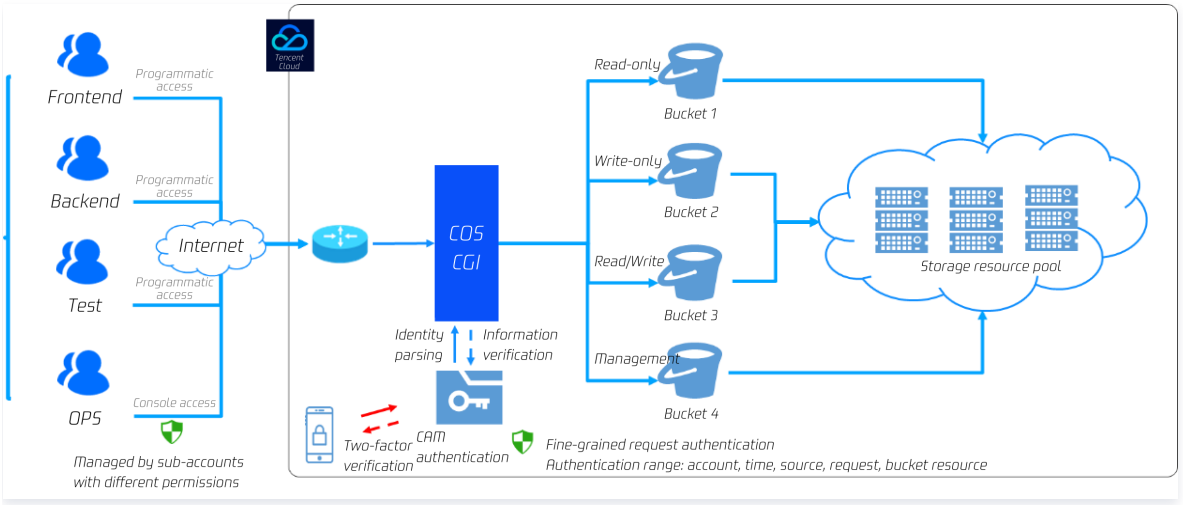
## Pre-Event Protection

### 1. Permission Isolation

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:

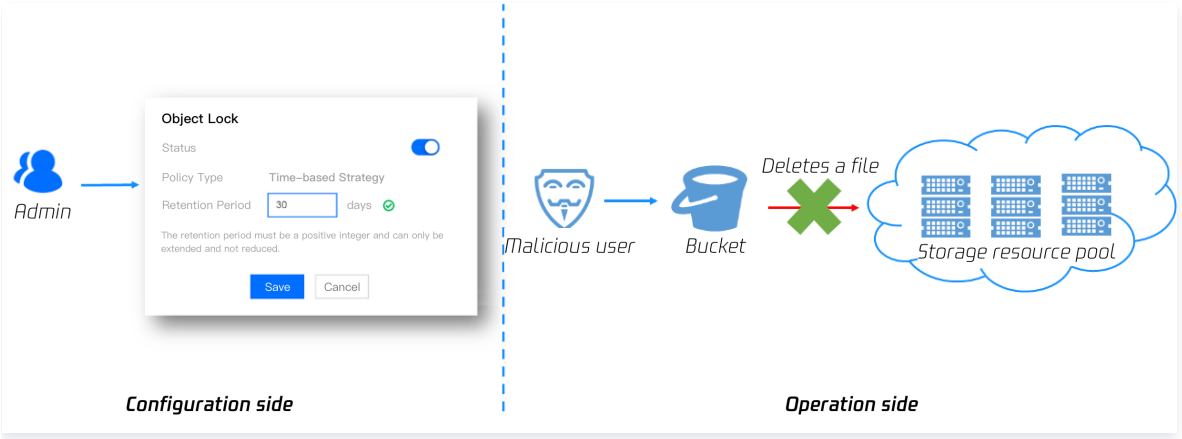
- Using the Tencent Cloud root account for daily operations.
- Creating sub-accounts for employees, but granting excessive permissions.
- Lack of access condition control for high-privilege sub-account users and high-risk operations.
- Failing to regularly audit user permissions and login information.
- No regulation for permission management

Tencent Cloud Access Management (CAM) ensures clear, secure, and controllable permissions through various measures such as account hierarchy and permission hierarchy. In terms of account hierarchy, the root account can grant different access permissions, such as programmatic access and console access, to all legitimate CAM users, including sub-accounts and collaborators. In terms of permission hierarchy, different levels of authorization, such as service level, interface level, and resource level, are used to specify **under what conditions, through which methods, and which resources can be accessed by CAM users**.
For example, you can create sub-accounts within the root account and assign resource management permissions to these sub-accounts without sharing the root account's credentials. Additionally, you can grant different access permissions to different personnel for various resources. For instance, you can allow certain sub-accounts to have read access to a Cloud Object Storage (COS) bucket, while other sub-accounts or the root account can have write access to a specific COS object. Resources, access permissions, and users can be bundled in bulk, enabling fine-grained permission management.
For high-risk operations (such as data deletion), permissions can be separated and granted, allowing users to perform actions only through the console while enabling multi-factor authentication (MFA) for secondary verification. When MFA is enabled, users will be prompted to verify via a text message code when executing such high-risk operations.




### 2. Object Locking

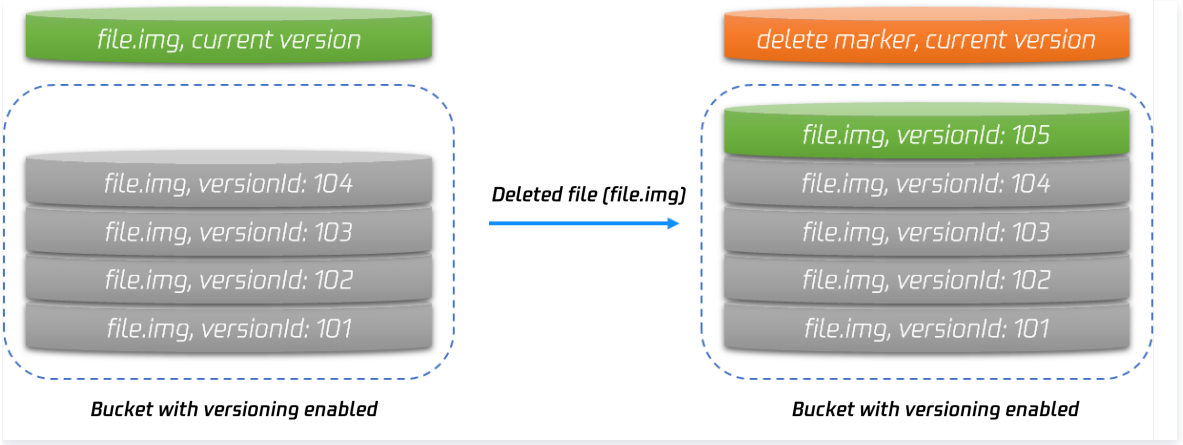
Users can enable the object locking feature for sensitive and essential data (e.g., financial transaction data and medical image data) to prevent them from being deleted or modified. After the object locking feature is enabled, all data in the bucket can only be read and cannot be overwritten or deleted during the effective period. This setting applies to all CAM users (including the root account) and anonymous users.




## 3. Data Disaster Recovery

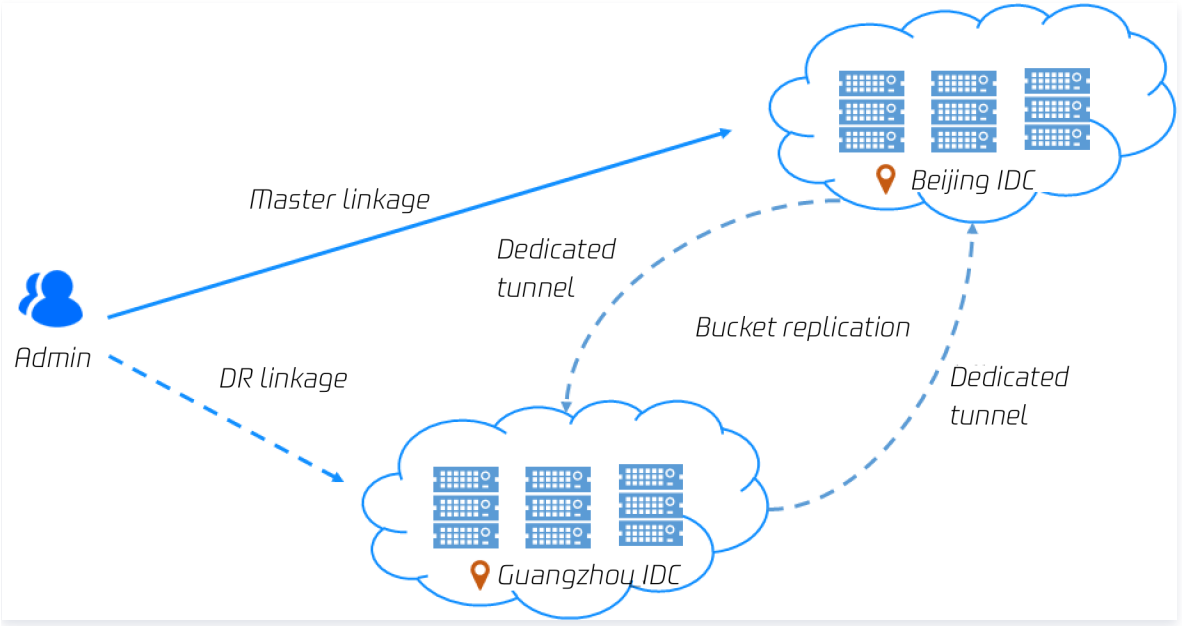
Tencent Cloud COS offers a wide range of data management capabilities, including data encryption, version control, bucket replication, and lifecycle management. Sensitive files can be protected through encryption to ensure secure data read and write operations. Version control and bucket replication enable disaster recovery across different locations, further ensuring data durability and allowing data recovery from backup sites in case of accidental or malicious deletion. Lifecycle management helps in data archiving and deletion, reducing storage costs.

The version control feature ensures that user files are not overwritten or deleted. Once version control is enabled, all write operations on files with the same name are treated as adding new versions of the same file, while delete operations are equivalent to adding a delete marker. You can access any past version of the data by specifying the version number, enabling data rollback and mitigating the risks of accidental deletion and overwriting.

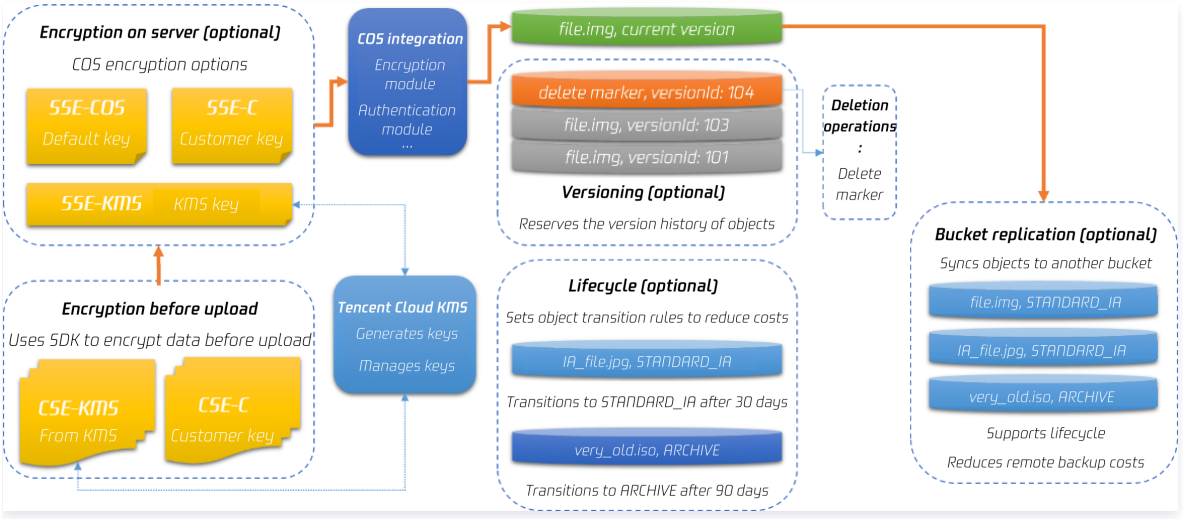



Additionally, object storage offers bucket replication functionality, which helps users replicate incremental files to data centers in other cities via dedicated lines, achieving offsite disaster recovery. When data in the primary bucket is deleted, it can be restored from the

backup bucket using batch copying.

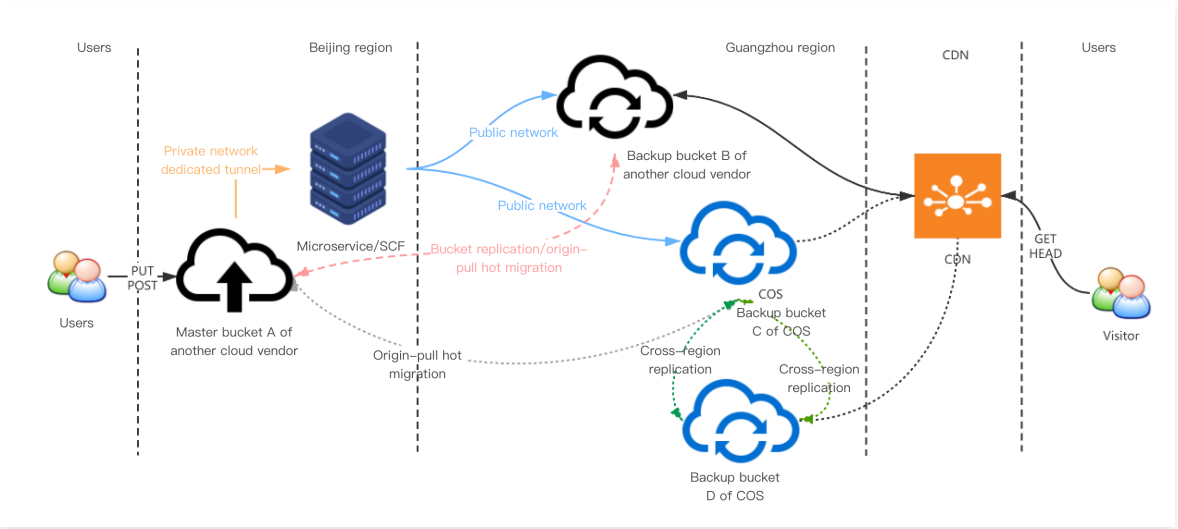



Considering that version control and bucket replication may increase the number of files, users can also use the lifecycle feature to transition backup data to lower-cost storage types, such as infrequent access or archival storage, to achieve cost-effective cold backups. By integrating data encryption, version control, bucket replication, and lifecycle features, Tencent Cloud Object Storage provides a comprehensive cold backup solution, as shown in the following diagram.



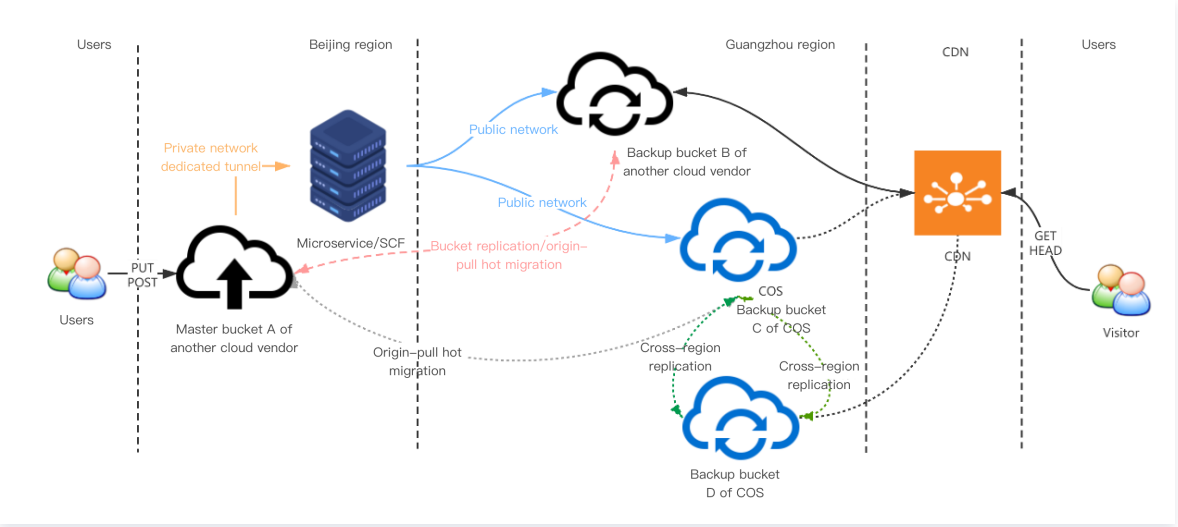

For customers who primarily store data with other cloud providers (such as AWS or OSS) and have stringent requirements for data persistence, COS also offers a multi-cloud disaster recovery solution based on cloud functions. First, data is stored with other cloud providers, and customers can trigger data synchronization or bucket replication through cloud functions to achieve offsite disaster recovery and ensure data persistence. Simultaneously, data migration is triggered through cloud functions to back up core data to Tencent Cloud's object storage service, and offsite disaster recovery is achieved through Tencent Cloud's bucket replication feature. Finally, Tencent Cloud's permission control manages data access rights for COS, ensuring that data can be restored from Tencent

Cloud COS in extreme situations.




## Mid-Event Monitoring

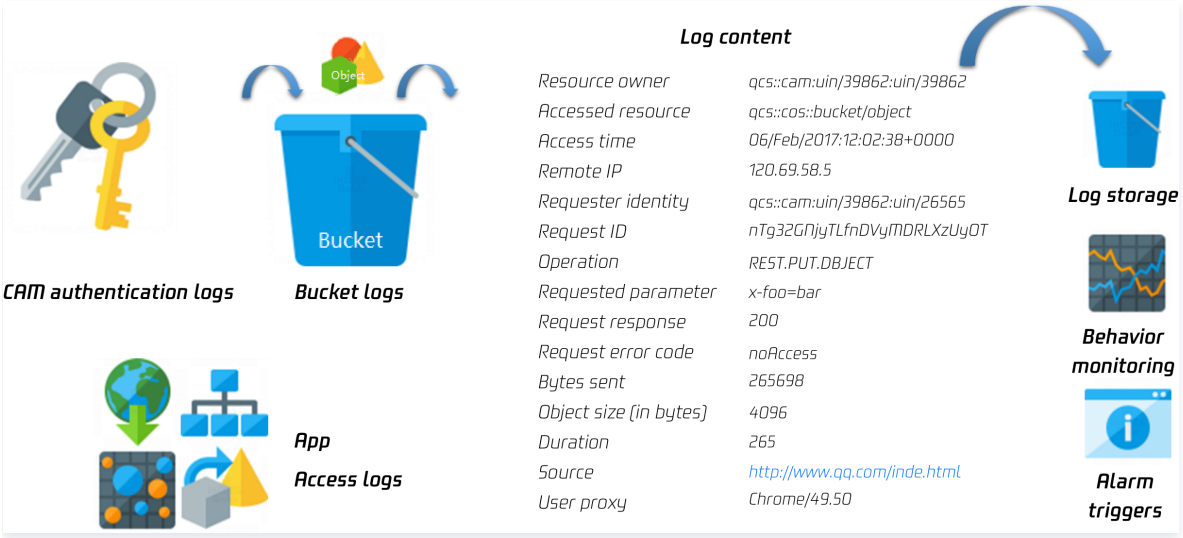
Tencent Cloud COS provides event notification features based on cloud functions. For high-risk operations such as file deletion, you can configure SCF for DeleteObject and other high-risk operations. When a high-risk operation occurs, notifications will be sent immediately to your email or mobile phone, ensuring timely detection of high-risk behavior and taking measures to stop it.




## Post-Event Tracing

Tencent Cloud COS provides users with accessible log monitoring and auditing features through multiple channels. For user access logs of storage buckets, operations such as deleting files (DeleteObject), overwriting files (PutObjectCopy), and modifying file permissions (PutObjectACL) can be tracked using the bucket access log feature, making high-risk actions like deletions traceable and verifiable. For storage bucket configuration management actions, such as deleting a bucket (DeleteBucket), modifying a bucket access control list (PutBucketACL), and modifying a bucket policy (PutBucketPolicy), these operations can be tracked through

CloudAudit logs, making permission configuration changes traceable and verifiable as well.



# Hotlink Protection Practice

Last updated: 2023-09-12 18:33:41

## Feature Overview

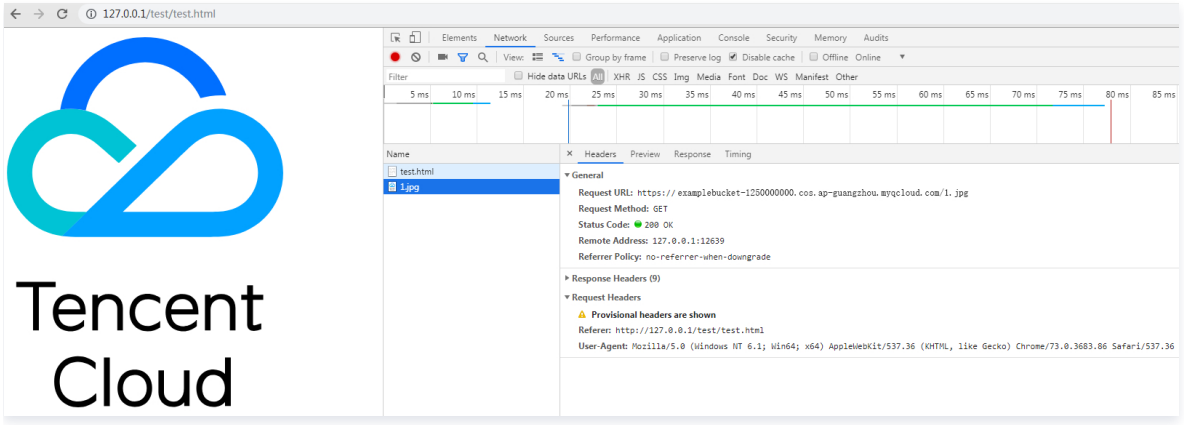
Tencent Cloud Object Storage (COS) supports anti-leech configuration, allowing users to set up anti-leech features for their buckets. This feature enables the creation of blacklists and whitelists for access sources, preventing unauthorized use of resources. This document provides a detailed guide on how to configure anti-leech settings for your bucket to protect your resources from unauthorized access.

## How to Determine Hotlink Protection

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 following scenario, an image `1.jpg` is embedded in `https://127.0.0.1/test/test.html`. When accessing `https://127.0.0.1/test/test.html`, the request carries a Referer pointing to the source of the access:




## Hotlink Protection Case Study

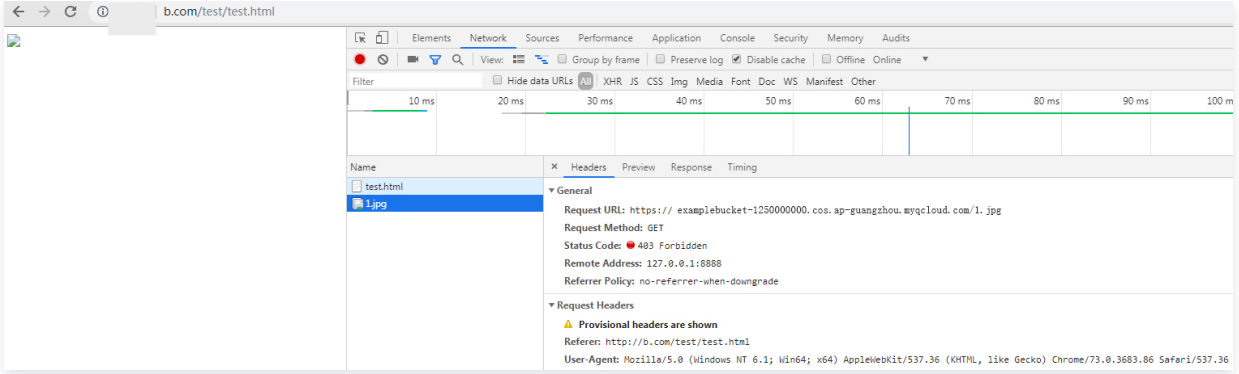
User A uploaded the image resource `1.jpg` to COS, and the accessible link to the image is `https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg`.

- User A embedded the image in their webpage `https://example.com/index.html` and the image is accessible.
- User B sees the image on User A's webpage and decides to embed it on their own webpage at `https://b.com/test/test.html`. Now, User B's webpage can also display the image normally. In the above example, User A's image resource `1.jpg` has been hotlinked by User B. At this point, without User A's knowledge, the resource on COS is continuously being used by User B's webpage, causing User A to bear additional traffic costs and resulting in financial loss.

## Solution

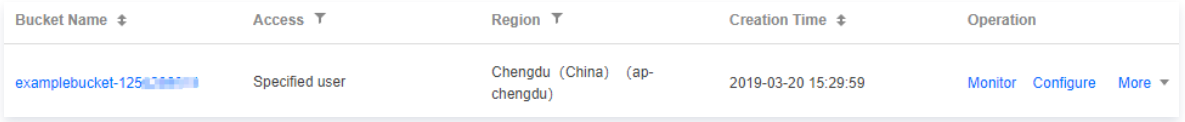
In the above Hotlink Protection Case Study, user A can prevent user B from hotlinking their image by setting hotlink protection in the following way:

1. Set a hotlink protection rule for the bucket "examplebucket-1250000000". There are two methods to prevent user B from hotlinking:
   - Method 1: Configure the blacklist by entering the domain name `*.b.com`, and save it.
   - Method 2: Configure a whitelist mode, 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.
   - Accessing `https://b.com/test/test.html` results in the image not being displayed, as shown below.

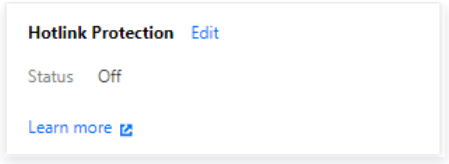



## Directions

1. Log in to the Object Storage Console and click **Bucket List** in the left navigation bar to enter the bucket list page.
2. Select the bucket for which you want to set up hotlink protection, and enter the bucket.



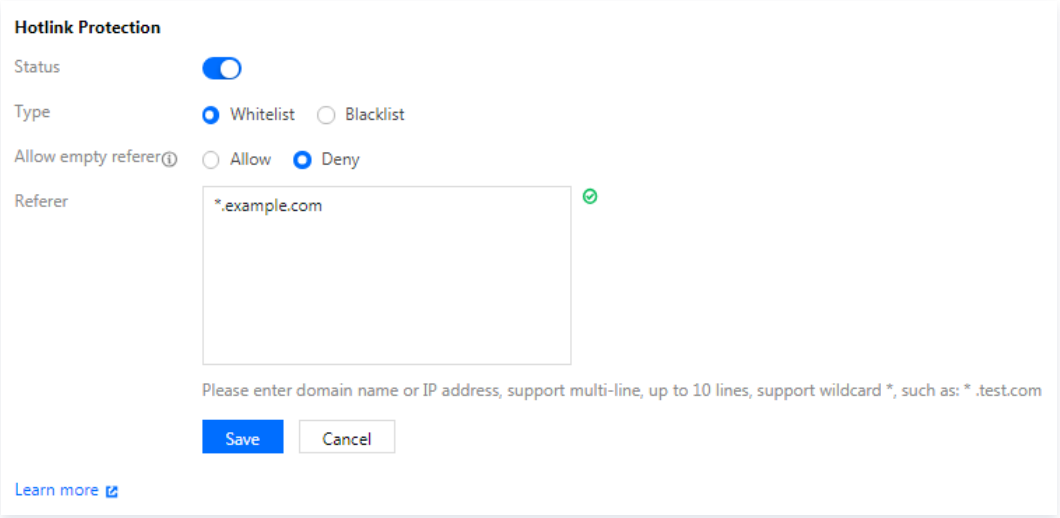

3. Click **Security Management > Hotlink Protection Settings** in the left sidebar to enter the bucket hotlink protection configuration page.
4. In the "Hotlink Protection Settings" section, click **Edit** to enter the editing mode.




5. Enable hotlink protection and configure the list type and domain name. Here, select enablement method 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 with IP and 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, its port domain name can 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** to finish.




## FAQs

For questions about hotlink protection, see the Data Security section in COS FAQs.

# Restricting Uploaded File Size and Type

Last updated: 2024-06-21 11:03:30

Cloud Object Storage (COS) allows you to set restrictions on the file types and sizes during the upload process. By limiting the file types, you can ensure that only the desired files are stored in your COS bucket. Restricting the file sizes enables you to manage your storage space more flexibly, preventing the waste of storage space and network bandwidth caused by excessively large or small files. The following are two examples illustrating how to precisely control the size of uploaded files.

## Prerequisites

The samples use the information below:

- APPID of the root account: 1250000000
- Bucket name: examplebucket-1250000000

In practice, please replace with your own bucket and use an account with the appropriate permissions and secret key to operate the bucket.

## Example 1: Specifying the object size range when uploading an object using the POST Object interface.

When uploading objects using `POST Object`, you can add `content-length-range` in the HTML form to control the object size in this request as follows:

```
[ "content-length-range", minNum, maxNum ]
```

Sample:

```
[ "content-length-range", 1, 10 ]
```

The JSON-formatted field above is added to policy > conditions in the POST request form. A complete policy with this field carried is as follows:

```
 {
    "expiration": "2021-12-31T12:00:00Z",
    "conditions": [
        { "bucket": "examplebucket-1250000000" },
        [ "starts-with", "$key", "exampleobject" ],
        { "q-ak": "AKIDQjz3ltompVjBni5LitkWHFlFpwkn****" },
        { "q-sign-algorithm": "sha1" },
        { "q-sign-time": "1567150692;1567157892" },
        [ "content-length-range", 1, 10 ]
    ]
}
```

For more information about how to construct a complete request, please see POST Object.

### Response

The following response will be returned as follows if the size of the object is within the specified size range:

```
HTTP/1.1 204
Content-Length: 0
Connection: close
Date: Wed, 23 Aug 2020 08:14:53 GMT
ETag: "ee8de918d05640145b18f70f4c3aa602"
Location: http://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject
Server: tencent-cos
x-cos-request-id: NWQ2NzgxMzZfMmViMDJhMDlfY2NjOF84NGQz****
```

The response will fail if the object size is not in the specified range.
If the object is too big, the response is as follows:

```
HTTP/1.1 400 Bad Request
Content-Type: application/xml
Content-Length: 498
Connection: keep-alive
Date: Wed, 23 Aug 2020 08:14:53 GMT
Server: tencent-cos
x-cos-request-id: NTk5ZDM5N2RfMjNiMjM1MGFfMmRiX2Y0****


<?xml version='1.0' encoding='utf-8' ?>
<Error>
        <Code>EntityTooLarge</Code>
        <Message>Condition key content-length-range doesn‘t match the value </Message>
        <Resource>examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject</Resource>
        <RequestId>NTk5ZDM5N2RfMjNiMjM1MGFfMmRiX2Y0****</RequestId>
</Error>
```

If the object is too small, the response is as follows:

```
HTTP/1.1 400 Bad Request
Content-Type: application/xml
Content-Length: 498
Connection: keep-alive
Date: Wed, 23 Aug 2020 08:14:53 GMT
Server: tencent-cos
x-cos-request-id: NTk5ZDM5N2RfMjNiMjM1MGFfMmRiX2Y0****


<?xml version='1.0' encoding='utf-8' ?>
<Error>
        <Code>EntityTooSmall</Code>
        <Message>Condition key content-length-range doesn‘t match the value </Message>
        <Resource>examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject</Resource>
        <RequestId>NTk5ZDM5N2RfMjNiMjM1MGFfMmRiX2Y0****</RequestId>
</Error>
```

## Example 2: Specifying object size range using a temporary key

The method used in sample 1 is easy, which requires only one parameter in the HTML form. However, it only supports `POST Object` but not `PUT Object`. Moreover, since the requester can still modify the parameter in requests, uploading objects beyond the specified size range is still possible, making it hard for central management.
Bucket administrators can use the following field to limit object size when applying for temporary keys. In COS, object size is represented by the field cos:content-length:

| Condition Field | Description | Sample |
| --- | --- | --- |
| numeric_greater_than | Greater than or equal to | {"numeric_greater_than": {"cos:content-length": 1}}, the object must be larger than 1 byte. |
| numeric_greater_than_equal | Value is greater than or equal to | {"numeric_greater_than_equal": {"cos:content-length": 1}}, The object size must be greater than or equal to 1 byte. |
| numeric_less_than | Less than | {"numeric_less_than": {"cos:content-length": 1}}, the object must be less than 1 byte. |

| | | |
|---|---|---|
| numeric_less_than_equal | Value is less than or equal to | {"numeric_less_than_equal": {"cos:content-length": 1}}, the object must be less than or equal to 1 byte. |

For a complete request, refer to the Obtaining Temporary Access Credentials API documentation. A complete policy is as follows:

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

            "condition": {
                "numeric_greater_than_equal": {"cos:content-length": 1}
                , "numeric_less_than": {"cos:content-length": 10}
            }
        }
    ]
}
```

With the temporary credential obtained using the following policy, you can call the `PUT Object` or `POST Object` API to upload objects to the `examplebucket-1250000000` bucket, with the object sizes limited to [1, 10), in bytes.

> **Note**
> This policy currently only applies to `cos:PutObject` and `cos:PostObject` actions; other actions (e.g., `cos:GetObject`) will fail.

This method allows bucket managers or the authentication center to centrally apply for temporary credentials and limit the size during the application, after which the credentials can be distributed to operators or business modules. In this way, object sizes can be controlled, avoiding uploading objects beyond the size range due to parameter modification.

### Response

If the size of the uploaded object is within the specified range, the upload request will succeed with 200 or 204 returned. Otherwise, 403 will be returned, as shown below:

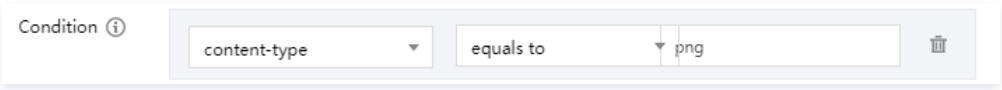
```
HTTP/1.1 403 Forbidden
Content-Type: application/xml
Content-Length: 298
Connection: keep-alive
Date: Wed, 23 Aug 2020 08:14:53 GMT
Server: tencent-cos
x-cos-request-id: NTk5ZDM5N2RfMjNiMjM1MGFfMmRiX2Y0****
```

## Example 3: Restricting file upload types using bucket policies

You can specify the file upload type in the bucket policy. For bucket policy syntax examples, please refer to Bucket Policy Examples. The steps to restrict file upload types using bucket policies are as follows:

1. Log in to the COS Console.
2. Click **Bucket List** on the left sidebar, select the bucket to which you want to add a bucket policy, and enter the bucket.
3. Click **Permission Management > Policy Settings**.
4. Based on your requirements, choose either **Graphical Settings** or **Policy Syntax** to add a bucket policy. For information on configuration items, please refer to Access Policy Language Overview.

5. When configuring bucket permission settings, set `content-type` to specify the file type, as shown in the image below:




> **Note**
> Supported file types: JPG, JPEG, PNG.

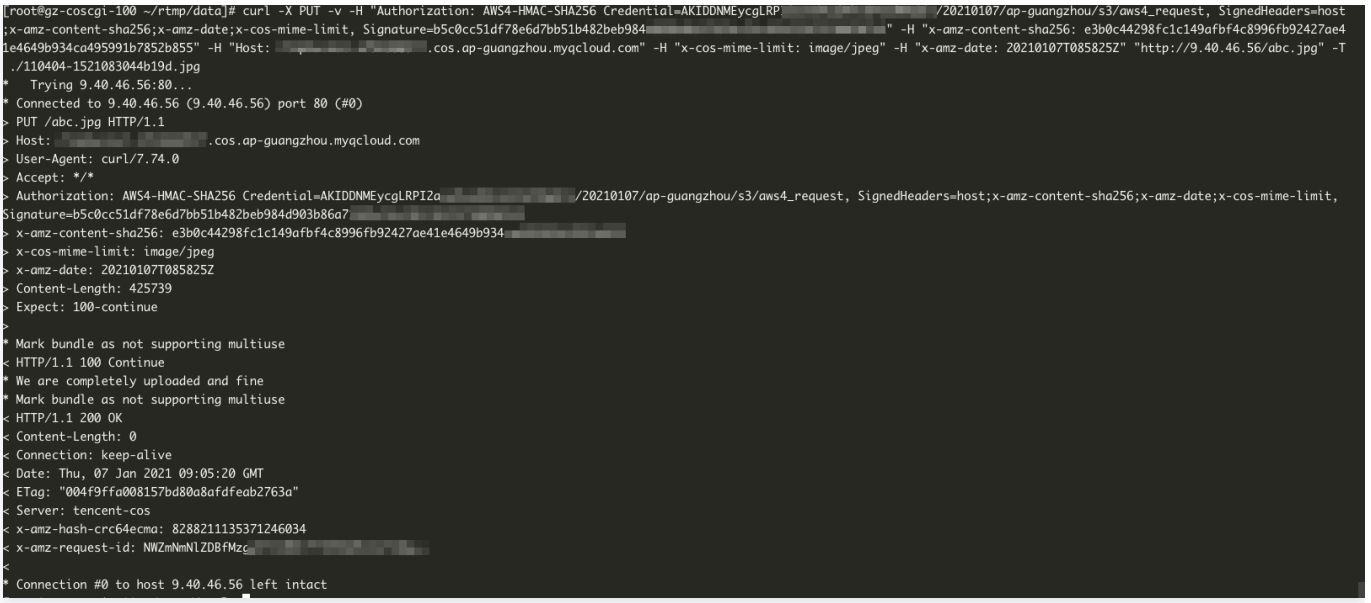
6. Click **Confirm** to save the policy. The policy is as follows:

```
{
  "Statement": [
{
  "Action": [
    "name/cos:PutObject"
  ],
  "Effect": "Allow",
  "Principal": {
    "qcs": [
      "qcs::cam::anyone:anyone"
    ]
  },
  "Resource": [
    "qcs::cos:ap-nanjing:uid/1250000000:examplebucket-1250000000/*"
  ],
  "Condition": {
            "ForAllValues:StringEquals": {
                "cos:content-type": ["image/png"]
            }
}
  ],
  "version": "2.0"
}
```

7. Verify if the policy is effective.
   7.1 Following the steps above, set the configuration to allow only `PNG` files to be uploaded, and upload a `PNG` format file:
   7.2 Verify if uploading a `PNG` format file is successful:

```
[root@gz-coscgi-100 ~/rtmp/data]# time curl -X PUT -v -H "Authorization: AWS4-HMAC-SHA256 Credential=AKIDDNMEycgLR/20210107/ap-guangzhou/s3/aws4_request, SignedHeaders
=host;x-amz-content-sha256;x-amz-date;x-cos-mime-limit, Signature=890684b7446cf1f7149eb98a14e46648a2" -H "x-amz-content-sha256: e3b0c44298fc1c149afbf4c8996fb924
27ae41e4649b934ca495991b7852b855" -H "Host: .cos.ap-guangzhou.myqcloud.com" -H "x-cos-mime-limit: image/png" -H "x-amz-date: 20210107T113008Z" "http://9.40.46.56/abc.png
" -T ./logo.png
*   Trying 9.40.46.56:80...
* Connected to 9.40.46.56 (9.40.46.56) port 80 (#0)
> PUT /abc.png HTTP/1.1
> Host: .cos.ap-guangzhou.myqcloud.com
> User-Agent: curl/7.74.0
> Accept: */*
> Authorization: AWS4-HMAC-SHA256 Credential=AKIDDNMEycg/20210107/ap-guangzhou/s3/aws4_request, SignedHeaders=host;x-amz-content-sha256;x-amz-date;x-cos-mime-limit,
Signature=890684b7446cf1f7149eb98a14e46648a2
> x-amz-content-sha256: e3b0c44298fc1c149afbf4c8996fb
> x-cos-mime-limit: image/png
> x-amz-date: 20210107T113008Z
> Content-Length: 6849
> Expect: 100-continue
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Content-Length: 0
< Connection: keep-alive
< Date: Thu, 07 Jan 2021 11:30:55 GMT
< ETag: "82b9c7a5a3f405032b1db71a25f67021"
< Server: tencent-cos
< x-amz-hash-crc64ecma: 6160088110901038901
< x-amz-request-id: NWZmNmYwZWRfMzg
<
```

### 7.3 Verify if uploading a `JPEG` format file is successful:




From this, it can be seen that the policy successfully intercepted non- `PNG` format files from being uploaded to COS.

# Data Verification
# MD5 Verification

Last updated: 2023-09-12 18:34:24

## Feature 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.
In COS, each object has an associated ETag, which is an identifier for the object's content when it is created. However, the ETag may not necessarily be equal to the MD5 checksum of the object's content. Therefore, you cannot use the ETag to verify whether the downloaded object is consistent with the original object. Instead, you can use custom object metadata (x-cos-meta-*) to perform consistency checks between the downloaded and original objects.

## Data Verification Methods

- Verify Uploaded Objects
  If you need to verify whether the object uploaded to COS is consistent with the local object, you can set the Content-MD5 field of the HTTP request to the Base64-encoded MD5 checksum of the object's content during the upload. In this case, the COS server will verify the uploaded object, and the object can only be successfully uploaded if the MD5 checksum received by the COS server is consistent with the Content-MD5 set by the user.
- Verify Downloaded Objects
  If you need to verify whether the downloaded object is consistent with the original object, you can calculate the object's checksum using a verification algorithm during the object upload, and set the object's checksum through custom metadata. After downloading the object, recalculate the object's checksum and compare it with the custom metadata for validation. In this approach, you can choose the verification algorithm independently, but for the same object, the verification algorithm used during upload and download should remain consistent.

## 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: ti4QvKtVqIJAvZxDbP/c+Q==
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO**&q-sign-time=1561109068;1561116268&q-key-time=1561109068;1561116268&q-header-list=content-length;content-md5;content-type;date;host&q-url-param-list=&q-signature=998bfc8836fc205d09e455c14e3d7e623bd2**
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-sign-time=1561109068;1561116268&q-key-time=1561109068;1561116268&q-header-list=content-length;content-md5;content-type;date;host&q-url-param-list=&q-signature=998bfc8836fc205d09e455c14e3d7e623bd2**
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
```

**Note**
For files uploaded in chunks, COS will only verify the MD5 value of each chunk and will not calculate the MD5 value of the complete file after merging.

### 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: NWQxZGUzZWNfNjI4NWQ2NF9lMWYyXzk1NjFj****
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. Initial 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)

# Set user attributes, including SecretId, SecretKey, and Region
```

```
# APPID has been removed from the configuration. Please include APPID in the Bucket parameter. The Bucket is composed of BucketName-APPID.
secret_id = os.environ['COS_SECRET_ID']     #  User SecretId. 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, visit https://cloud.tencent.com/document/product/598/37140.
secret_key = os.environ['COS_SECRET_KEY']   # User SecretKey. 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, visit https://cloud.tencent.com/document/product/598/37140.
region = 'ap-beijing'      # Replace with your own region (which is Beijing in this sample)
token = None               # Temporary key token. For more information on how to generate and use a temporary key, visit https://cloud.tencent.com/document/product/436/14048.
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token)  # Get the configured object
client = CosS3Client(config)
```

### 2. Verify Simple Uploaded Objects

#### (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

In the code, EnableMD5=True indicates that MD5 verification is enabled for object uploads. The Python SDK will calculate the Content-MD5, which will increase the upload time. The object can only be successfully uploaded if the MD5 checksum received by the COS server is consistent with the Content-MD5.

x-cos-meta-md5 is a user-defined parameter (custom parameter names are formatted as 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 Bucket name, where 'examplebucket' is a sample bucket and '1250000000' is a sample APPID.
    Body='hello cos',                // Uploaded object content
    Key='example-object-1',          # Replace with the Object Key value you uploaded
    EnableMD5=True,                  #Enable MD5 verification for uploads
    Metadata={                       # Set custom parameters, storing the MD5 checksum of the object as a parameter value in the COS server
        '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 an Object
response = client.get_object(
    Bucket='examplebucket-1250000000',  # Replace with your Bucket name, where 'examplebucket' is a sample bucket and '1250000000' is a sample APPID.
    Key='example-object-1'                  # Key value of the object to be downloaded
)
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 "x-cos-meta-md5"
```

### (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 against that of the uploaded
object
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```

## 3. Verify Multipart Uploaded Objects

### (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 Bucket name, where 'examplebucket' is a sample
bucket and '1250000000' is a sample APPID.
    Key='exampleobject-2',              # Replace with the Object Key value you uploaded
    StorageClass='STANDARD',            # The storage class of the object
    Metadata={
        'x-cos-meta-md5' : md5_str      # Set custom parameter, set to MD5 checksum value
    }
)
#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 the last part which may be
smaller
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 Part
    response = client.upload_part(
        Bucket='examplebucket-1250000000',  # Replace with your Bucket name, where 'examplebucket' is a
sample bucket and '1250000000' is a sample APPID.
        Key='exampleobject-2',              # Object's Key value
        Body=body,
        PartNumber=part_number,
        UploadId=upload_id,
        EnableMD5=True       #Enable chunk verification, and the COS server will perform MD5 verification on
each chunk.
    )
    etag = response['ETag']     #ETag represents the MD5 value of each chunk
    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 Bucket name, where 'examplebucket' is a sample
bucket and '1250000000' is a sample APPID.
    Key='exampleobject-2',              # Object's Key value
    UploadId=upload_id,
    MultipartUpload={       #Requires each part's ETag and PartNumber to correspond one-to-one
        'Part' : part_list
    },
)

ETag represents the unique tag value of the merged object, which is not the MD5 checksum of the object
content and can only be used to verify the object's uniqueness
print "ETag: " + response['ETag']
print "Location: " + response['Location']   #URL
print "Key: " + response['Key']
```

### (5) Download Object

Download the object and get the custom parameter.

```
# Download an Object
response = client.get_object(
    Bucket='examplebucket-1250000000',  # Replace with your Bucket name, where 'examplebucket' is a sample
bucket and '1250000000' is a sample APPID.
    Key='exampleobject-2'               # Object's Key value
)
print 'ETag: ' + response['ETag']                       # The ETag of the object is not the MD5 checksum of
the object content
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5']    # Get the custom parameter "x-cos-meta-md5"
```

### (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 against that of the uploaded
object
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```

# CRC64 check

Last updated: 2023-09-12 18:34:37

## Feature 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 value 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.

## Notes

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 detailed usage of the Python SDK, please refer to the Object Operations documentation for Python SDK.

### 1. Initial 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)

# Set user attributes, including SecretId, SecretKey, and Region
# APPID has been removed from the configuration. Please include APPID in the Bucket parameter. The Bucket is
composed of BucketName-APPID.
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 sample)
token = None               # If a temporary key is used, Token needs to be passed in, which is left empty by
default
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=token)  # Get the
configured object
client = CosS3Client(config)
```

### 2. Calculate the object's check value

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, rev=True)
local_crc64 =str(c64(object_body))
```

### 3. Initialize Multipart Upload

```
Initialize the multipart upload
response = client.create_multipart_upload(
    Bucket='examplebucket-1250000000',  # Replace with your Bucket name, examplebucket is a sample storage
bucket, and 1250000000 is a sample APPID
    Key='exampleobject',               # Replace with the object Key value you uploaded
    StorageClass='STANDARD',           # Object storage type
)
#Get the UploadId of the multipart upload
```

```
upload_id = response['UploadId']
```

### 4. Multipart Upload Object

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 that 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 the last part which may be
smaller
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_crc64 = str(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 verification
        print 'crc64 check FAIL'
        exit(-1)
    etag = response['ETag']
    part_list.append({'ETag' : etag, 'PartNumber' : part_number})
```

### 5. Complete 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 Bucket name, examplebucket is a sample storage
bucket, and 1250000000 is a sample APPID
    Key='exampleobject',             # Object's Key value
    UploadId=upload_id,
    MultipartUpload={                   #Requires each part's ETag and PartNumber to correspond one-to-one
        '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 COSClient creation, refer to: [Quick Start](https://cloud.tencent.com/document/product/436/10199);
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: 2023-09-12 18:34:51

## 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 from Github official platform.

### Installing hadoop-cos

To use COS as Deep Storage in Druid, you need to utilize the Druid-hdfs-extension:
After downloading HADOOP-COS, copy the required version `hadoop-cos-2.x.x.jar` from the dep directory to the Druid installation path `extensions/druid-hdfs-storage` and `hadoop-dependencies/hadoop-client/2.x.x`. Since Druid accesses COS using the HDFS plugin, the version must be consistent with the HDFS plugin version in Druid.

## How to Use

### Modifying configuration

1. Modify the `conf/druid/_common/common.runtime.properties` file in the Druid installation path, add the HDFS extension to `druid.extensions.loadList`, and specify HDFS as the deep storage for Druid. Set the path to the COSN path:

```
properties
druid.extensions.loadList=["druid-hdfs-storage"]
druid.storage.type=hdfs
druid.storage.storageDirectory=cosn://bucket-appid/<druid-path>
```

2. Create a new HDFS configuration file, hdfs-site.xml, in the `conf/druid/_common/` directory, and fill in the COS key information and other details:

```
<?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>xxxxxxxxxxxxxxxxxxxxxxx</value>
 </property>
 <property>
     <name>fs.cosn.userinfo.secretKey</name>
     <value>xxxxxxxxxxxxxxx</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 above configuration support items are fully consistent with the official HADOOP-COS documentation. For more information, please refer to the Hadoop Tools document.

### Getting Started

Sequentially start the Druid processes, and the Druid data will be loaded into COS.

# Using Terraform to Manage COS

Last updated: 2023-09-12 18:35:05

## Feature Overview

Terraform is an open-source automated resource orchestration tool that uses code to manage and maintain IT resources. Since 2017, Tencent Cloud has supported resource orchestration using Terraform. To date, over 10 core products seamlessly support Terraform, covering categories such as computing, storage, networking, and databases. This article provides a detailed guide on how to use Terraform to manage Cloud Object Storage (COS).

## Install Terraform

Terraform is easy to install. For more information on how to install it, see the "Installing Terraform" section in Tencent Cloud Terraform User Guide (I).

## Using Terraform to Manage COS

### Step 1: Initialization

> **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. Create a working directory, such as the terraform-test directory.
2. Create or prepare a Tencent Cloud account for Terraform, and obtain the SecretID and SecretKey.
3. Create a provider configuration file and include your SecretID, SecretKey, and other account information in the provider.tf file. Save it in the working directory (terraform-test directory). A sample provider.tf file is as follows:

```
provider "tencentcloud" {
 secret_id  = "AKIDdFUUEjgud7WpujygoiNsENJ1UigO****"
 secret_key = "yjFWbN4oJbeQwDG4e0AKN9f191f4****"
 region     = "ap-beijing"
}
```

> **Note**
> For other configuration methods, please refer to the "Configuring Tencent Cloud Provider File" section in Tencent Cloud Terraform User Guide (I).

4. Execute the init command to initialize the working directory (e.g., terraform-test), as shown in the following example:

```
[root@tigger terraform-test]#terraform init
```

If the following message is displayed, the initialization is successful.

```
Terraform has been successfully initialized!
```

> **Note**
> Each Terraform project requires a working directory where all operations are performed. Some Terraform commands allow specifying the working directory in their parameters. For more information, see Tencent Cloud Terraform User Guide (III). If not specified, the current directory is used as the default working directory.

### Step 2: Create a Bucket

1. Write a resource configuration file to define resources. For example, to create a private storage bucket named "examplebucket-1250000000", the configuration would be as follows:

```
resource "tencentcloud_cos_bucket" "mycos" {
  bucket = "examplebucket-1250000000"     #Bucket name, bucket name format: BucketName-APPID
  acl    = "private"  #ACL permission set to private
}
```

> **Note**
> During actual operation, be sure to replace the bucket name suffix with the user's real APPID; otherwise, COS will refuse to create the bucket.

Configuration files starting with "resource" and ending with `*.tf` define resources.

- tencentcloud_cos_bucket: Describes the resource type as a bucket. For more information on other resource types, see the left column here.
- mycos: Represents the resource name, which is user-defined.
  To create a bucket that supports static websites, see the following example:

```
resource "tencentcloud_cos_bucket" "examplebucket2" {
  bucket = "examplebucket2-1250000000"

  website {
 index_document = "index.html"
 error_document = "error.html"
  }
}
```

If you want to set up CORS, run the following sample code:

```
resource "tencentcloud_cos_bucket" "examplebucket3" {
  bucket = "examplebucket3-1250000000"
  acl    = "public-read-write"

  cors_rules {
 allowed_origins = ["http://*.abc.com"]
 allowed_methods = ["PUT", "POST"]
 allowed_headers = ["*"]
 max_age_seconds = 300
 expose_headers  = ["Etag"]
  }
}
```

> **Note**
> Terraform can manage COS bucket properties such as website, ACL, cors_rules, and lifecycle_rules. For more information, see the Argument Reference section in tencentcloud_cos_bucket.

2. Execute the `terraform apply` command to deploy resources and create the storage bucket examplebucket-1250000000. You can log in to the COS console to view the newly created storage bucket examplebucket-1250000000.

> **Note**
> If the bucket "examplebucket-1250000000" already exists, Terraform will first delete the original bucket and then create an empty one.

Before running the `terraform apply` command, you can execute the `terraform plan` command first. The `terraform plan` command allows you to verify if the execution plan meets expectations without making any changes to the actual resources or state.

## Step 3: Create Object Resource

1. Create a resource configuration file to define resources. For example, to upload the file picture.jpg to the storage bucket examplebucket-1250000000, follow the template below:

```
resource "tencentcloud_cos_bucket_object" "myobject" {
  bucket             = "examplebucket-1250000000"  # Bucket name, format: BucketName-APPID
  key                = "picture.jpg"  # Object key
  source = "D:\folder\picture.jpg"  # Path of the file to be uploaded, including both the directory and
file name
}
```

2. Run the `terraform apply` command to deploy resources and upload picture.jpg.

> **Note**
> - Terraform can be used to manage the attributes of COS objects such as upload, ACL, content, ETag, and storage_class. For more information, see the "Argument Reference" section here.
> - Currently, Terraform does not support downloading objects, because Terraform is designed to be a resource orchestration tool that focuses on the orchestration and deployment of cloud resources.

## Step 4: Query Resources

1. Run the `terraform show` command to view all resource information (i.e., metadata for all buckets and objects). To query specific information for a particular bucket or object, you need to create a configuration file and then execute the `terraform apply` command.
2. Refined Query for Bucket Information.
   2.1 Create a data source configuration file to define specific query conditions. For example, to query buckets with the prefix "example", the configuration would look like this:

```
data "tencentcloud_cos_buckets" "cos_buckets" {
  bucket_prefix      = "example"  # Bucket prefix
  result_output_file = "mytestpath"  # Filename for saving query results
}
```

   Configuration files starting with "data" and ending with `*.tf` define the query conditions.
   - tencentcloud_cos_buckets: Describes the resource type as a bucket. For more information, see the left column here.
   - cos_buckets: Describes the resource name which is customizable.

   2.2 Run the `terraform apply` command to perform the query. The query results are saved in the file mytestpath.

> **Note**
> To query the properties of a storage bucket, please refer to the Argument Reference section of tencentcloud_cos_buckets for more information.

3. Refined object information query.
   3.1 Create a data source configuration file to define specific query conditions. For example, to query the metadata of the object "picture.jpg", the configuration would look like this:

```
data "tencentcloud_cos_bucket_object" "mycos" {
  bucket             = "examplebucket-1250000000"  # Bucket name, format: BucketName-APPID
  key                = "picture.jpg"  # Object key
  result_output_file = "mytestpath"  # Filename for saving query results
}
```

4. Run the `terraform apply` command to perform the query. The query results are saved in the file mytestpath.

> **Note**
> To query object attribute information, please refer to the Argument Reference section of tencentcloud_cos_bucket_object.

### Step 5: Delete Resources

Execute the `terraform destroy` command to delete all resources in the working directory.
To delete a specific bucket or object, remove the configuration information defining the bucket or object resource from the configuration file, and then execute the `terraform apply` command. It is recommended to use the `terraform show` command to verify the deletion results.

# Importing/Exporting COS Using DataX

Last updated: 2023-09-12 18:35:19

# Environment 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 from 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/hdfswriter/libs/*:.") %
(DATAX_HOME, DATAX_HOME, DATAX_HOME)
```

### 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"
                }
            }
        }]
    }
}
```

The configuration is as described below:

- Configure `hadoopConfig` as required for cosn.
- Set defaultFS to the cosn path, such as `cosn://examplebucket-1250000000/`.
- Modify fs.cosn.userinfo.region to the region where the bucket is located, for example, `ap-beijing`. For more information, please refer to Regions and Access Domains.
- For `COS_SECRETID` and `COS_SECRETKEY`, use your own COS key information.

Other configurations can be the same as HDFS configuration items.

## Migrating data

Save the configuration file as `hdfs_job.json`, place it in the `job` directory, and run the following command:

```
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                    | minDeltaCpu
                -1.00%                         | -1.00%                         | -1.00%
```

```
         [total gc info] =>
                NAME                 | totalGCCount       | maxDeltaGCCount    | minDeltaGCCount    |
totalGCTime         | maxDeltaGCTime     | minDeltaGCTime
                PS MarkSweep         | 1                  | 1                  | 1                  | 0.024s
| 0.024s              | 0.024s
                PS Scavenge          | 1                  | 1                  | 1                  | 0.014s
| 0.014s              | 0.014s

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 records, 33 bytes | Speed
3B/s, 0 records/s | Error 0 records, 0 bytes |  All Task WaitWriterTime 0.000s |  All Task WaitReaderTime
0.033s | Percentage 100.00%
2020-03-09 16:49:59.544 [job-0] INFO  JobContainer -
Job start time                    : 2020-03-09 16:49:48
Task end time                     : 2020-03-09 16:49:59
Job duration                      :                11s
Average task traffic                     :               3 B/s
Recorded write speed                     :              0rec/s
Read records                      :                  2
Read/Write failure count                    :                  0
```

# Configuring COSN for CDH

Last updated: 2023-09-12 18:35:31

## Feature 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 COSN, a flexible, cost-effective big-data solution, in a CDH environment to separate big data computing from storage.

> **Note**
> COSN is an abbreviation for Hadoop-COS file system.

The table below shows which big data modules are supported by COSN:

| Module | Required | Service Module to Restart |
| --- | --- | --- |
| Yarn | This feature is supported. | NodeManager |
| Hive | This feature is supported. | HiveServer and HiveMetastore |
| Spark | This feature is supported. | NodeManager |
| Sqoop | This feature is supported. | NodeManager |
| Presto | This feature is supported. | HiveServer, HiveMetastore, and Presto |
| Flink | This feature is supported. | Not required |
| Impala | This feature is supported. | Not required |
| EMR | This feature is supported. | Not required |
| Self-built components | Will be supported later | - |
| HBase | Not recommended | - |

## Versions

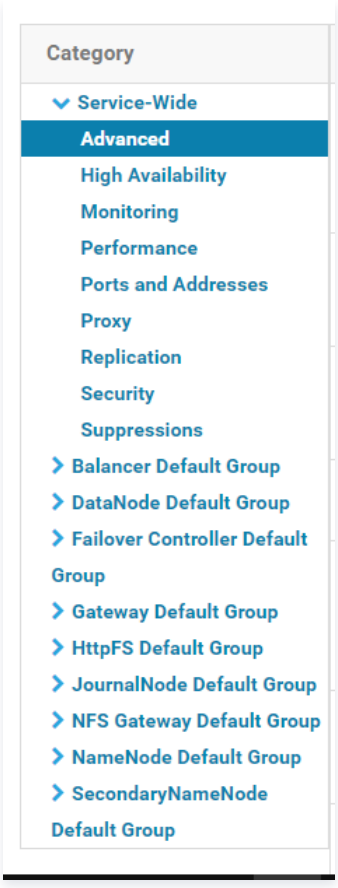
This example uses software versions as follows:

- 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 system homepage, select **Configuration** > **Service Scope** > **Advanced** to access the advanced configuration code snippet page, 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 are the required COSN configuration items (to be added to core-site.xml). For other COSN configurations, please refer to the Hadoop Tools documentation.

| COSN Parameter | 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 in FileSystem, which is always org.apache.hadoop.fs.CosFileSystem. |
| fs.AbstractFileSystem.cosn.impl | org.apache.hadoop.fs.CosN | The implementation class of COSN for AbstractFileSystem, which is fixed at `org.apache.hadoop.fs.CosN` |

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 package must be placed in the same location on each machine within 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

**Operation Steps**
(1) Follow the instructions in the Data Migration section to configure the HDFS settings and place the COSN SDK jar file in the appropriate HDFS directory.
(2) On the CDH system homepage, locate YARN and restart the NodeManager service (TeraGen command does not require a restart, but TeraSort requires a NodeManager restart due to internal business logic; it is recommended to restart the NodeManager service for both).
**Example**
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**
> Please replace the `cosn:// schema` with the storage bucket path for your big data business.

### 2. Hive

#### 2.1 MR engine

**Operation Steps**
(1) Follow the Data Migration section to configure the relevant HDFS settings and place the COSN SDK jar file in the appropriate HDFS directory.
(2) On the CDH main page, locate the HIVE service and restart the Hiveserver2 and HiverMetastore roles.
**Example**
To query your actual business data, use the Hive command line to create a location as a partitioned table on CHDFS:

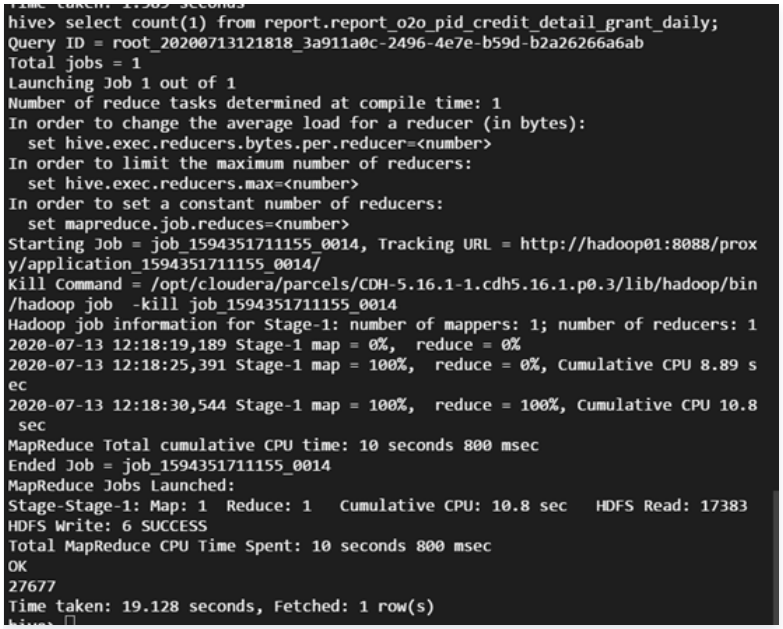
```
CREATE TABLE <1>report.report_o2o_pid_credit_detail_grant_daily</1>(
  <1>cal_dt</1> string,
```

```
  <1>change_time</1> string,
  <1>merchant_id</1> bigint,
  <1>store_id</1> bigint,
  <1>store_name</1> string,
  <1>wid</1> string,
  <1>member_id</1> bigint,
  <1>meber_card</1> string,
  <1>nickname</1> string,
  <1>name</1> string,
  <1>gender</1> string,
  <1>birthday</1> string,
  <1>city</1> string,
  <1>mobile</1> string,
  <1>credit_grant</1> bigint,
  <1>change_reason</1> string,
  <1>available_point</1> bigint,
  <1>date_time</1> string,
  <1>channel_type</1> bigint,
  <1>point_flow_id</1> bigint)
PARTITIONED BY (
  <1>topicdate</1> 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_credit_detail_grant_daily'
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 observed results are as follows:



### 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:
**Operation Steps**
(1) Locate the Tez package installed in the CDH cluster and extract it, for example, /usr/local/service/tez/tez-0.8.5.tar.gz.
(2) Place the COSN .jar file in the extracted directory, and then re-compress it into a new archive.
(3) Upload the new archive to the path specified by tez.lib.uris (replace the existing file if it already exists).
(4) In the CDH main page, locate HIVE and restart hiveserver and hivemetastore.

## 3. Spark

**Operation Steps**
(1) Follow the instructions in the Data Migration section to configure the HDFS settings and place the COSN SDK jar file in the appropriate HDFS directory.
(2) Restart the NodeManager service.
**Example**
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 execution result is as follows:

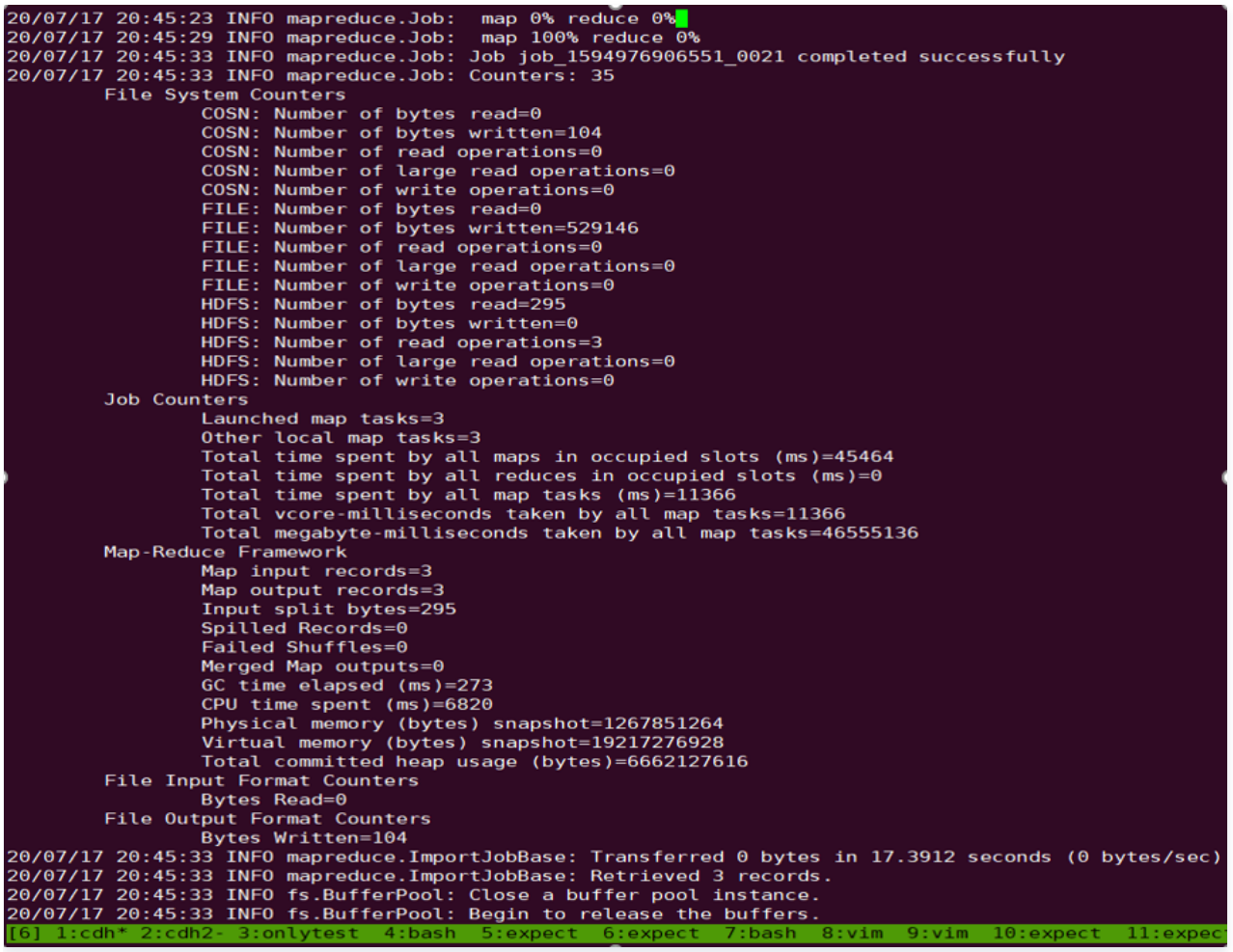
```
20/07/17 20:39:03 INFO cluster.YarnScheduler: Removed TaskSet 1.0, whose tasks have all completed, from pool
20/07/17 20:39:03 INFO scheduler.DAGScheduler: ResultStage 1 (collect at JavaWordCount.java:68) finished in 0.077 s
20/07/17 20:39:03 INFO scheduler.DAGScheduler: Job 0 finished: collect at 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://10.0.0.42:4040
```

## 4. Sqoop

**Operation Steps**
(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 the NodeManager service.
**Example**
For example, to export MYSQL tables to COSN, please 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 execution result is as follows:




## 5. Presto

**Operation Steps**

(1) Follow the instructions in the Data Migration section to configure the HDFS settings and place the COSN SDK JAR package in the appropriate HDFS directory.

(2) The COSN SDK JAR package also needs to be placed in the Presto directory (e.g., /usr/local/services/cos_presto/plugin/hive-hadoop2).

(3) Since Presto does not load gson-2...jar from the Hadoop common directory, you need to place gson-2...jar in the Presto directory as well (e.g., /usr/local/services/cos_presto/plugin/hive-hadoop2, only CHDFS depends on gson).

(4) Restart the HiveServer, HiveMetaStore, and Presto services.

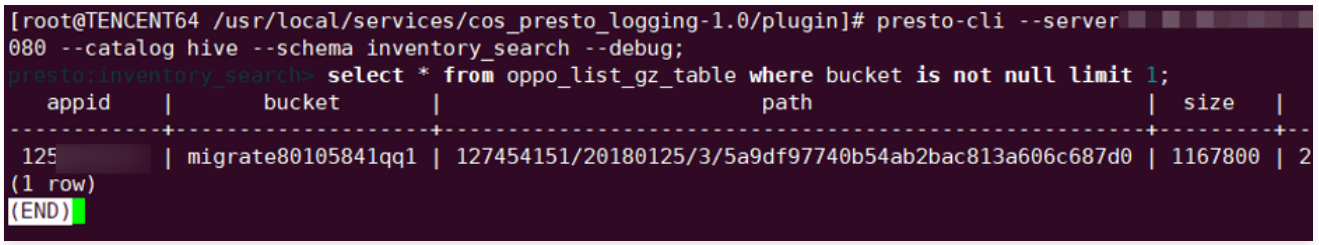
**Example**

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 query results are as follows:



# COS Ranger Permission System Solution

Last updated: 2023-09-12 18:35:46

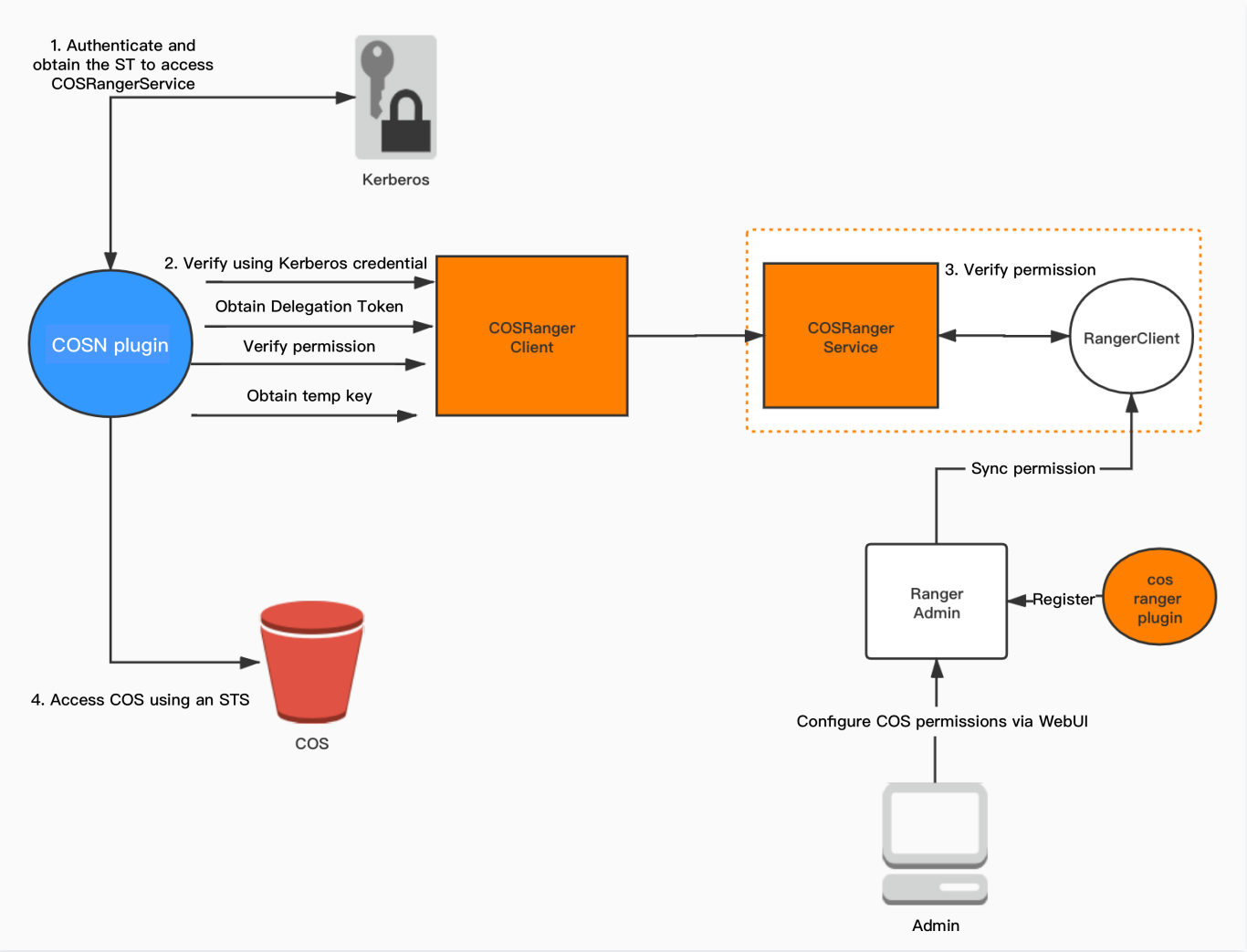
## Example

Hadoop Ranger is a permission solution for big data scenarios. A user adopting the compute/storage separation mode can host data in COS. However, COS uses the CAM permission system, meaning that the user roles and permission policies could be different from those of Hadoop Ranger. Therefore, we introduce a solution to integrate COS with Ranger herein.

## Advantages

- Fine-grained and Hadoop-compatible permission control, allowing users to centrally manage permissions for big data components and data hosted in 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, the authentication is offered by Kerberos and authorized by Ranger. On the basis of this, the following components are introduced 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
- Kerberos (required if authentication is needed), ZooKeeper, and Ranger

> **Note**
> As these services are mature open-source components, customers can install them independently.

## Component Deployment

Deploy components in the following sequence: COS Ranger Plugin, COS Ranger Service, COS Ranger Client, COSN.

### Deploying COS Ranger Plugin

COS-Ranger-Plugin extends the service types of the Ranger Admin console. Users can configure the COS-related permissions in the Ranger console.

#### Source code download

You can go to GitHub > ranger-plugin to obtain the source code.

#### Version

v1.0 or later.

#### Deployment steps

1. Create a new COS directory under the Ranger service definition directory (ensure that the directory permissions include at least read and execute permissions).
   a. For Tencent Cloud EMR environment, the path is ranger/ews/webapp/WEB-INF/classes/ranger-plugins.
   b. For self-built Hadoop environments, locate the directory by searching for components like HDFS that have already been integrated with Ranger services in the Ranger directory.

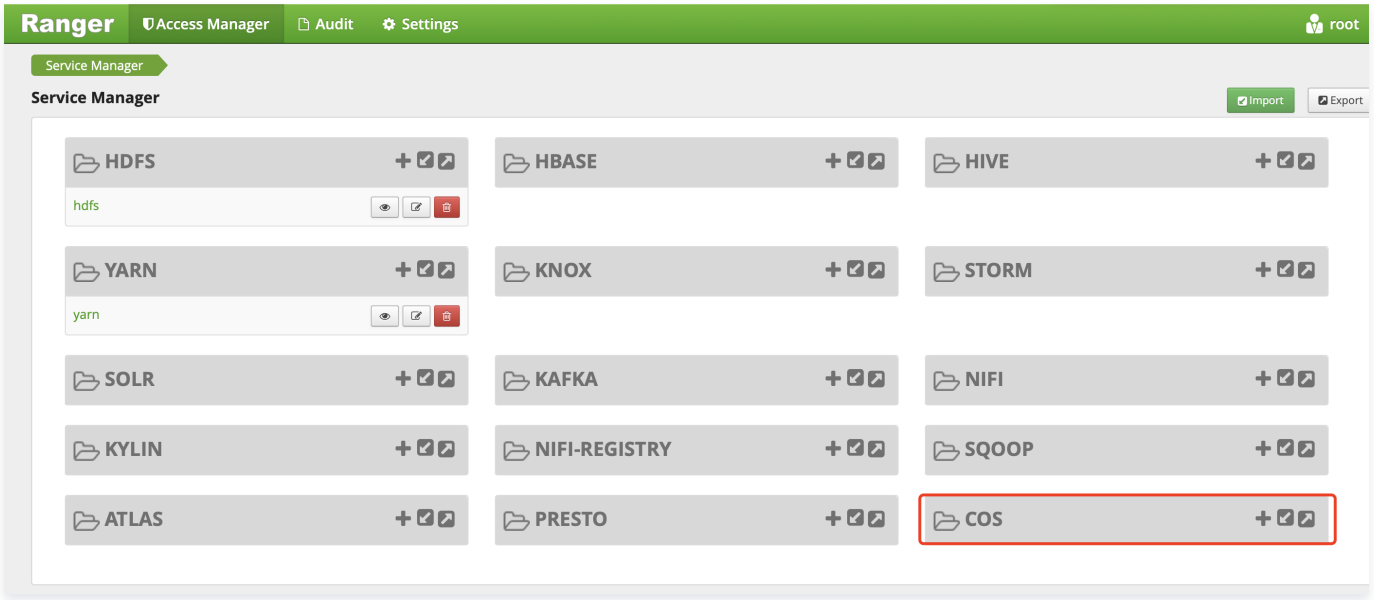
```
[hadoop@10 ranger-plugins]$ ls -l
total 68
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 atlas
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 chdfs
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 cos
drwxr-xr-x 2 hadoop hadoop 4096 Feb 25  2020 hbase
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 hdfs
drwxr-xr-x 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 the cos-chdfs-ranger-plugin-xxx.jar file in the COS directory (ensure the JAR file has at least read permission). Additionally, include the cos-ranger.json file, which can be obtained from Github.
3. Restart Ranger.

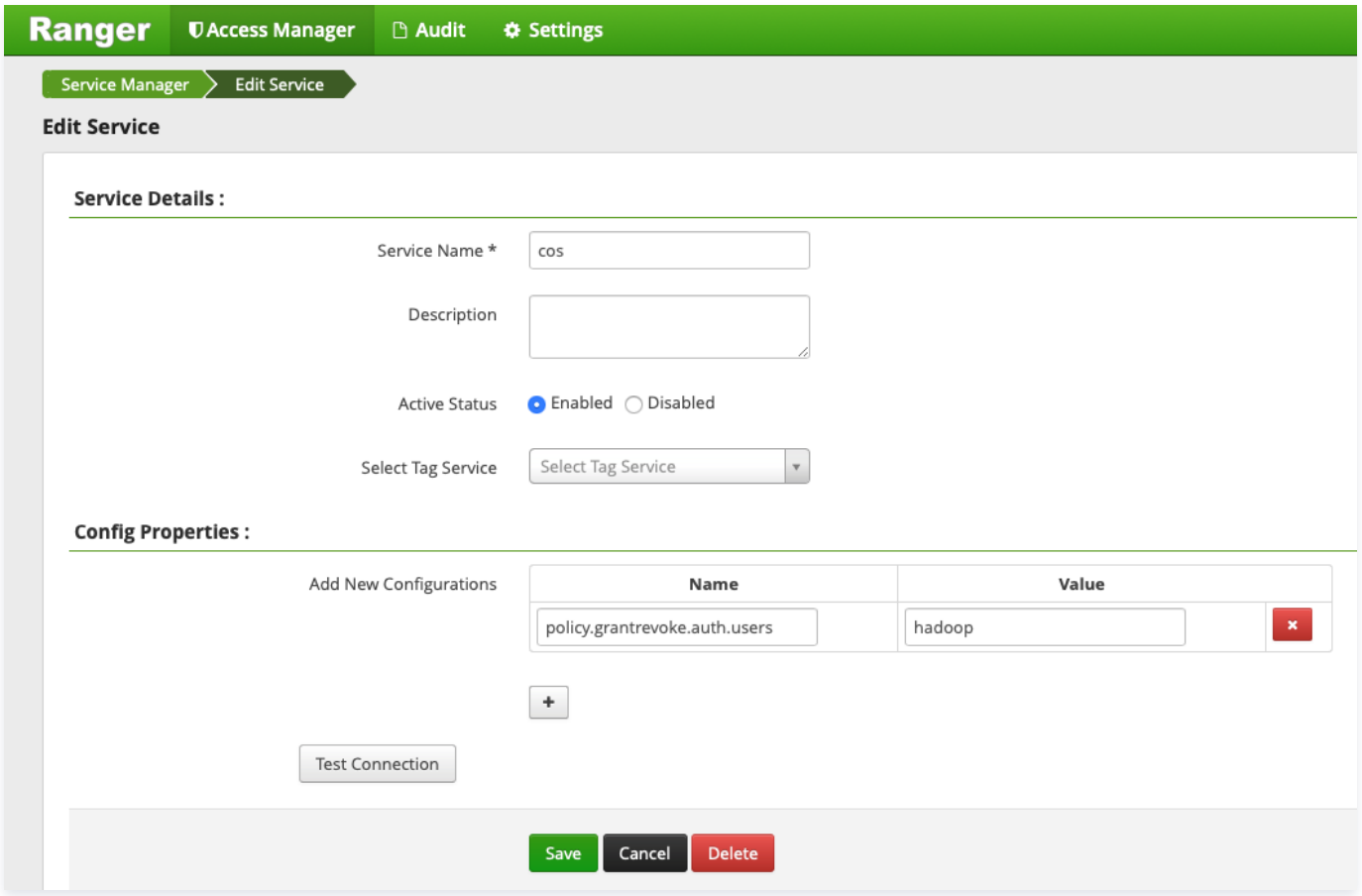
4. Register COS service in Ranger. A command example is as follows:

```
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
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 master 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. Upon successful service creation, the COS service will be visible in the Ranger console, as shown below:



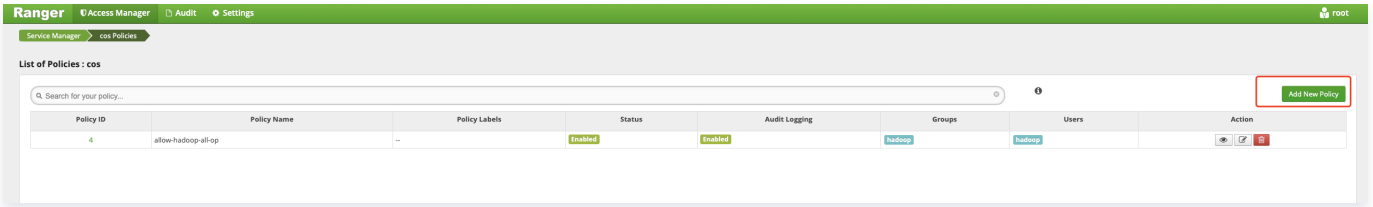

6. In the COS service, click + to define a new service instance with a custom name, such as `cos` or `cos_test`. Configure the service as shown below.




The policy.grantrevoke.auth.users should be set to the username for starting the COS Ranger Service (i.e., the user allowed to fetch permission policies). It is generally recommended to set it as "hadoop" so that the COS Ranger Service can be started using this username.

7. Click the newly created COS service instance.

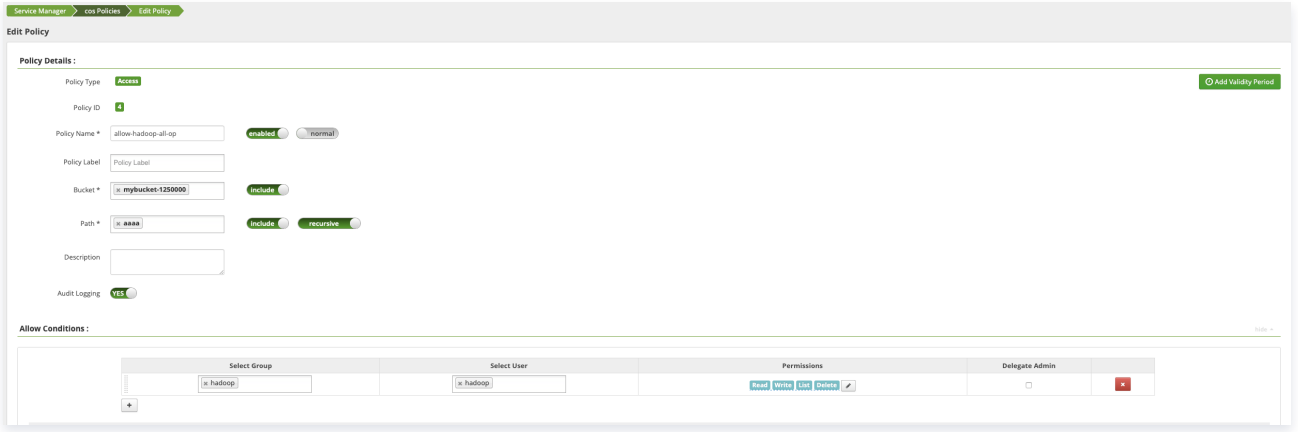

Add a policy as shown below:




8. On the page that is displayed, configure the following parameters:

- **bucket**: The bucket name, such as examplebucket-1250000000, can be viewed by logging into 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: Enumeration permission, corresponding to List Object in object storage.




## Deploying COS Ranger Service

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.

### Source code download

You can obtain it from the cos-ranger-server directory on Github.

### Version

v5.1.2 or later.

### Deployment steps

1. Copy the code of COSRangerService to several nodes of the cluster. In the production environment, the code should be copied to at least two nodes (one master node, and one slave node). As sensitive information is involved, you are advised to use jump servers or nodes with tight permission control.
2. Modify the relevant configurations in the cos-ranger.xml file, with the mandatory items shown below. For configuration item descriptions, please refer to the comments in the file (the configuration file can be obtained from the cos-ranger-service/conf directory on 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 configurations in the ranger-cos-security.xml file. The mandatory configurations to be modified are as follows. For explanations of the configurations, please refer to the comments in the file (the configuration file can be obtained from the cos-ranger-service/conf directory on 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 corresponds to the directory of the Hadoop configuration files (for example, `core-site.xml` and `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 launch 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:

```
# Please replace the following 10.xx.xx.xxx with the IP address of the machine where Ranger Service is
deployed.
# Set port 9998 as the configuration value for qcloud.object.storage.status.port
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.

## Deploying COS Ranger Client

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.

### Source code download

You can obtain the cos-ranger-client and cosn-ranger-interface from the Github directories.

### Version

The cos-ranger-client requires v5.0 or later, while the cosn-ranger-interface requires v1.0.4 or later.

### Deployment directions

1. Copy the JAR packages of COS Ranger Client and COS 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 whose major version is the same as that of Hadoop and make sure that them 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 credentials for the COS Ranger Service side, referring to the COS Ranger
Service side configuration. Ensure consistency; if there is no authentication requirement, no
configuration is needed. -->
        <property>
                <name>qcloud.object.storage.kerberos.principal</name>
                <value>hadoop/_HOST@EMR-XXXX</value>
        </property>

      <!--*Optional Configuration*-->
      <!-- ZooKeeper path for the recorded Ranger server IP address. The default value is used here,
and the configuration must be consistent with the COS Ranger Service configuration. -->
        <property>
                <name>qcloud.object.storage.zk.leader.ip.path</name>
                <value>/ranger_qcloud_object_storage_leader_ip</value>
        </property>
      <!-- The IP address path of the COS Ranger Service follower recorded on ZooKeeper. The default
value is used here. It must be consistent with the COS Ranger Service configuration, as both primary and
secondary nodes provide services simultaneously. -->
        <property>
                <name>qcloud.object.storage.zk.follower.ip.path</name>
                <value>/ranger_qcloud_object_storage_follower_ip</value>
        </property>
</configuration>
```

Deploying COSN

### Version

v8.0.1 or later

### Deployment directions

For detailed directions on the deployment of COSN, please see Hadoop. Please 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 performed via Ranger. The following 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 related to COSN access, i.e., check whether the current operations comply with the permissions set by the root account.

```
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, related services, such as Yarn and Hive, should be restarted first.

## 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 temp key is cached on the COSN side. 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: 2023-09-12 18:36:00

## 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. For more information, see Create a Dedicated Cluster documentation.

### 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 writing to COS, the region where the Oceanus job is running must be the same as the COS region.

## Practical Steps

Go to the Oceanus Console and create an SQL job. Select a cluster in the same region as COS. For more information, refer to the Create SQL Job documentation.

### 1. Create Source

```
CREATE TABLE random_source (
  f_sequence INT,
  f_random INT,
  f_random_str VARCHAR
  ) WITH (
  'connector' = 'datagen',
  'rows-per-second'='10',                 -- Number of data records generated per second
  'fields.f_sequence.kind'='random',      -- Random number
  'fields.f_sequence.min'='1',            -- Minimum value of the random number
  'fields.f_sequence.max'='10',           -- Maximum value of the random number
  'fields.f_random.kind'='random',        -- Random number
  'fields.f_random.min'='1',              -- Minimum value of the random number
  'fields.f_random.max'='100',            -- Maximum value of the random number
  'fields.f_random_str.length'='10'       -- Length of the random string
);
```

> **Note**
> Here, the built-in connector `datagen` is used. Please choose the appropriate data source according to your actual business needs. For more information, refer to the Oceanus Upstream and Downstream Development Guide.

### 2. Create Sink

```
-- Please replace <bucket name> and <folder name> with your actual bucket name 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 for data writing
    'format' = 'json',                                           --- Data write format
    'sink.rolling-policy.file-size' = '128MB',                   --- Maximum file size
    'sink.rolling-policy.rollover-interval' = '30 min',          --- Maximum file write duration
    'sink.partition-commit.delay' = '1 s',                       --- Partition commit delay
    'sink.partition-commit.policy.kind' = 'success-file'         --- Partition commit method
);
```

**Note**
For more Sink WITH parameters, please refer to the Filesystem (HDFS/COS) documentation.

### 3. Business Logic

```
INSERT INTO cos_sink
SELECT * FROM random_source;
```

**Note**
This demonstration serves no actual business purpose.

### 4. Job Parameter Settings

In **Built-in Connector**, select `flink-connector-cos`, and configure the COS address 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 location>
fs.cosn.userinfo.appid: <COS User's App ID>
```

The job is configured as follows:

- Please replace `<COS Region>` with your actual COS region, for example: ap-guangzhou.
- Please replace `<COS user's appid>` with your actual APPID, which can be found in the Account Center.

**Note**
For specific job parameter settings, please refer to the Filesystem (HDFS/COS) documentation.

### 5. Start the job.

Click **Save > Syntax Check > Publish Draft** in sequence. After the SQL job starts, you can go to the corresponding COS directory to view the written data.

# Using COS in the Third-party Applications
# Using COS in APICloud

Last updated: 2023-09-20 11:40:33

## Feature Overview

This document demonstrates how to quickly integrate the COS module using APICloud and develop cross-platform applications.

## APICloud Related Resources

- APICloud Official Website
- Video Tutorials

## Preparations

Follow the APICloud tutorial to create an App, then search for the cosClient module in the module library and click Add.

## Basic Usage of cosClient

### 1. Obtain the Module Instance

```
var demo = api.require('cosClient');
```

### 2. Configuring the Secret Key

The COS SDK module offers two key configuration methods:

- **Temporary Key (Recommended)**

  Implement the following within the script tag:

```
function refreshCredentail() {
  // Obtain temporary credentials <1>secretID</1>, <3>secretKey</3>, <5>token</5>, <7>startDate</7>, and
<9>expirationDate</9> via API request;
  // Concatenate into the following format and return:
  return  "secretID=<1>*&secretKey=</1>*&token=<3>*&startDate=</3>*&expirationDate=***";
      }
```

- **Permanent Key**

  Set up a permanent key before registering the service:

```
var demo = api.require('cosClient');
demo.setupPermanentCredentail({"secretID":secretID,"secretKey":secretKey});
```

### 3. Registering a COS Service Instance

```
demo.registerTransferManger({"serviceKey":"test","useHttps":true})
```

## COS Client API Documentation

### 1. Register for Basic Services

#### Example Code

```
var demo = api.require('cosClient');
demo.registerServiceForKey({"serviceKey":"test","useHttps":true})
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | Unique identifier for the COS service instance; if not provided, the default service will be registered. | Not required |
| appId | String | Your APPID | Not required |
| region | String | Service Region Name: For available service region names, please refer to the regions provided on the official website at `https://www.qcloud.com/document/product/436/6224`. Enter the region abbreviation as provided on the website, such as ap-beijing. | Not required |
| isPrefixURL | Boolean Value | - | Not required |
| timeOut | Integer | Timeout Period | Not required |
| serviceName | String | Base service name, default value: myqcloud.com | Not required |
| suffix | String | Custom Domain: `http://bucketname.suffix` If this parameter is not specified, the bucket host is http://bucketname.**. **If this parameter is set to testsuffix, the bucket host is http://bucketname.testsuffix.**. This takes effect on the iOS platform. | Not required |
| useHttps | Boolean Value | Whether to use HTTPS | Not required |
| userAgent | String | Setting Custom User-Agent | Not required |
| host | String | Custom domain name, excluding "http://", effective on Android devices. | Not required |
| port | Integer | Custom ports are effective on the Android platform. | Not required |

- Response Parameters
  None

## 2. Register for Data Transmission Service

### Example Code

```
var demo = api.require('cosClient');
demo.registerTransferManger({"serviceKey":"test","useHttps":true})
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | Unique identifier for the COS service instance; if not provided, the default service will be registered. | Not required |
| appId | String | Your AppID.<br>AppID is the account you receive after successfully applying for a Tencent Cloud account. It is automatically assigned by the system and has a fixed and unique value. You can view it in Account Information. The AppID of a Tencent Cloud account has a unique corresponding relationship with the account ID. | Not required |
| region | String | Service Region Name: For available service region names, please refer to the regions provided on the official website at | Not required |

| | | | |
|---|---|---|---|
| | | `https://www.qcloud.com/document/product/436/6224`. Enter the region abbreviation as provided on the website, such as ap-beijing. | |
| isPrefixURL | Boolean Value | Default is YES. This controls the position of the bucket name in the URL when concatenating the link.<br>Example:<br>NO: `https://ap-chengdu.cos.myqcloud.com/0-12500000000`<br>YES: `https://0-12500000000.cos.ap-chengdu.myqcloud.com` | Not required |
| timeOut | Integer | Timeout Period | Not required |
| serviceName | String | Base service name, default value: myqcloud.com | Not required |
| suffix | String | Custom Domain: `http://bucketname.suffix` If this parameter is not specified, the bucket host is http://bucketname.**. **If this parameter is set to testsuffix, the bucket host is http://bucketname.testsuffix.**.** This is effective on the iOS platform. | Not required |
| useHttps | Boolean Value | Whether to use HTTPS | Not required |
| userAgent | String | Setting Custom User-Agent | Not required |
| host | String | Custom domain name, excluding "http://", effective on Android devices. | Not required |
| port | Integer | Custom ports are effective on the Android platform. | Not required |

- Response Parameters
  None

## 3. Retrieving the Bucket List

### Example Code

```
/// Module name: cosClient. Obtain the module instance through the module name.
var demo = api.require('cosClient');
demo.getBucketList(
    {"serviceKey":"test"}
    ,function(ret,err){
    /// When err is empty, it returns ""
        if(err != ""){
            alert(err.data);
        }else{
            alert(ret.data);
        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
|---|---|---|---|
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |

- Response parameters

```
{
 "result":"success",
 "data":{
  buckets = [ /// Bucket list
{
```

```
            createDate = "2021-06-02T07:33:33Z"; /// Creation Time
            location = "ap-chengdu"; /// Region
            name = "0-1253960454";  /// Bucket Name
            type = ""; Bucket Type
        }];
    owner =     {
        displayName = 2832742109; /// Owner's name
        id = "qcs::cam::uin/2832742109:uin/2832742109"; /// Holder ID,
    };
          }
          }
```

## 4. Deleting a Bucket

### Example Code

```
/// Module name: cosClient. Obtain the module instance through the module name.
var demo = api.require('cosClient');
demo.deleteBucket({
    "serviceKey":"test",
    "region":"ap-chengdu",
    "bucket":"0-appid",},
    function(ret,err){
    /// When err is empty, it returns ""
        if(err != ""){
            alert(err.data);
        }else{
            alert(ret.data);
        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |
| region | String | Bucket Region | Required |
| bucket | String | Bucket name format: BucketName-APPID | Required |

- Response parameters
  - Deleted successfully

```
{
 "result":"success"
       }
```

  - Deletion failed and returned error information.

```
{
 "result":"error",
 "error":{} /// Error information
       }
```

## 5. Creating a Bucket

### Example Code

```
/// Module name: cosClient. Obtain the module instance through the module name.
var demo = api.require('cosClient');
demo.createBucket({
    "serviceKey":"test",
"name":"apicloudtest",
"appId":"appId",
"region":"ap-chengdu"},function(ret,err){
    /// When err is empty, it returns ""
        if(err != ""){
            alert(err.data);
        }else{
            alert(ret.data);
        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
|---|---|---|---|
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |
| name | String | Bucket name does not include appid. | Required |
| appId | String | User `appid` | Required |
| region | String | Bucket Region | Required |
| accessControlList | String | Defines the ACL attribute of a bucket. Valid values: private, public-read-write, public-read; Default: private | Not required |
| readAccount | String | Grant read permissions to the authorized user, id="OwnerUin"; | Not required |
| writeAccount | String | Grant write permissions to the authorized user. Format: id="OwnerUin"; | Not required |
| readWriteAccount | String | Grant read and write permissions to the authorized user. Format: id="OwnerUin"; | Not required |
| enableMAZ | Boolean Value | Use Multi-AZ by default: No | Not required |

- Response parameters
  - Created successfully

```
{
 "result":"success"
       }
```

  - Creation failed and returned error message.

```
{
 "result":"error",
 "error":{} /// Error information
       }
```

## 6. Retrieving Bucket Contents

### Example Code

```
var demo = api.require('cosClient');
demo.listBucketContent({"serviceKey":"test","bucket":"0-appid","region":"ap-chengdu"},function(ret,err){
    /// When err is empty, it returns ""
        if(err != ""){
            alert(err.data);
        }else{
            alert(ret.data);
        }
});
```

## Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |
| region | String | Bucket Region | Required |
| bucket | String | Bucket name format: BucketName-APPID | Required |
| prefix | String | Prefix match, used to specify the prefix address of the returned file. | Not required |
| delimiter | String | A symbol used to limit the results of a list operation. If a particular prefix is specified, identical paths between the prefix and the delimiter will be grouped together and defined as a common prefix, and then all common prefixes will be listed. If no prefix is specified, the listing will start from the beginning of the path. | Not required |
| marker | String | Entries are listed in UTF-8 binary order by default, with all entries starting from the marker. | Not required |
| maxKeys | String | Maximum number of entries returned at a time. Default is 1000. | Not required |

- Response parameters
  - Loading succeeded

```
{
 "result":"success",
 "data":{
     "Contents":[
{
             "ETag": "\"c57b7cd3647561480b355a92dc2e971a\"", // The MD5 checksum value calculated based
on the object content,
For instance, "22ca88419e2ed4721c23807c678adbe4c08a7880"; ensure to include double quotes before and
after.
             "Owner":{
                 "ID": "1253960454", // The complete ID of the object owner, in the format of
qcs::cam::uin/[OwnerUin]:uin/[OwnerUin],
For example, qcs::cam::uin/100000000001:uin/100000000001, where 100000000001 is the UIN.
                 "DisplayName": "1253960454" // Name of the object owner
             },
             "StorageClass": "STANDARD", // Standard type
             "Key":"02_Common_Syntax.pdf", // Filename
             "LastModified":"2023-01-09T11:15:25.000Z", // Last modified time
             "Size": 816838 // File size
         }
     ],
     "MaxKeys":1000,
```

```
      "IsTruncated": "False", // Indicates whether the response request items are truncated, with values
'true' or 'false'
      "CommonPrefixes": [  // Group paths with the same prefix between Prefix and delimiter, defined as
Common Prefix
{
              "Prefix": "1\/" // The prefix of a single Common Prefix
        }
      ],
      "Delimiter":"\/", // Delimiter
      "Name": "0-1253960454" // The name of the storage bucket, formatted as BucketName-APPID, such as
examplebucket-1250000000
        }
        }
```

- Retrieving failure and returning error details

```
{
 "result":"error",
 "error":{} /// Error information
        }
```

## 7. Uploading Files

### Example Code

```
var uploadTaskId;
var demo = api.require('cosClient');
demo.uploadObject({
    "serviceKey":"test",
    "region":"ap-chengdu",
    "bucket":"0-appid",
    "filePath":"file://test.jpg",
    "object":"test.gif"},function(ret,err){
        if(err != ""){
            alert(err.data);
        }else{
            if(ret.result == "begin"){
                // Obtain the taskId at the beginning, which is used to cancel the task.
                uploadTaskId = JSON.parse(ret.data).taskId;
                alert(uploadTaskId);
        }
        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |
| region | String | Bucket Region | Required |
| bucket | String | Bucket name format: BucketName-APPID | Required |
| object | String | File name uploaded to the server | Required |
| filePath | String | Local File Path | Required |

| | | | |
|---|---|---|---|
| uploadId | String | No need to set during the first file upload, but required for subsequent resumable uploads. | Not required |
| stroageClass | String | Storage class of an object | Not required |
| trafficLimit | String | For the download traffic control speed limit value, it must be a numeric value with the default unit of bit/s. The speed limit range is 819,200 – 838,860,800, i.e., 800 Kb/s – 800 Mb/s. If it exceeds this range, a 400 error will be returned. | Not required |
| enableVerification | String | Upon completion of the upload, the file's MD5 returned by COS is compared with the locally calculated MD5. This feature is enabled by default. If the verification fails, the file will still be uploaded to COS, but a verification failure error will be thrown locally. | Not required |

- Response parameters
  - Upload Initialization Phase

```
{
 "result":"begin",
 "data":{
    "taskId": taskId, // Current task ID for user to cancel the task.
    "uploadId": uploadId // The current upload ID, used for resuming uploads.
        }
        }
```

  - Upload Phase

```
{
 "result":"processing",
 "data":{
    "totalBytesSent": totalBytesSent, // Current transfer progress.
    "totalBytesExpectedToSend": totalBytesExpectedToSend // Total file size.
        }
        }
```

  - Complete
  - Successful Uploads

```
{
 "result":"success",
"data": {} // File information
        }
```

  - Upload failure and return of error information

```
{
 "result":"error",
 "error":{} /// Error information
        }
```

## 8. Suspend Upload

### Example Code

```
var demo = api.require('cosClient');
// The uploadTaskId is obtained from the upload interface and saved by the business.
demo.pauseUploadObject({"taskId":uploadTaskId});
```

### Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| taskId | String | Retrieve from the upload interface, saved by the business, for canceling the current task. | Not required |

## 9. Downloading Files

### Example Code

```
var downloadTaskId;
var demo = api.require('cosClient');
demo.registerTransferManger({"serviceKey":"test","useHttps":true})
demo.uploadObject({
    "serviceKey":"test",
    "region":"ap-chengdu",
    "bucket":"0-appid",
    "filePath":"0-1253960454",
    "object":"test.gif"},function(ret,err){
        if(err != ""){
            alert(err.data);
        }else{
            if(ret.result == "begin"){
                // Obtain the taskId at the beginning, which is used to cancel the task.
                downloadTaskId = JSON.parse(ret.data).taskId;
                alert(downloadTaskId);
        }
        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |
| region | String | Bucket Region | Required |
| bucket | String | Bucket name format: BucketName-APPID | Required |
| object | String | Server-side File Name | Required |
| versionId | String | File Version ID | Not required |
| localPath | String | Local File Path | Required |
| trafficLimit | String | For the download traffic control speed limit value, it must be a numeric value with the default unit of bit/s. The speed limit range is 819,200 - 838,860,800, i.e., 800 Kb/s - 800 Mb/s. If it exceeds this range, a 400 error will be returned. | Not required |
| enableVerification | String | After the upload is completed, should the file MD5 returned by COS be verified against the locally calculated MD5? This feature is enabled by default. If the verification fails, the file will still be uploaded to COS, but a verification failure error will be thrown locally. Default verification is used. | Not required |

- Response parameters
  - Download Initialization Phase

```
{
 "result":"begin",
 "data":{
    "taskId": taskId, // Current task ID for user to cancel the task.
        }
        }
```

- Download Stage

```
{
 "result":"processing",
 "data":{
    "totalBytesDownload": totalBytesDownload, // Current transfer progress.
    "totalBytesExpectedToDownload": totalBytesExpectedToDownload // Total file size.
        }
        }
```

- Complete
- Downloaded successfully

```
{
 "result":"success",
"data": {} // File information
        }
```

- Download failed and returned error message

```
{
 "result":"error",
 "error":{} /// Error information
        }
```

## 10. Pause Download

### Example Code

```
var demo = api.require('cosClient');
// The uploadTaskId is obtained from the download interface and saved by the business.
demo.pauseDownloadObject({"taskId":downloadTaskId});
```

### Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| taskId | String | Retrieve the download interface to be saved by the business, used for canceling the current task. | Not required |

## 11. Retrieve File Information

### Example Code

```
var demo = api.require('cosClient');
demo.headObject({
    "serviceKey":"test",
    "region":"ap-chengdu",
    "bucket":"0-appid",
    "object":"test.jpg"},function(ret,err){
        if(err != ""){
            alert(err.data);
```

```
        }else{
            alert(ret.data);
        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |
| region | String | Bucket Region | Required |
| bucket | String | Bucket name format: BucketName-APPID | Required |
| object | String | Server-side File Name | Required |
| versionId | String | File Version ID | Not required |

- Response parameters
  - Loading succeeded

```
{
 "result":"success",
 "data":{} // File object information
        }
```

  - Retrieving failure and returning error details

```
{
 "result":"error",
 "error":{} /// Error information
        }
```

## 12. Pause Download

### Example Code

```
var demo = api.require('cosClient');
// The uploadTaskId is obtained from the download interface and saved by the business.
demo.pauseDownloadObject({"taskId":uploadTaskId});
```

### Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| taskId | String | Retrieve the download interface to be saved by the business, used for canceling the current task. | Not required |

## 13. Delete a File

### Example Code

```
var demo = api.require('cosClient');
demo.deleteObject({
    "serviceKey":"test",
    "region":"ap-chengdu",
    "bucket":"0-1253960454",
```

```
    "object":"test.jpg"},function(ret,err){
        if(err != ""){
            alert(err.data);
        }else{
            alert(ret.result);

        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |
| serviceKey | String | COS service instance unique identifier; if not provided, the default service instance will be used for network requests. | Not required |
| region | String | Bucket Region | Required |
| bucket | String | Bucket Name-AppID | Required |
| object | String | Server-side File Name | Required |
| versionId | String | File Version ID | Not required |

- Response parameters
- Deleted successfully

```
{
 "result":"success",
        }
```

- Deletion failed and returned error information.

```
{
 "result":"error",
 "error":{} /// Error information
        }
```

## 14. Cancel All Request Tasks

### Example Code

```
var demo = api.require('cosClient');
demo.cancelAll({
    "serviceKey":"test"
    },function(ret,err){
        if(err != ""){
            alert(err.data);
        }else{
            alert(ret.result);

        }
});
```

### Description

- Request Parameters

| Parameter | Local Disk Types | Note | Required |
| --- | --- | --- | --- |

| | | | |
|---|---|---|---|
| serviceKey | String | Unique identifier for the COS service instance, cancel network requests initiated by the corresponding service instance. | Not required |

- Response Parameters
  None

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

Last updated: 2023-09-20 12:17:03

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. Register a Tencent Cloud account

(If you already have a Tencent Cloud account, skip this step.)

Click here to register a Tencent Cloud

#### Step 2. Verify your identity

(If you have already done so, skip this step.)

Click here to complete identity verification.

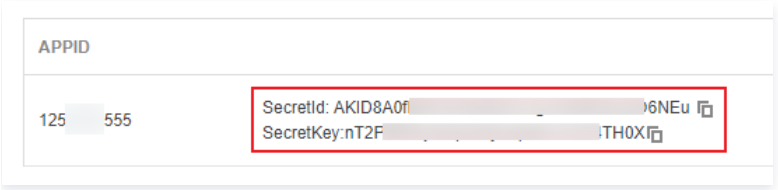
For a detailed verification process, please refer to Identity Verification Introduction.

#### Step 3. Activate COS service

Click here to activate the COS service.

#### Step 4. Prepare the APPID and access key

Obtain and record the APPID, **SecretId**, and **SecretKey** from the **API Key Management** page in the Access Management 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 process, feel free to contact us. When reaching out, please mention that you followed the guidance in this document and provide information such as the application name and screenshots, so we can assist you more efficiently.

| Common Configuration Item Name | Note |
| --- | --- |
| Provider/Service Provider/Storage Service Provider/Service Provider/Storage Provider/Provider, etc. | Here, the primary consideration is which storage type the application should use, which may involve the following scenarios:<br>If there is an option with wording like S3 Compatible Storage/S3 Compatible, prioritize using this option.<br>If you only see terms like Amazon Web Services/AWS/Amazon S3, select that option first. In the subsequent configurations for server endpoint, service address, service URL, Endpoint, Custom Endpoint, or Server URL, pay attention to the related instructions for these settings.<br>If there are no similar options, but the application description mentions support for S3 or S3-compatible services, you can proceed with configurations such as service endpoint, service address, service URL, Endpoint, Custom Endpoint, or Server URL. Pay attention to the related instructions for these configuration items.<br>In other cases, we apologize, but the application may not be able to use COS. |
| Service Endpoint/Service URL/Endpoint/Custom Endpoint/Server URL, etc. | Here, enter the service address for the S3-compatible service. When using COS, fill in the COS service address in the format: `cos.<Region>.myqcloud.com` or `https://cos.<Region>.myqcloud.com`.<br>Whether to include `https://` depends on the specific application, and you can try it yourself. The `<Region>` represents the available region of COS.<br>In the application, you can only create or select a bucket in the region specified by the service address.<br>• For example, if your bucket is located in the Guangzhou region, the service address should be configured as `cos.ap-guangzhou.myqcloud.com`. If you configure it for another region, you will not be able to find the bucket in the Guangzhou region within the application.<br>• If the application's service provider only allows you to select `Amazon S3` and the service endpoint is configurable, you can change the endpoint to `cos.<Region>.myqcloud.com` or `https://cos.<Region>.myqcloud.com` as mentioned earlier.<br>• If the service endpoint is not configurable or there is no service endpoint configuration option, your application cannot use COS. |
| Access key, access key ID, etc. | Enter the SecretId noted in Step 4. |
| Secret key, secret, secret access key, etc. | Enter the SecretKey obtained in Step 4. |
| Region, etc. | Select "Default", "Auto", or "Automatic". |
| Bucket, etc. | Select or enter an existing bucket name in the format `<BucketName-APPID>`, such as `examplebucket-1250000000`. Here, `BucketName` is the bucket name entered in Step 5 when creating the bucket, and `APPID` is the `APPID` recorded in Step 4. As mentioned earlier, the bucket will be limited to the region specified by the service address, and buckets from other regions will not be listed or may not function properly. If you need to create a new bucket, the new bucket name must also follow the `<BucketName-APPID>` format, or the bucket creation will fail. |

## 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 Ports and Protocols
  COS supports both HTTP and HTTPS protocols, using the default ports 80 and 443 respectively. For security reasons, we recommend using COS via the HTTPS protocol.
- COS supports Virtual Hosted Style.
- AWS V2 Signature and AWS V4 Signature
  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.
Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# Storing Discuz! Forum Remote Attachments in COS

Last updated: 2023-09-12 18:37:31

## Feature Overview

Discuz! forum can configure remote attachment functionality to store forum attachments on Tencent Cloud Object Storage (COS), which offers higher reliability and data persistence for forum attachments.

> **Note**
> This practice is applicable to Discuz V3.4 version. Other versions may have compatibility issues and may not be usable.

## Preparations

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.

## Practical Steps

### Building Discuz! Forum

Tencent Cloud Marketplace offers Discuz! images. If you are not familiar with Linux commands, it is recommended to deploy Discuz! forum using image deployment. If you are proficient in using Linux and have higher scalability requirements for your business website, you can also manually set up Discuz! forum.

### Prepare a COS Bucket

1. Create a bucket with **public-read/private-write** access permissions. It is recommended that the bucket's region is the same as the region of the CVM running the Discuz! forum. For more information on creating a bucket, see the Creating Bucket documentation.
2. In the bucket list, locate the newly created bucket and click **Configuration Management**.
3. In the left sidebar, select the **Overview** tab, view the **Access Domain Name** and take note of it.

### Install the COS Plugin

> **Note**
> The COS plugin currently supports only Discuz V3.4 version. Other versions may have compatibility issues and may not be usable.

1. In your local browser, visit the address `CVM Public IP/index.php`. Log in with the administrator account and enter the **Admin Center** page.
2. At the top of the page, click **Plugins**.
3. In the plugin page, click **Get More Plugins**. Search for "Tencent Cloud Object Storage" in the Discuz! App Center, select it, and click "Install App."
4. After the installation is complete, enter the **Plugin List** to view the installed COS plugin.
5. Click **Settings** to start configuring the COS plugin. The **configuration items** are explained as follows:

| Configuration items | Description |
| --- | --- |
| Tencent Cloud Secret ID, Tencent Cloud Secret Key | Access key information can be created and obtained in Cloud API Key. |
| Regions | When creating a bucket, it is recommended to choose a region that is the same as the region of the CVM running the Discuz! forum. |
| Bucket Name | When creating a bucket, use a custom bucket name, such as examplebucket-1250000000. The attachment files are uploaded to the COS bucket with a date-named path, in the format `forum/YYYYMM/DD`, for example, `forum/202212/16`. |

6. After verifying that the configuration information is correct, click **Submit** to proceed.

## Posting Test

1. Navigate to the forum homepage, click **Default Section**, click **Post**, enter a title and content, and add an image within the content.
2. After completing the image upload and passing the verification code, click **Post Thread**.
3. Click **Admin Center > Plugins > Tencent Cloud Object Storage 1.0.2**, click **Upload images and attachments to COS**, then click **Start**. Wait for the attachments to be successfully uploaded, and you can find the uploaded image attachments in the COS bucket.

> **Note**
>
> Attachment files are uploaded to the COS bucket with a date-based naming format: `forum/YYYYMM/DD`, for example, `forum/202212/16`.

## Summary

Of course, COS not only offers the above applications and services but also provides a variety of popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# Storing Remote WordPress Attachments to COS

Last updated: 2023-09-20 11:46:03

## Feature 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 plug-ins, 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.

## Preparations

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.

## Practical Steps

### 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)
- Manually Building a WordPress Personal Website (Windows)

You can deploy a WordPress website via an image easily as follows:

1. Deploy WordPress using an image.
   1.1 Log in to the CVM console and click **Create Instance** on the instance management page.
   1.2 Follow the on-screen instructions to select the instance type, and click **Image Marketplace** under **Instance Configuration > Image**. Choose **Select from Image Marketplace**.
   1.3 In the "Image Marketplace" pop-up, select **Basic Software** and enter **WordPress** in the search field.
   1.4 Choose an image as needed, for example, select **WordPress Blogging Platform_v5.5.3 (CentOS | LAMP)** and click **Use for Free**.
   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 the new version 6.0.2.
   Click on the left side menu of the dashboard, enter the "Updates" menu, and update to the new version 6.0.2.

For more information, see Deploying a WordPress Personal Website Using an Image.

### Creating COS bucket

1. Create a **Public Read/Private Write** bucket as instructed in Creating a Bucket, preferably in the same region as the CVM instance where WordPress is running.
2. Find the bucket you created on the **Bucket List** page and click its name to enter the details page.
3. Click the **Overview** tab on the left sidebar. Then, find and record the endpoint.

## Installing and Configuring Plugin

You can install the plugin in the plugin library or via the source code.

> **Note:**
> The COS plugin is currently only compatible with WordPress version 6.0.2, which supports direct installation of the COS plugin from the plugin library. Other versions may have compatibility issues and may not be installable.

### Installation in the plugin library (recommended)

In the WordPress admin panel, click **Plugins**, search for **tencentcloud-cos**, and click **Install Now**.

### Using 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 on the "Plugins" menu on the left side, and you will see the plugin. Click to enable it.

### Configuring the plugin

Configure the COS bucket information in the tencent-cloud-cos plugin:

1. Click the "Settings" button to configure the tencent-cloud-cos plugin.
2. Configure the COS information as follows:

| Configuration items | Description |
| --- | --- |
| SecretId,SecretKey | Access key information can be created and obtained in Cloud API Key. |
| Region | The region you chose when you created the bucket |
| Bucket Name | The bucket name customized during bucket creation, such as `examplebucket-1250000000` |
| Access Endpoint Domain Names | The default COS bucket domain is generated by the system based on the bucket name and region when you create a bucket. Different regions have different default domain names. You can view the domain information in the Object Storage Console under Bucket 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 Thumbnail | After this option is enabled, thumbnail files will not be uploaded. |
| CI | By enabling CI, you can edit, compress, convert formats, and add watermarks to images. For more information, please refer to CI Product Introduction. |
| File Moderation | Enabling file moderation provides intelligent security checks for multimedia content such as images, videos, audio, text, documents, and webpages. This helps users effectively identify prohibited content like pornography, illegal activities, and offensive materials, thus avoiding operational risks. For more information, see Content Moderation Overview. |
| Previewing document | By enabling file preview, you can convert files into images, PDFs, or HTML5 pages, addressing the display issues of document content. For more information, please refer to File Preview Overview. |
| Debugging | This feature logs errors, exceptions, and warnings. |

3. After configuring, click **Save Configuration** to finish.

## 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. Create a post with an image. In the WordPress dashboard, click on the "Posts" menu on the left side.
2. Edit the default "Hello, World!" post in WordPress.
3. Click the "+" button on the right.
4. Next, select an image to upload.
5. After uploading, check the URL of the uploaded image to confirm that it is a COS address, such as `https://wd-125000000.cos.ap-nanjing.myqcloud.com/2022/10/summer-1200x675.jpeg`. The format is: `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/<ObjectKey>`, indicating that the image has been uploaded to the COS bucket.
6. Log in to the COS console, and you will see the image you just uploaded in the COS bucket.

> **Note**
> After successful testing, if you need to synchronize old resources to the COS bucket (using COSCMD tool or COS Migration tool), **otherwise, the old resources cannot be previewed properly in the backend**. Once synchronization is complete, you can enable origin-pull settings by referring to Setting Origin-pull below.

## Others

1. Accelerated access with CDN
   To configure CDN acceleration for your storage bucket, refer to the CDN Acceleration Configuration documentation. In the plugin settings, change the URL prefix to the default CDN acceleration domain name or a custom acceleration domain name.
2. Replace resource URLs in the database
   For existing websites, the database contains old resource links. The plugin provides a replacement feature to update these URLs. Remember to back up your data before performing the initial replacement.
   - 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. To set up cross-origin access, you may encounter an error message `No 'Access-Control-Allow-Origin' header is present on the requested resource` when referencing resource links in your articles. This is due to the absence of the header. You need to add an HTTP Header configuration in the CORS settings. Here are two ways to configure it:
   - Configuring in the COS Console

     > **Note**
     > For information on configuring cross-origin access, see Setting Cross-Origin Access documentation.

- 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. Origin-pull configuration
   If you don't upload resources to the WordPress media library, it's recommended to enable origin-pull. For detailed steps, refer to the Origin-pull configuration document.
   Once enabled, when a client accesses a COS source file for the first time and COS cannot find the object, it returns a 302 HTTP status code and redirects the client to the origin-pull address. The object is then provided by the origin server, ensuring access. Meanwhile, COS copies the file from the origin server and saves it to the corresponding directory in the storage bucket. On the second access, COS directly retrieves the object and returns it to the client.

## Summary

Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# Backing up Files from PC to COS

Last updated: 2023-09-12 18:38:37

## Preamble

Data is invaluable, and many people can attest to this. The loss of digital photos, electronic documents, work outputs, or game saves can be quite distressing. File loss due to hard drive failure, accidental deletion, computer crashes, or software malfunctions, as well as the awkwardness of being asked to "roll back a version" only to find that no historical versions were saved, are all vexing issues in both work and life. Therefore, the importance of backups cannot be overstated.
When it comes to backups, many people think of using portable hard drives or setting up NAS storage within a local area network and simply uploading files to it. But is it really that simple in practice?
Backup is, in fact, a systematic process. In addition to copying files to the backup medium, it is also necessary to verify the accuracy of the backup content. Both copying and verification tasks need to be performed regularly to minimize losses in the event of file loss. Moreover, the backup medium itself requires maintenance, such as timely replacement of damaged hard drives.
In light of the above, is there a simpler way to ensure the safety of files? The answer is affirmative.
With the development of cloud services, we now have reliable enterprise-grade cloud storage services, such as Tencent Cloud Object Storage (COS). As broadband speeds increase and costs decrease due to national initiatives, backing up files to the cloud has become a reality. What we need next is a software solution that connects files on our computers to cloud storage, automatically backs up our files to the cloud on a regular basis, and periodically verifies the accuracy of the backup files.

## Software Introduction

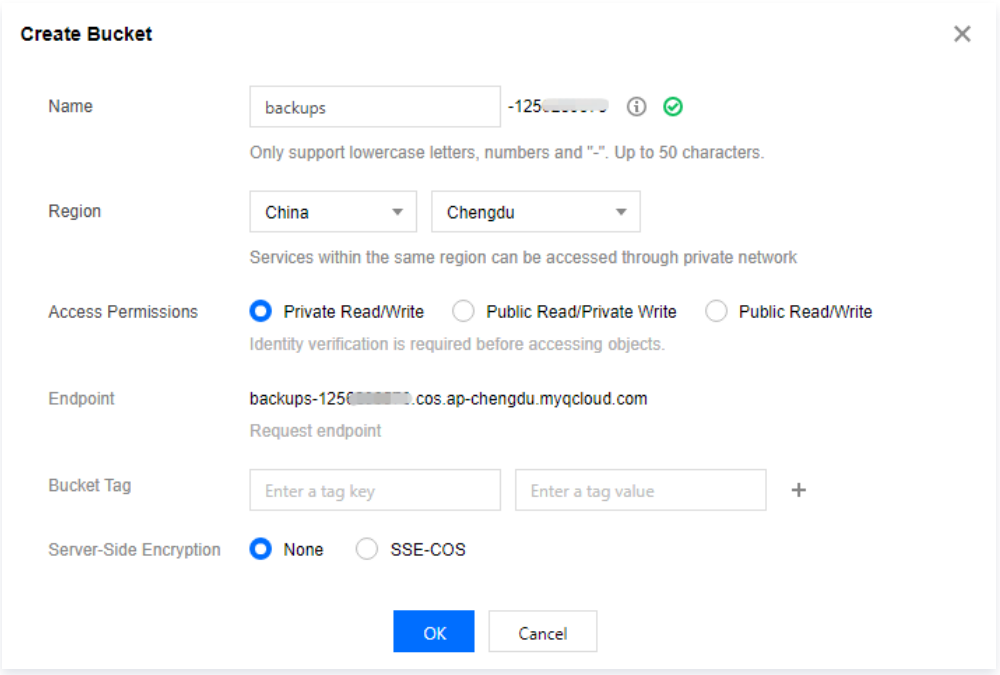
Arq® Backup is a commercial backup software that supports Windows and macOS systems. The software runs in the background and automatically backs up specified directories at regular intervals according to the configuration. It also retains backups from each time point, making it easy to find historical versions of a file. Moreover, backups at each time point only include differential files, and duplicate files in different paths are backed up only once, minimizing backup size and maximizing speed. Before transferring backup files to the network, the software encrypts them based on the user's input password, ensuring that they are not misused during transmission or while stored in the cloud, thus safeguarding the security of sensitive data.
Arq® Backup offers a commercial license for $49.99 per user, which allows usage on a single computer. Additionally, the software provides a 30-day free trial, allowing users to test it before making a purchase.

> **Note**
> Currently, there is no Simplified Chinese version of Arq® Backup software. The download, purchase, and related instructions can be found on the software's official website.

## Prepare Tencent Cloud Object Storage

> **Note**
> If you are currently using COS, please skip steps 1 and 2.

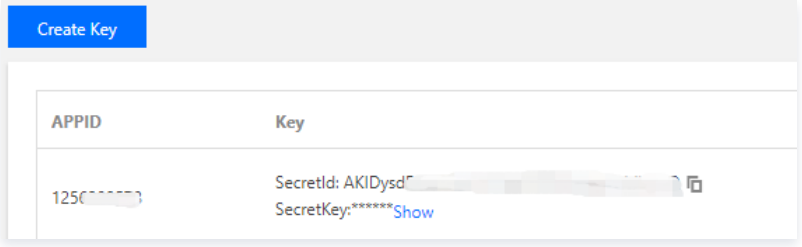
1. Sign up for a Tencent Cloud account and complete identity verification.
2. Log in to the COS console and follow the prompts to activate COS.
3. In the Cloud Object Storage (COS) console, click on the **Bucket List** in the left sidebar, and then click **Create Bucket** to start creating a bucket:
   - Name: Bucket name, for example, "backups".
   - Region: You can choose a region close to your location, but please do not select a financial region. Currently, we offer price discounts for the Southwest region, so you can also choose "Chengdu" or "Chongqing" to enjoy more favorable pricing.




Keep other configuration options at their default settings, copy and save the **Request Domain** address, and then click **Confirm** to complete the creation.

> **Note**
> For detailed steps on creating a bucket, please refer to Creating a Bucket.

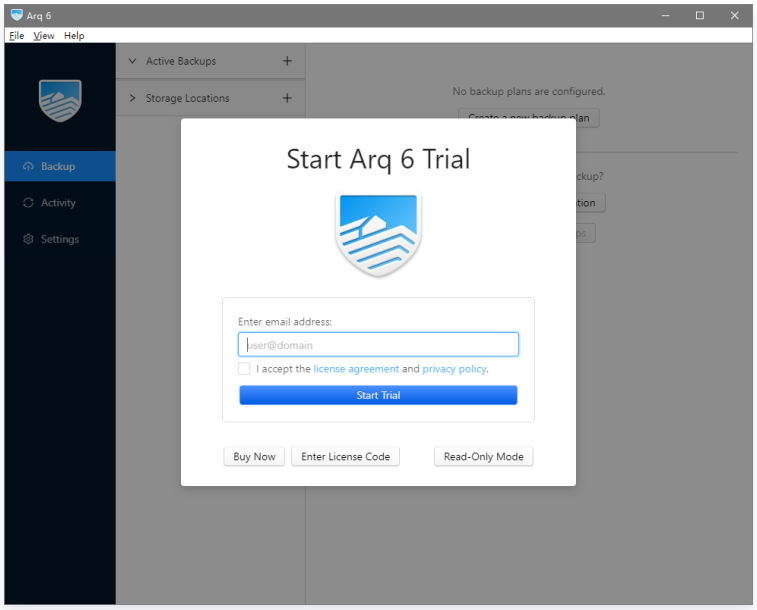
4. Log in to the API Key Management Console and create and record the key information SecretId and SecretKey.



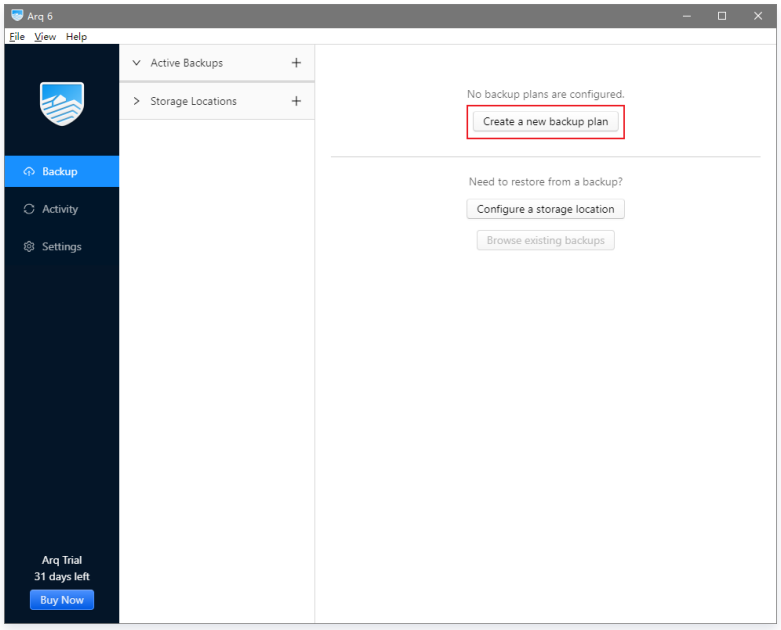

## Install and Configure Arq® Backup

> **Note**
> This article uses Arq® Backup version 6.2.11 for Windows as an example.

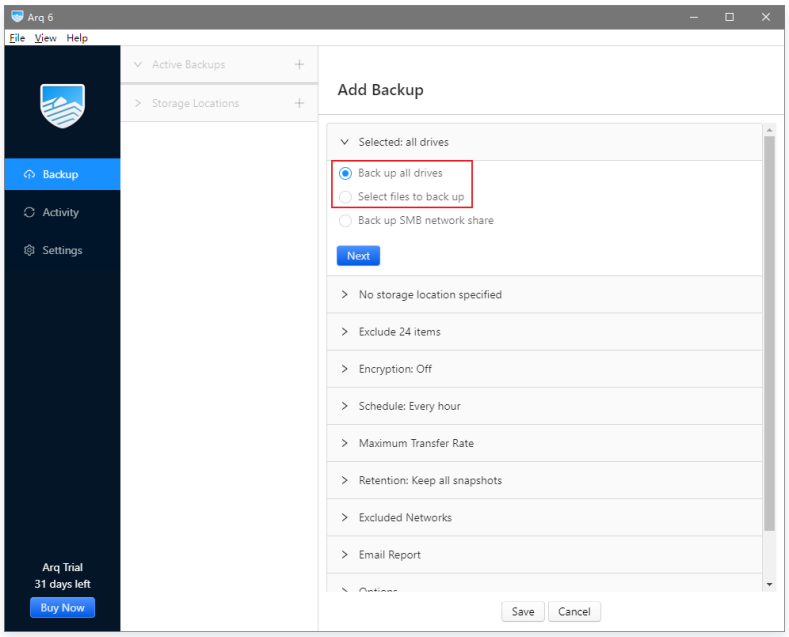
1. Download the software from the Arq® Backup official website.
2. Follow the prompts to complete the software installation. After installation, the software will automatically launch. Upon the first launch, you will be prompted to log in. Enter your email address and click **Start Trial**.



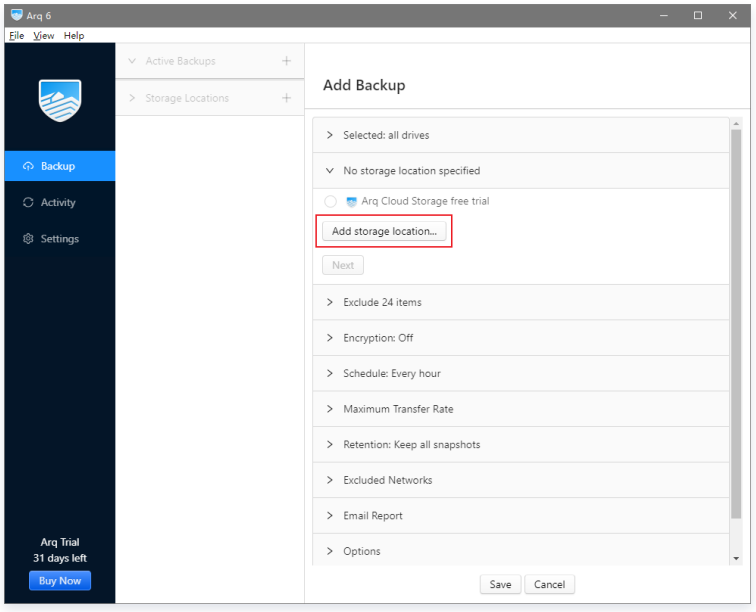

3. In the **Backup** interface, click **Create a new backup plan** to add a backup plan.



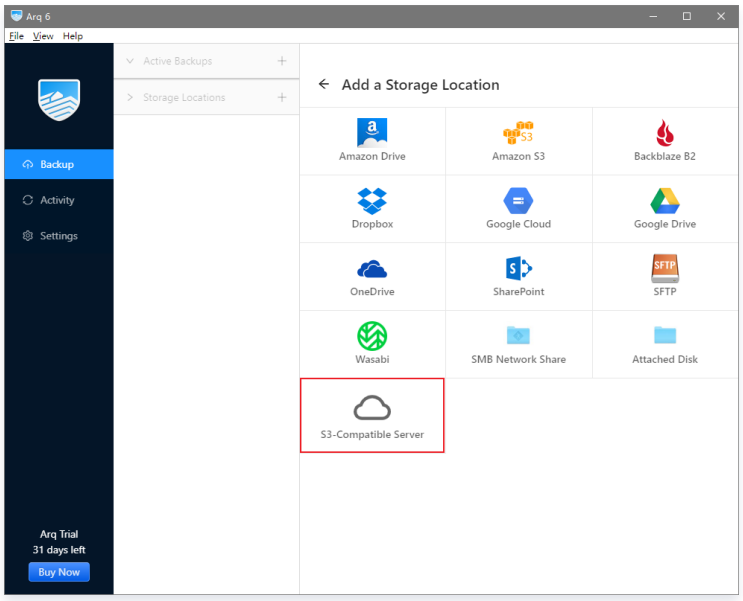
4. In the interface, select the directory you want to back up. You can choose all hard drives or a specific directory.



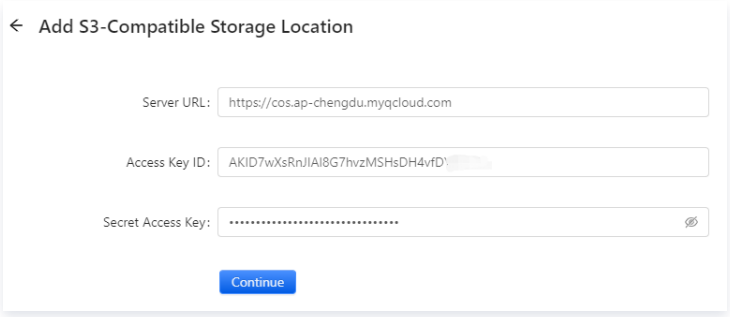

5. Click **Add Storage Location** to add a backup storage location, as shown below.



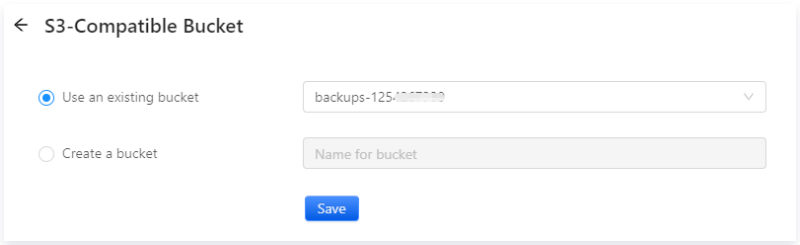
6. Here, we select **S3-Compatible Server**.



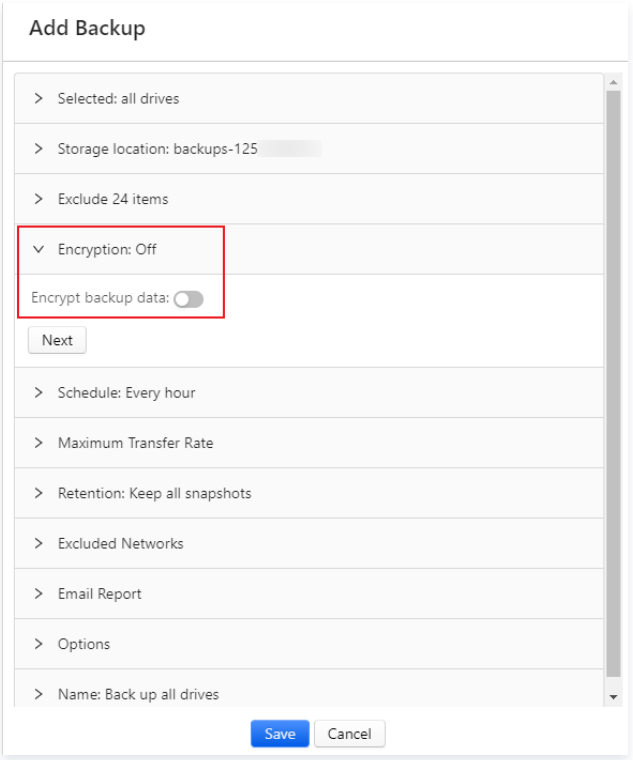

7. In the redirected interface, follow the instructions below to configure. Once completed, click **Continue**.
   - Server URL: Enter the portion of the previously recorded request domain name starting with `cos`, and prepend `https://` to it, for example, `https://cos.ap-chengdu.myqcloud.com`. Please note that this does not include the bucket name.
   - Access Key ID: The SecretId found in the previously recorded key information.
   - Secret Access Key: The SecretKey from the key information recorded earlier.



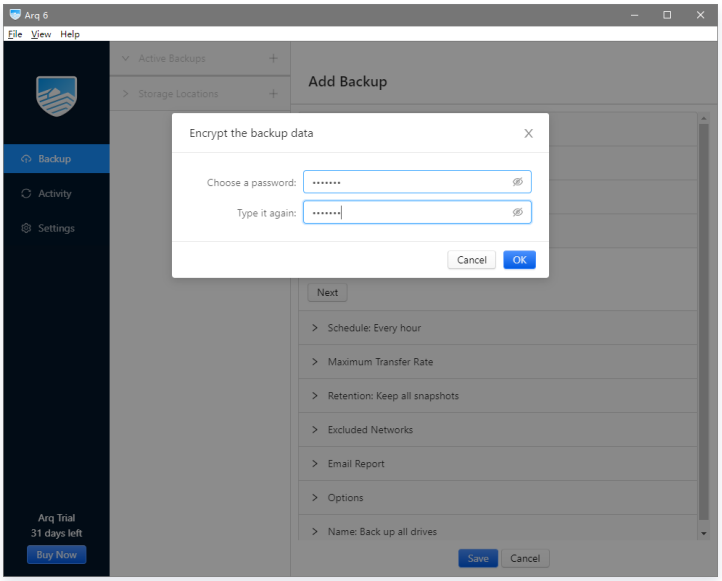

8. In the subsequent interface, select **Use an existing bucket** and choose the bucket created earlier, such as **backups-1250000000**, then click **Save**.



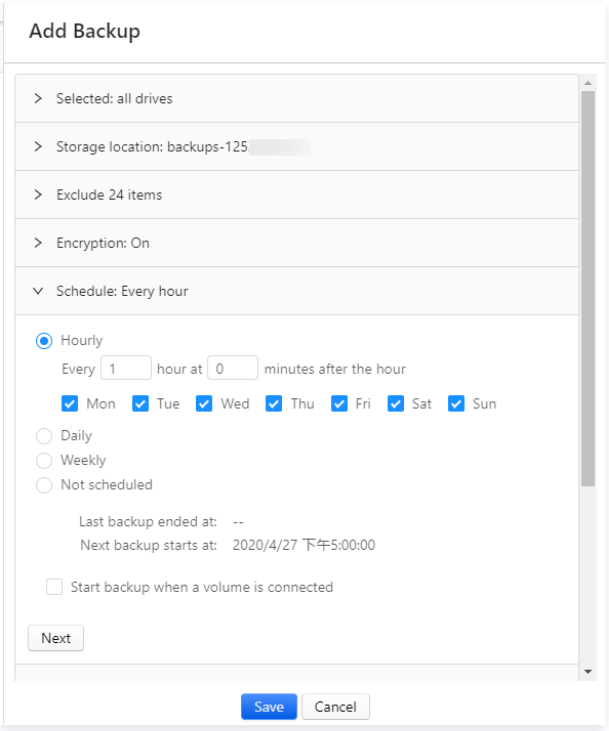
9. (Optional) Choose whether to encrypt the backup data; here, we select the **Enable** button.



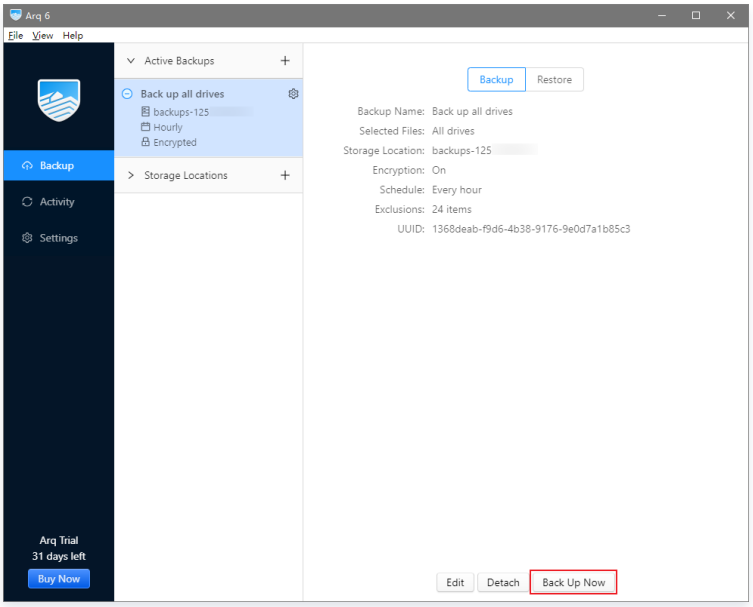

10. In the pop-up window, set the password for encryption. Enter the encryption password for the backup file twice and click **OK**. **Please remember the backup password, as you will not be able to restore files from the backup without it!**



11. (Optional) Set the backup cycle.

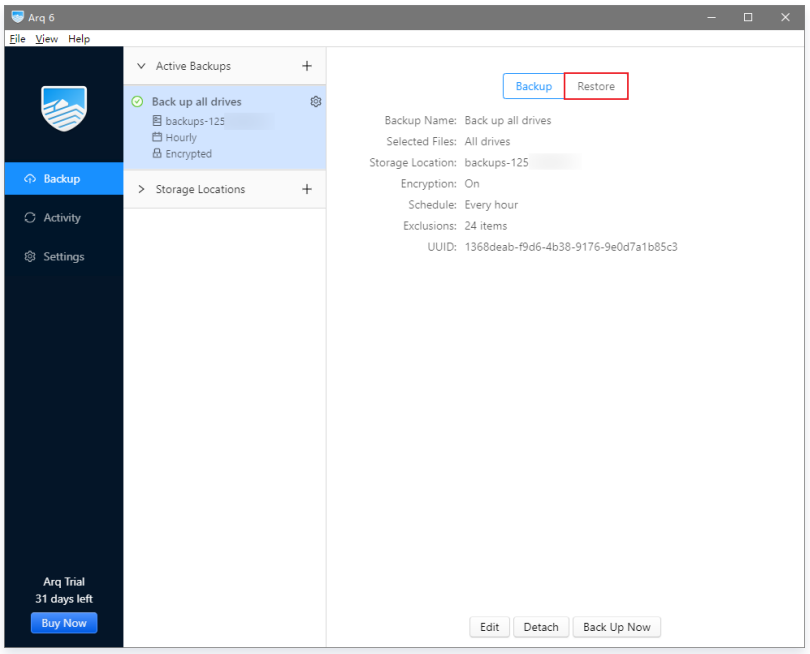



12. Click **Save** to save the settings, and then click **Back Up Now** to start the backup.

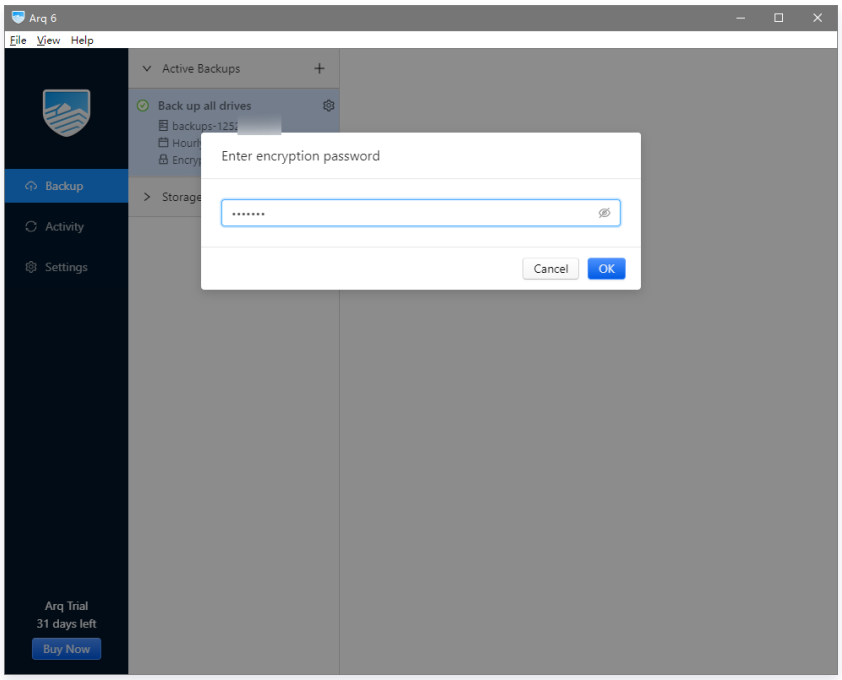



## Restoring Files from Backups

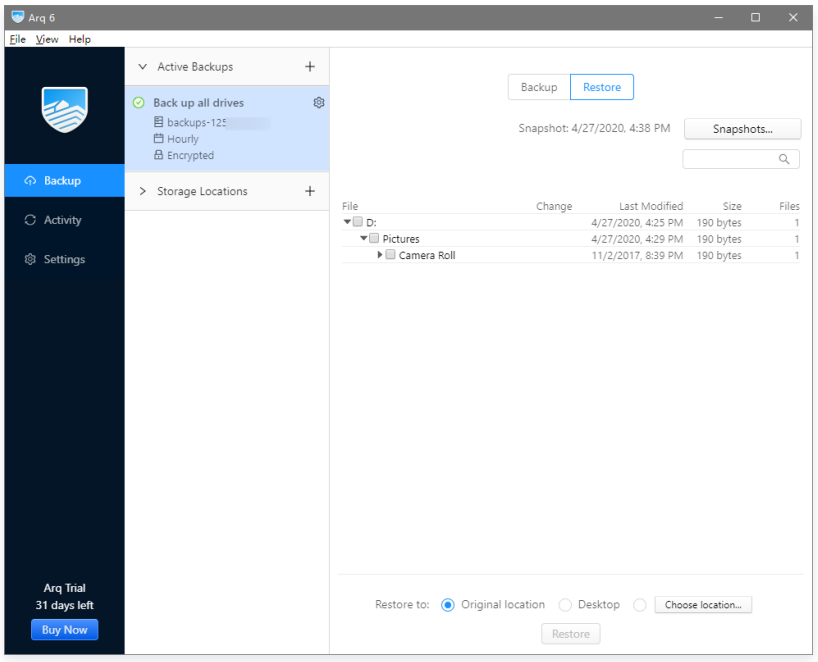
1. In the left **Backup** list on the main interface, click **Restore**.



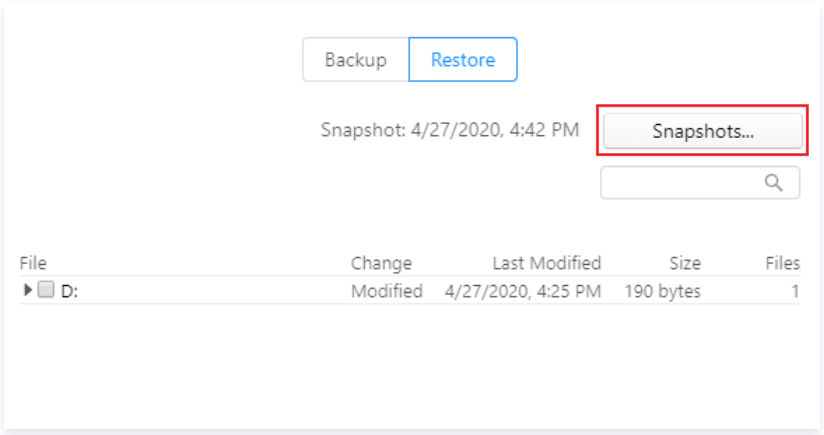

2. If you have set up encrypted backup data according to step 9 above, you will need to enter the password.



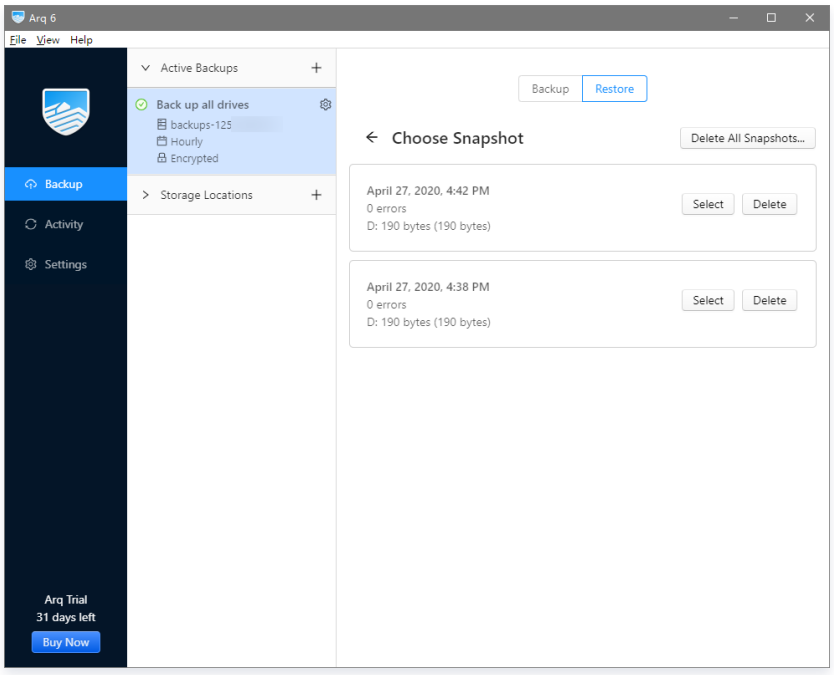

3. Select the directory or file to be restored, as well as the location to save the restored directory or file, and click **Restore** to begin the recovery process.




4. The recovery operation defaults to restoring from the latest backup. If needed, you can find historical version backups in snapshots and restore from those. Click **Snapshots** to view historical snapshots.



5. Opt for historical snapshots.




6. Select the historical directory or file to be restored, as well as the location to save the restored directory or file, and click **Restore** to begin the recovery process.
7. Upon receiving the recovery completion prompt, you can view the restored files in the previously specified directory.

## Summary

Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# Mounting COS to Windows Server as Local Drive

Last updated: 2025-09-10 17:46:09

## Scenario

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**
> This practical example is applicable to Windows 7, Windows Server 2012, 2016, 2019, and 2022 systems.

## Instructions

### 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. Visit GitHub to download Winfsp. For this example, we used version winfsp-1.12.22301. After downloading, follow the default installation steps.

   > **Note**
   > For Windows Server 2012 R2, WinFsp 1.12.22242 is not compatible, but WinFsp 1.11.22176 is compatible.

2. Visit the Git official website or GitHub to download the Git tool. For this example, we downloaded Git-2.38.1-64-bit. After downloading, follow the default installation steps.
3. Visit the Rclone official website or GitHub to download the Rclone tool. The version used in this example is rclone-v1.60.1-windows-amd64. No installation is required; simply extract the files to any English directory (errors may occur if the path contains Chinese characters). In this example, the path is E:\AutoRclone.

> **Note**
> The download speed from GitHub might be slow or inaccessible; you can download from other official channels if needed.

### 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, locate **This PC** in the left navigation pane, right-click and select **Properties > Advanced System Settings > Environment Variables > System Variables > Path**, then 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, enter the command `rclone --version`, and press **Enter** to check if Rclone is installed successfully.
4. After confirming the successful installation of Rclone, enter `rclone config` in Windows PowerShell and press **Enter**.
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**.

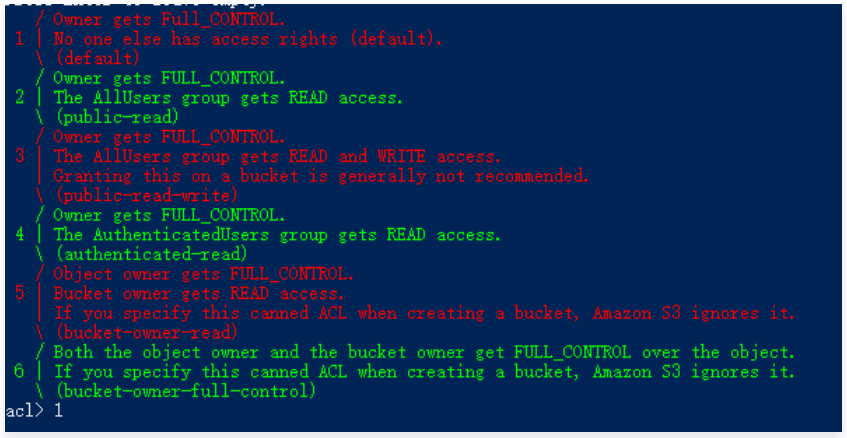
7. In the displayed options, choose the one containing "Tencent COS" by entering **5** and pressing **Enter**.


8. In the displayed options, choose the one containing "TencentCOS", enter **21**, and press **Enter**.

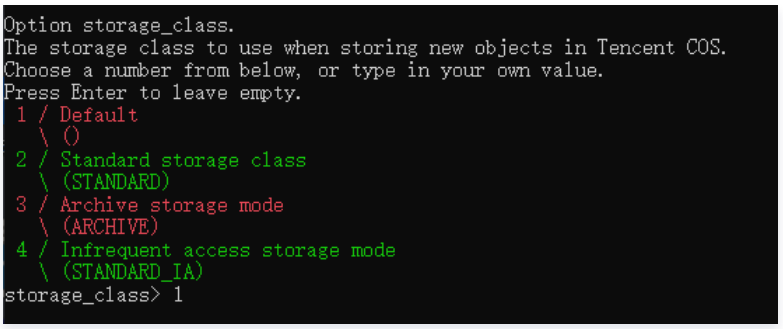

9. Press Enter when executing `env_auth>`.
10. When prompted for `access_key_id`, enter the Tencent Cloud COS SecretId and press **Enter**.

> **Note**
> It is recommended to use sub-account permissions. You can visit API Key Management to view your SecretId and SecretKey.

11. When prompted for `secret_access_key>`, enter the Tencent Cloud COS SecretKey and press **Enter**.
12. Based on the displayed Tencent Cloud gateway addresses for each region, check the bucket's region and select the corresponding one. In this example, we use Guangzhou, so choose `cos.ap-guangzhou.myqcloud.com`, input **4**, and press **Enter**.
13. In the displayed Tencent Cloud COS permission types, choose from options like default or public-read based on your needs. The selected permission type applies to newly uploaded files. In this example, we use default. Enter **1** and press **Enter**.




14. In the displayed Tencent Cloud Object Storage types, you can choose the storage type to upload files to COS based on your needs. In this example, we use Default. Enter **1** and press **Enter**.




- Default: default option
- Standard storage class: STANDARD
- Archive storage mode: ARCHIVE
- Infrequent access storage mode: STANDARD_IA

> **Note**
> To set up Intelligent Tiering or Deep Archive storage types, please use the **Modify Configuration File** method. In the configuration file, set the value of storage_class to INTELLIGENT_TIERING or DEEP_ARCHIVE. For more information about storage types, see Storage Class Overview.

15. When prompted with `Edit advanced config? (y/n)`, press **Enter**.
16. After confirming the information, press **Enter**.
17. Enter **q** to complete the configuration.


## Modifying the configuration file

After completing the steps above, a configuration file named rclone.conf will be generated, usually located in the `C:\Users\Username\AppData\Roaming\rclone` folder. If you want to modify rclone's configuration, you can edit it directly. If you cannot find the configuration file, run the `rclone config file` command in the command window to view it.

## 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:\temp --vfs-cache-mode
     writes &
     ```

   - 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 with a custom disk name chosen by the user.

- Y: Replace with the desired drive letter after mounting, ensuring it does not conflict with local drive letters such as C, D, or E.
- E:\temp is the local cache directory, which can be customized. Ensure that the user has the necessary directory permissions.

If the message "The service rclone has been started" appears, the mounting is successful.

2. Enter **exit** to close the terminal.
3. In your local computer's **My Computer**, you can find a disk named myCOS(Y:). Open the disk to view all the bucket names in the Guangzhou region. Now, you can perform common local disk operations such as uploading, downloading, creating, and deleting.

> **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:\temp --vfs-cache-mode
writes &
```

   - 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 modify the paths in the code to match your actual paths.

4. Cut the `startup_rclone.vbs` file to the `%USERPROFILE%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup` directory.
5. Restart the server.

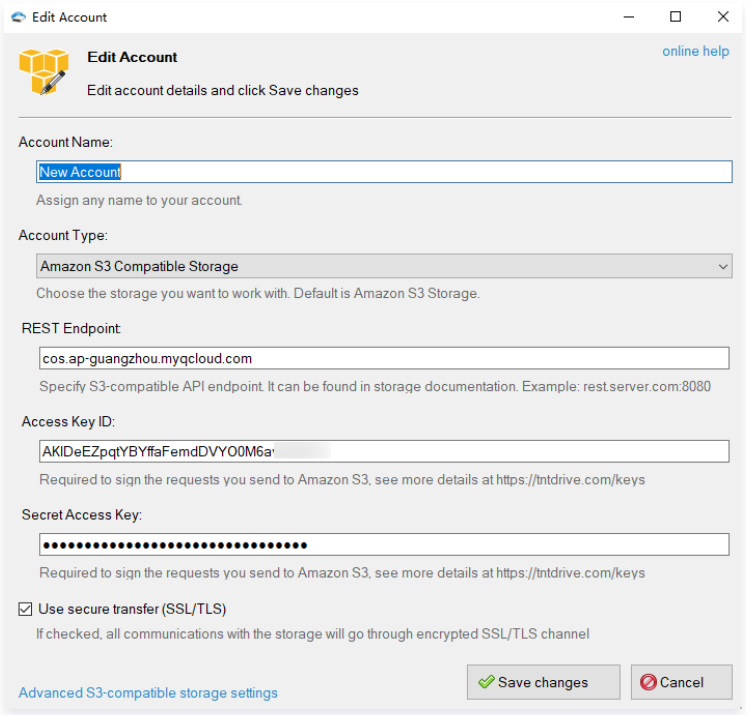
> **Note**
> Usually, the successful mounting prompt will be displayed tens of seconds after auto-mounting configuration is complete and the server is restarted.

# Related Actions

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 click **Account > Add New Account** to create a user account.



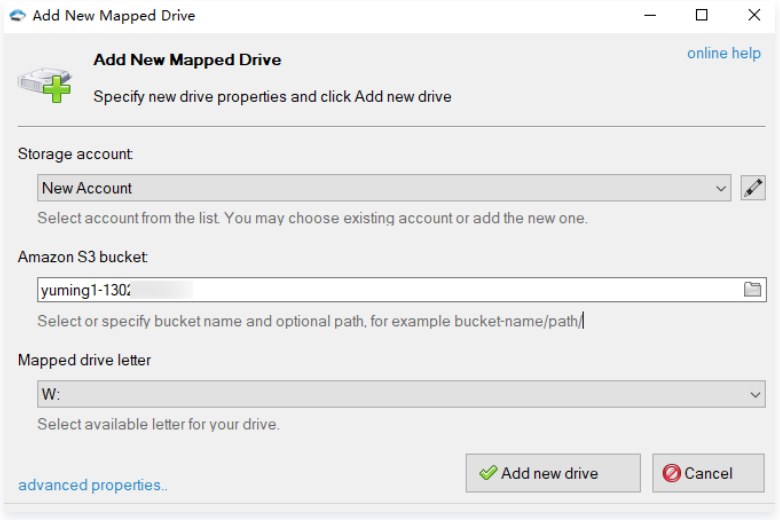

The main parameters are as follows:

- **Account Name**: user-defined account name
- **Account Type**: Since COS is compatible with S3, you can select **Amazon S3 Compatible Storage** here.
- **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`.

> **Note:**
> Buckets created after January 1, 2024, do not support using path-style domain names (format: `cos.<Region>.myqcloud.com`). For more details, please refer to the COS Bucket Domain Name Security Management Notice (Effective January 2024).

- **Access Key ID**: Enter the SecretId, which can be created and obtained in the API Key Management page.
- **Secret Access Key**: the value of `SecretKey`.

3. Click **Add new account.**
4. In the TntDrive interface, click **Add New Mapped Drives** to create a Mapped Drive.




The main parameter information is as follows:

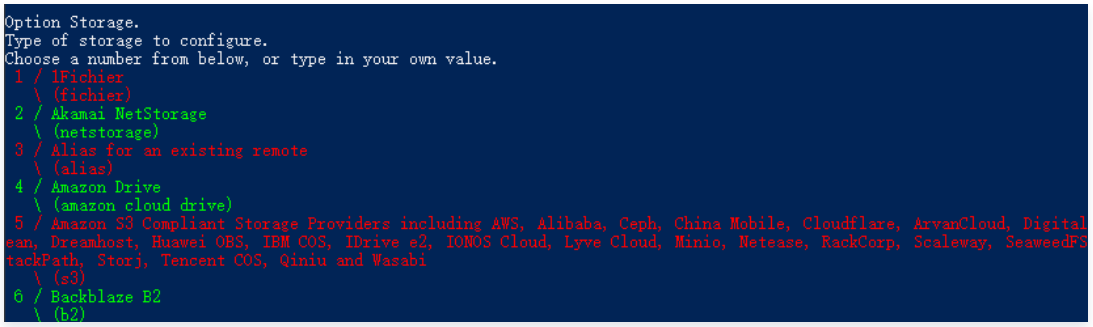
- **Amazon S3 bucket**: path or name of the bucket. You can click the icon on the right to select a bucket. In this example, the bucket residing in the Guangzhou region that is set in step 2 is selected (mapping a bucket as a single network drive).
- **Mapped drive letter**: drive letter of the mapped drive, which cannot overlap with existing drive letters.

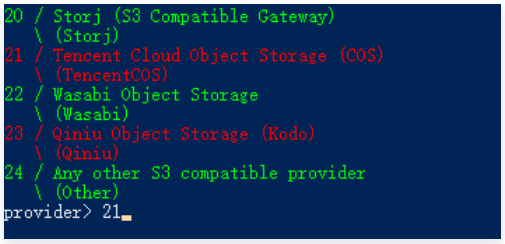
5. Click **Add new drive**.
6. Find the drive in **This PC**. If you want to map all buckets to the Windows server, repeat the steps above.

## Reasons for Mounting Failure

1. Check if the following parameters are configured correctly:

- Ensure that SecretId and SecretKey are entered correctly.
- When selecting Option Storage, make sure to choose TencentCOS.




2. If there is an error, modify the parameters and restart the server to remount.
3. If you are using a non-Administrator built-in account for the mounting operation, there may be issues with not seeing drive letters or directories within the drive. If you must use a non-Administrator account, please carefully disable the Windows UAC feature, and a restart is required for it to take effect. The command to disable Windows UAC is as follows:

```
reg add "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\" /v EnableLUA /t REG_DWORD /d 0 /f
```

## Summary

Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

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

Last updated: 2023-09-12 18:40:03

## Feature 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:

- **Cost-effective:** Low storage unit price, pay-as-you-go billing, and resource pack discounts available.
- **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.

### Instructions

1. Install PicGo and set relevant COS parameters.

   > **Note**
   > In this tutorial, PicGo version 2.3.1 is used. The configuration process may vary for other versions; please make necessary adjustments accordingly.

   After downloading and installing PicGo from the PicGo official website, navigate to the image hosting settings, find **Tencent Cloud COS**, 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 AppId: A unique user-level resource identifier for COS access. It can be obtained at Manage API Key.
   - Set storage region: Specify the region where the bucket is located. For a list of available regions, refer to the Available Regions documentation, such as ap-beijing, ap-hongkong, eu-frankfurt, etc.
   - 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:
   2.1 In Typora's **Image** preferences settings, configure as follows:

- Select **Upload image** for **When Insert....**
- In **Upload Service Settings**, select **PicGo(app)**, and configure the location of the recently installed PicGo.exe.

2.2 Restart Typora for the settings to take effect.

2.3 Enter the Typora editor area, and simply drag and drop or paste an image to upload it and automatically replace it with a COS file link. (If the link is not automatically replaced with a COS link after pasting, you can check if the server settings in PicGo are 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.

## Instructions

### 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.

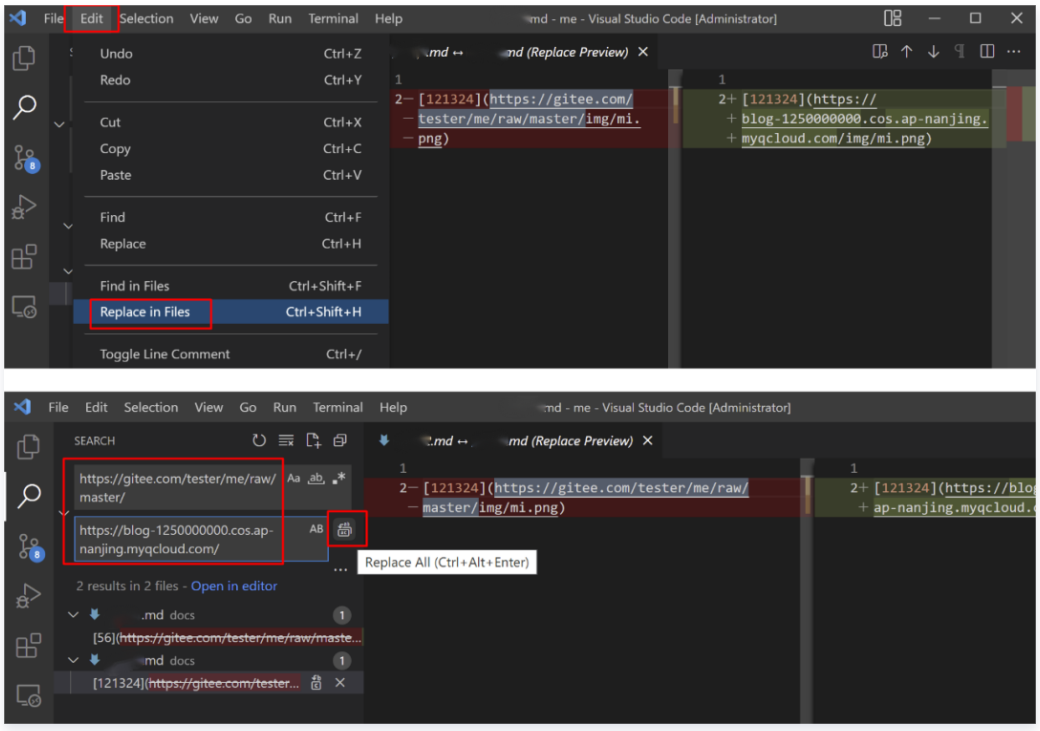
> **Note**
> If the number of images is high, you can also use COSBrowser to upload images quickly.

### Step 4. Globally replace the domain name

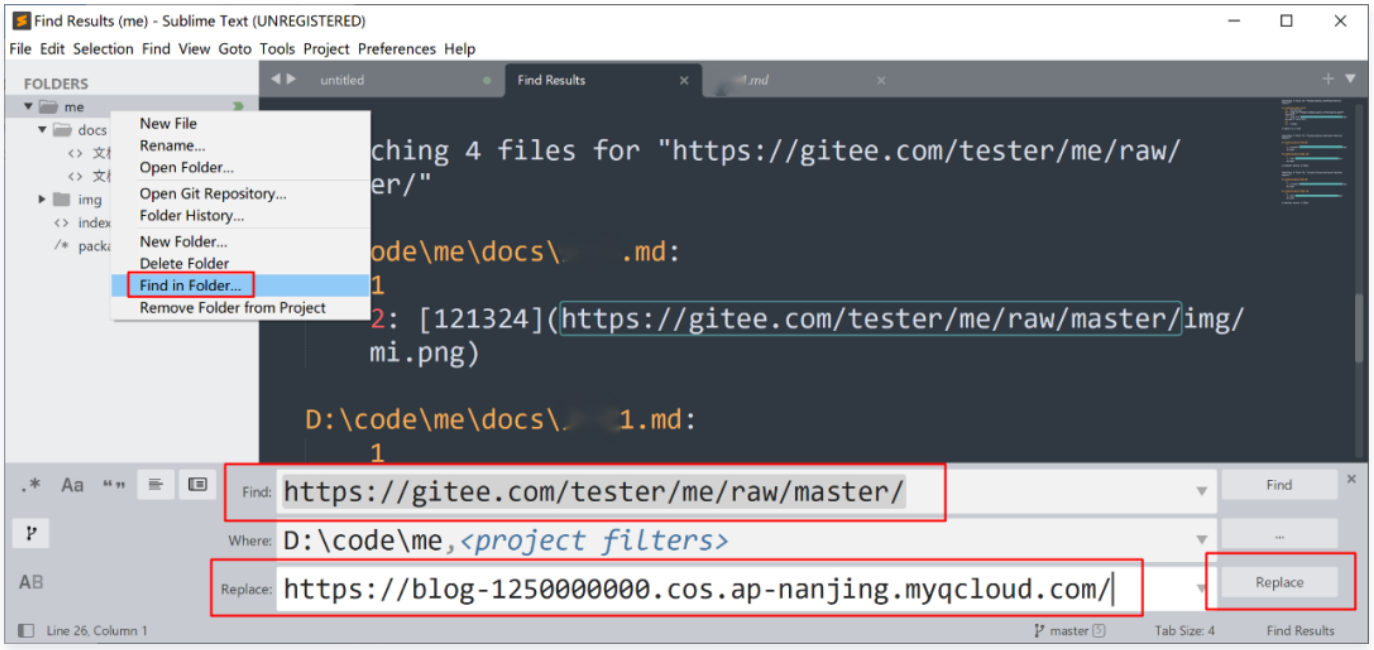
On the bucket overview page in the COS console, copy the default domain name of the bucket (you can also associate a custom CDN acceleration domain name). Then, use a common code editor to search for and replace the invalid URL prefix globally with the default domain name of the COS bucket.

> **Note**
> For more information on the default domain name, see Regions and Access Endpoints.

- Example search-and-replace with Visual Studio Code:




- Sublime Text search and replace example:




## Summary

Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# Managing COS Resource with CloudBerry Explorer

Last updated: 2025-04-18 17:25:00

## Feature 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.

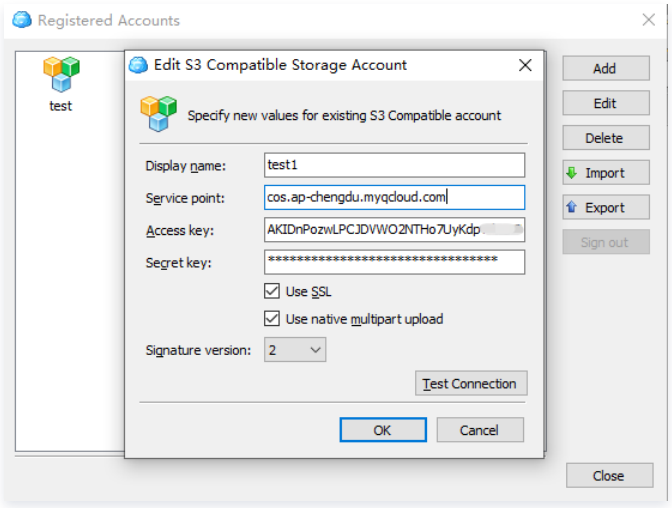
## Download Address

Download CloudBerry Explorer here.

## Installation and Configuration

> **Note**
> The following configuration steps are based on CloudBerry Explorer Windows v6.3. The configuration process for other versions may differ slightly, so please adjust accordingly.

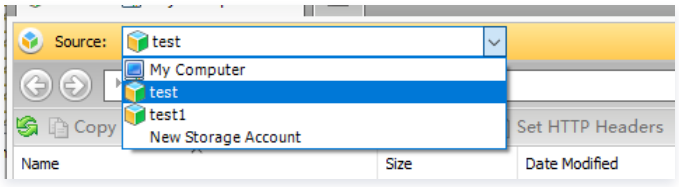
1. Double-click the installation package and complete the installation as prompted.
2. Open the tool and double-click **S3 Compatible**.
3. Configure the following information in the pop-up window and click Test Connection. If the connection is successful, it will be displayed.




The configuration items are explained below:

- Display name: Enter a custom username.
- Service point: The format is `cos.<Region>.myqcloud.com`. For example, to access a bucket in the Chengdu region, enter `cos.ap-chengdu.myqcloud.com`. For applicable region abbreviations, please refer to Regions and Access Domain Names.
- Access key: Enter the access key SecretId. You can go to the API Keys page of the console to create and view access keys. |
- Secret key: Enter the access key SecretKey. You can go to the API Keys page of the console to create and view access keys. |

4. After adding the account information, select the previously set username in the Source section to view the bucket list under that username, indicating that the configuration is complete.



# Managing COS File

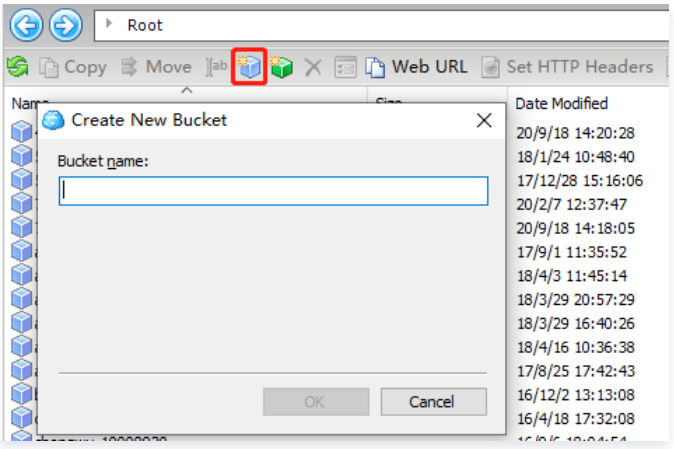
## Querying the bucket list

Select the configured username in **Source** to view the list of buckets under the username.

> **Note**
> You can only view buckets corresponding to the region configured in the Service point. To view buckets in other regions, click **File > Edit Accounts**, select the username, and change the Service point parameter to another region.

## Create a bucket.

Click the icon in the image, and enter the complete bucket name in the pop-up window, such as examplebucket-1250000000. After entering the correct name, click **OK** to complete the creation.
For bucket naming conventions, please refer to Bucket Naming Conventions.

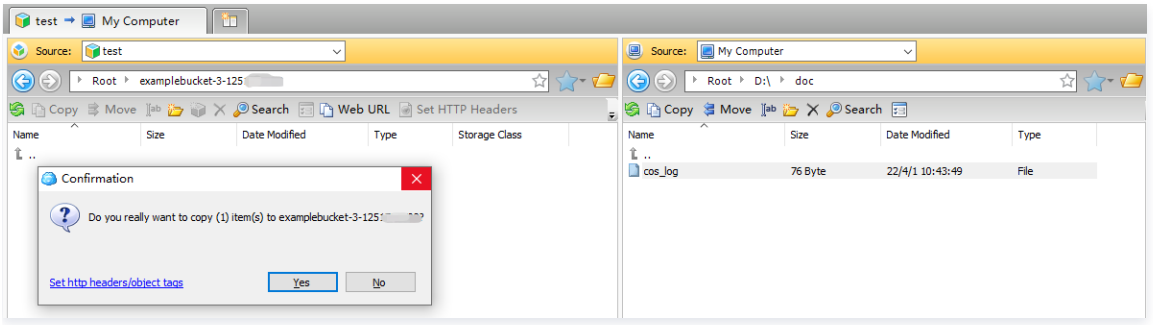



## Deleting a bucket

Right-click the target bucket in the bucket list and select **Delete** in the context menu.

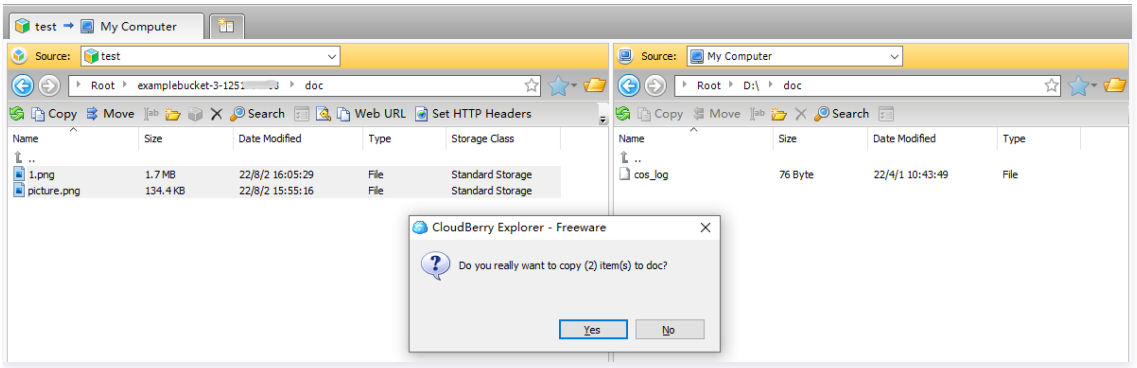
## Uploading object

Select the bucket or path where you want to upload the object from the bucket list, then choose the object to be uploaded from your local computer and drag it into the left window to complete the upload process.



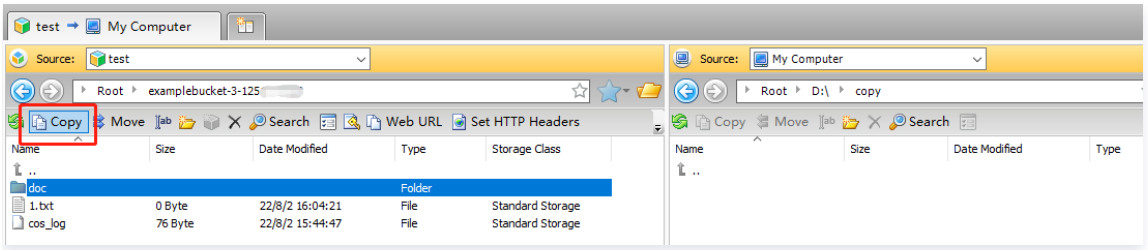

## Downloading object

Select the desired object in the left window and drag it to the folder on your local computer in the right window to complete the download process.




## Replicating Objects

In the right window, select the destination path for the copied object. Then, in the left window, choose the object to be copied, right-click **Copy**, confirm the pop-up information, and the object copying process is complete.




## Renaming an object

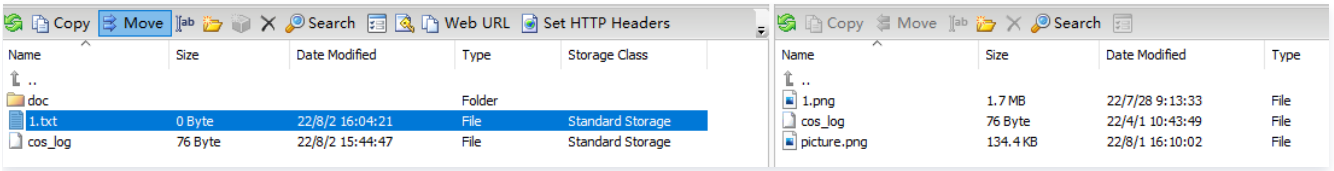
Right-click the target object in the bucket, select **Rename**, and enter a new name.

## Deleting an object

Right-click the target object in the bucket and select **Delete**.

## Moving an object

In the right pane, select the target path for the moved object. In the left pane, select the object to be moved, right-click **Move**, confirm the pop-up information, and the object moving operation is complete.




## Additional 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.

## Summary

Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# Complete data migration and recovery from self-built ES to Tencent Cloud ES using COS and snapshots

Last updated: 2025-04-17 18:26:19

## Overview

Tencent Cloud Elasticsearch Service (ES) is a highly available, scalable, cloud-managed Elasticsearch service built on the open-source search engine Elasticsearch, including Kibana and common plugins, and integrating advanced features such as security, SQL, machine learning, alerts, and monitoring (X-Pack). With Tencent Cloud ES, you can quickly deploy, easily manage, and scale your cluster on demand, simplify complex operations and maintenance, and quickly create various services such as log analytics, anomaly monitoring, website search, enterprise search, and BI analysis.
Tencent Cloud ES integrates the leading technological advantages of Tencent Cloud computing in areas such as computing, storage, and security. It also maintains the compatibility and openness of Elasticsearch itself. It has rich cluster management features as well as security, elasticity, and high availability features. At the same time, it also integrates official advanced commercial features (X-Pack). On the basis of being open-source, it adds features such as permission management, SQL, machine learning, and alarm. This helps you simplify basic Ops tasks such as cluster deployment and operational management, and focus more on the business itself.
Through Tencent Cloud ES, you can quickly build applications for massive data storage search and real-time log analysis, such as website search navigation, enterprise-level search, service log anomaly monitoring, clickstream analysis, etc.
The usage scenarios between ES and COS are mainly reflected in data migration and data backup recovery. The principle is actually a process where COS is used for intermediate storage of source ES data, and then the stored data is asynchronously restored to the target ES cluster.

## Preparations

Create a **public-read/private-write** bucket. For details on how to create it, see Creating a Bucket document.

> **Notes:**
> The region of the COS bucket must be the same as that of ES.

## Installing the COS Plug-In

Tencent Cloud ES has these plug-ins internally integrated by default. If it is a user-built ES cluster and COS is required, then the COS plug-in corresponding to the user's ES version needs to be installed.

### Features of COS Plug-In

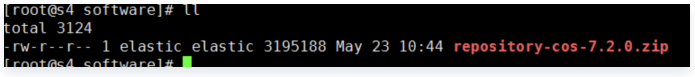
You can directly back up snapshot files from a user-built cluster to a COS bucket, and then perform recovery on the peer end.

### Download Plug-in

Go to Github to download the COS plug-in launched by Tencent Cloud ES. This practice takes the 7.2.0 version of the plug-in as an example. For other version differences, please see releases.

### Plug-In Installation Workflow

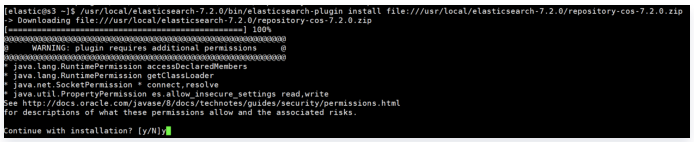
1. Retrieve the plug-in corresponding to the ES version. For example, version 7.2.0.
2. Grant all privileges of the plugin file to the ES startup account elastic.




3. Switch to a regular user account, install the plugin, and restart the ES service. Note: Perform the operation on each node of the cluster. Execute the command as follows.

```
bin/elasticsearch-plugin install file:///path/repository-cos.zip
```

As shown below:




4. After execution, you can use `get _cat/plugins` on kibana to confirm whether the installation is completed. If you get the following results, it indicates successful installation.

```
node2 repository-cos 7.2.0
node1 repository-cos 7.2.0
node3 repository-cos 7.2.0
```

## Use COS to Achieve Data Migration From Self-Built ES to Tencent Cloud ES

1. Register a COS Repository on a local cluster.

```
PUT _snapshot/my_cos_backup
{
 "type": "cos",
 "settings": {
     "access_key_id": "xxxxxx",
     "access_key_secret": "xxxxxxx",
     "bucket": "A bucket name without the appId suffix",
     "region": "ap-guangzhou",
     "compress": true,
     "chunk_size": "500mb",
     "base_path": "/yourbasepath",
     "app_id": "xxxxxxx"
 }
}
```

Execute this command will create a warehouse named my_cos_backup in the base_path directory of the corresponding bucket in COS.
The parameters are described as follows:

| Parameter Name | Parameter Description |
| --- | --- |
| access_key_id,access_key_secret | Access key information can be created and obtained in Cloud API Key. |
| bucket | Bucket name. Note that do not include the appid. |
| app_id | APPID is one of the account identifiers assigned by the system after successfully applying for a Tencent Cloud account. It can be viewed in Tencent Cloud Console [Account Info]. |
| region | The region where the bucket is located. For the abbreviation of COS region, please refer to regions and access endpoints. |
| base_path | Backup directory, such as /dir1/dir2/dir3. The initial `/` needs to be written. No `/` is needed at the end of the directory. |

> **Notes:**
> If you use version 8.15.1 of the plug-in, note that the parameter name has added the prefix cos.client. For example, cos.client.access_key_id.

2. Create a snapshot file in the local repository. The snapshot file will be automatically uploaded to the specified repository in COS. You can execute the snapshot by using `put _snapshot/repository name/snapshot name`.
3. Similarly, register a repository on Tencent Cloud ES. The repository name may not be the same as that above.

```
PUT _snapshot/my_cos_backup
{
 "type": "cos",
 "settings": {
     "access_key_id": "xxxxxx",
     "access_key_secret": "xxxxxxx",
     "bucket": "A bucket name without the appId suffix",
     "region": "ap-guangzhou",
     "compress": true,
     "chunk_size": "500mb",
     "base_path": "/yourbasepath",
     "app_id": "xxxxxxx"
 }
}
```

4. Perform recovery on Tencent Cloud ES. Run the following command to perform snapshot recovery.

```
POST _snapshot/repository name/snapshot name/_restore
```

> **Notes:**
> The name of the snapshot restored here is the name of the snapshot you created in the source cluster earlier.

## Summary

Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# Implementing Cloud Storage for Web Applications with Django + COS

Last updated: 2023-09-12 18:41:22

## Feature Overview

Django is an open-source web application framework based on Python, which greatly simplifies the development process of web applications. To better meet the needs of modern web applications, Django offers numerous extensions, including cloud storage. This article mainly discusses how to use the COS plugin to implement remote attachment functionality, storing Django application data on Tencent Cloud Cloud Object Storage (COS).

## Preparations

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.

## Environment Dependencies

- Python version: v3.8 or later.
- Django version: Greater than or equal to 2.2, and less than 3.3.

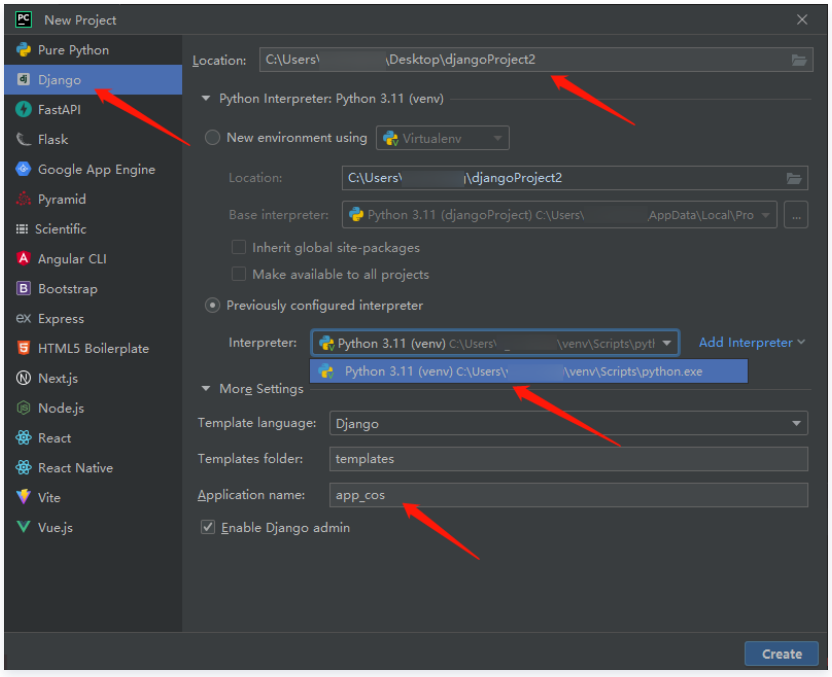
## Practical Steps

### Creating COS bucket

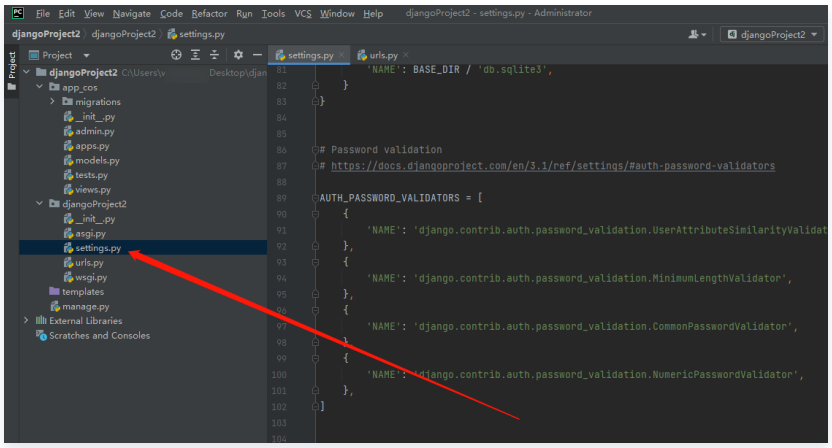
1. Create a bucket with **public-read/private-write** access permissions. It is recommended that the bucket's region matches the region of the CVM running Django. For more information on creating a bucket, please refer to the Creating Bucket documentation.
2. Locate the newly created bucket in the bucket list and obtain the bucket name, such as examplebucket-1250000000.

### Create Django Project

1. Visit the PyCharm Official Website and select the appropriate PyCharm version for your operating system.
2. After installing and opening PyCharm, click on NEW project or create project, and select Django from the options.



3. After creating the bucket, locate and open the setting.py file in your directory.



4. Copy and paste the following code, and configure the COS service according to the parameter descriptions:

```
DEFAULT_FILE_STORAGE = "django_cos_storage.TencentCOSStorage"

TENCENTCOS_STORAGE = {
 "BUCKET": "xxx",
 "CONFIG": {
     "Region": "ap-guangzhou",
     "SecretId": "xxxx",
     "SecretKey": "xxxx",
 }
}
```

The parameters are described as follows:

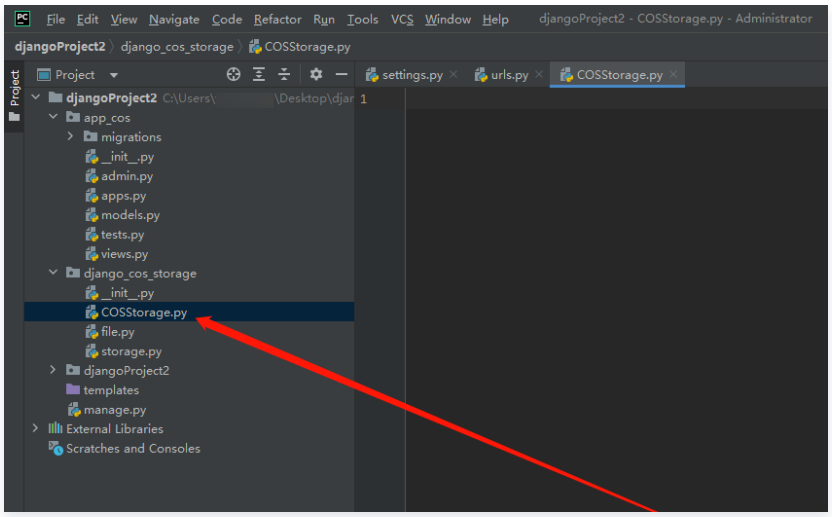
| Configuration items | Description |
| --- | --- |
| Bucket | The name customized during bucket creation such as `examplebucket-1250000000`. |
| Region | The region selected during bucket creation |
| SecretId | Access key information can be created and obtained in Cloud API Key. It is recommended to use a sub-account key and follow the principle of least privilege to reduce risks. For more information, see Sub-account Access Key Management. |
| SecretKey | Access key information can be created and obtained in Cloud API Key. It is recommended to use a sub-account key and follow the principle of least privilege to reduce risks. For more information, see Sub-account Access Key Management. |

## Download and Configure COS Plugin

1. Visit Github to download the COS plugin. After downloading, extract the django_cos_storage directory into your Django project directory.

> **Note**
> To view the plugin information, open the terminal and enter `pip freeze` to display the module information.

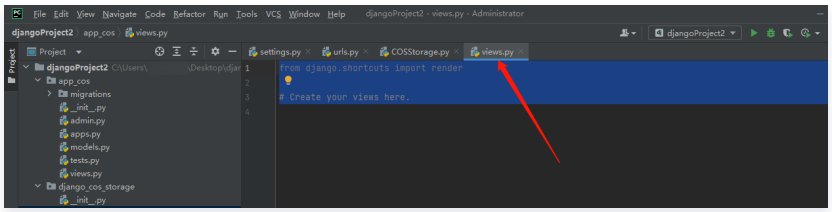
2. Create a Python file in the django_cos_storage directory, for example, COSStorage.py.




Copy and paste the following code into it:

```
from .storage import TencentCOSStorage
from functools import wraps

def  decorator(cls):

 instance = None
 @wraps(cls)
 def inner(*args,**kwargs):
     nonlocal instance
     if not instance:
         instance=cls(*args,**kwargs)
     return  instance
 return inner

@decorator
class QFStorage:
 def __init__(self):
     pass
     self.storage =TencentCOSStorage()
     self.bucket =self.storage.bucket
     self.client =self.storage.client

 Upload Object
 def upload_file(self, Key, LocalFilePath, PartSize=1, MAXThread=5, EnableMD5=False):
     #
     try:
         response =self.client.upload_file(
             Bucket=self.bucket,
             Key=Key,
             LocalFilePath=LocalFilePath,
             PartSize=PartSize,
             MAXThread=MAXThread,
             EnableMD5=EnableMD5
         )
         return response
     except Exception as e:
         print('Failed to upload object, error:', e)
         return None
```

3. Open the views.py file in the app_cos directory.

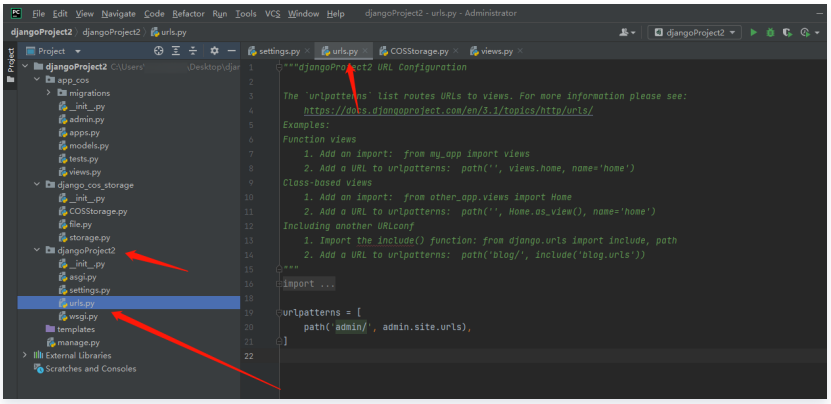


Copy and paste the following code:

```
from django.shortcuts import render,redirect
from django.http import HttpResponse
from django_cos_storage.COSStorage import QFStorage
from django.conf import settings


Upload Object

def upload_file_view(request):
 response=QFStorage().upload_file(
     Key='1.png',
     LocalFilePath=settings.BASE_DIR  /  'cessu/1.png'
 )

 if response:
     return HttpResponse('File uploaded successfully!')
 return HttpResponse('File upload failed')
```

> **Note**
> In this example, `cessu/1.png` represents creating a folder named cessu and uploading the image 1.png to the cessu folder. After a successful upload, you can find the image 1.png in the cessu folder of the COS bucket.

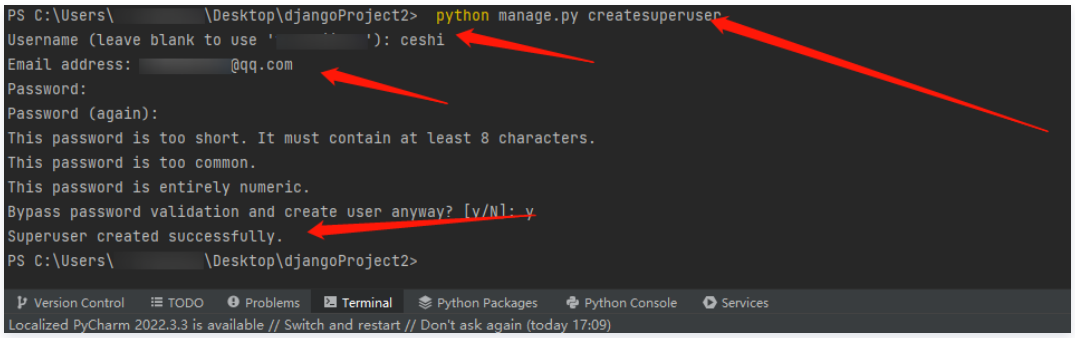
4. Locate and open the urls.py file in the djangoProject2 directory.



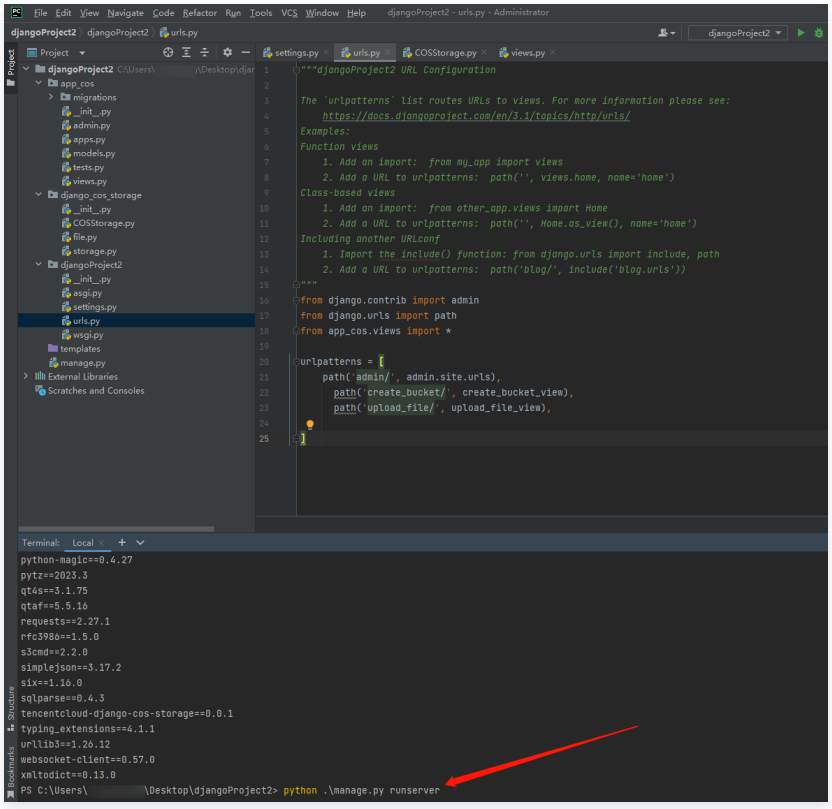
Copy and paste the code:

```
from django.contrib import admin
from django.urls import path
from app_cos.views import *

urlpatterns = [
 path('admin/', admin.site.urls),

    path('upload_file/', upload_file_view),

]
```

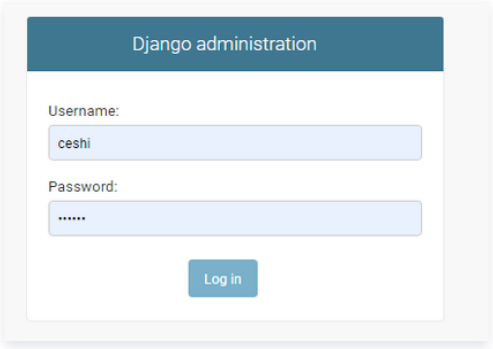
5. In the terminal, enter `python manage.py createsuperuser` and follow the prompts to enter the username and password.




6. Then, enter `python .\manage.py runserver` in the terminal to run.




7. Open the website `http://127.0.0.1:8000/admin/` and enter the account credentials you just set to log in.




**Note**

If the website displays an error message as follows:

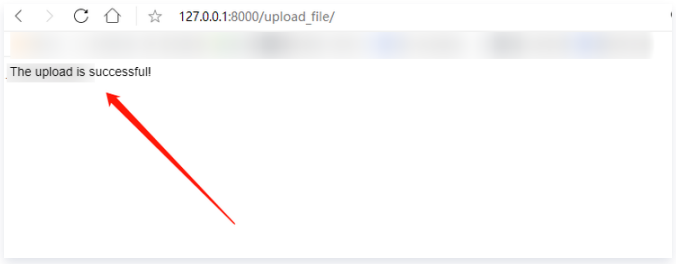

Return to PyCharm, reopen the terminal, and enter the following commands sequentially:

```
python manage.py makemigrations
python manage.py migrate
```

Finally, enter `python .\manage.py runserver` in the terminal to run the server, and then open `http://127.0.0.1:8000/admin/` to access the application.

## Verify Django Attachment Storage in COS

1. Visit `http://127.0.0.1:8000/upload_file` to complete the file upload process. When prompted as shown below, the upload is successful.




2. Log in to the COS console, select the previously created bucket, and you can view the uploaded images under the cessu path.

## Summary

Of course, COS not only offers the above applications and services but also supports various popular open-source applications integrated with Tencent Cloud COS plugins. Click here to launch and start using them immediately!

# COS Cost Solution

Last updated: 2023-09-20 11:54:11

As more enterprises migrate to the cloud, cost concerns regarding cloud usage are increasingly emphasized. The growth of businesses generates massive storage demands, so how can cost optimization be achieved when storing data in the cloud to alleviate the burden on businesses?
Before optimizing costs, it is essential to understand the cost structure of Tencent Cloud Object Storage (COS). The primary billing components for object storage include storage capacity fees, traffic fees, request fees, data retrieval fees, and management feature fees.
For most enterprises, storage capacity fees and traffic fees are the main components of their cloud storage costs. In terms of **storage fees**, Tencent Cloud Object Storage offers various storage types, including Standard Storage, Infrequent Access Storage, Archive Storage, and Deep Archive Storage. Each storage type has different product specifications and prices, allowing enterprises to choose the most cost-effective storage type based on their business model. Regarding **traffic fees**, there are several types of traffic, such as public network outbound traffic, CDN back-to-origin traffic, cross-region replication traffic, and global acceleration traffic. Different business models may result in varying traffic fee structures. For example, if an enterprise primarily operates an e-commerce website with large-scale image distribution, it will have a higher amount of CDN back-to-origin traffic, and thus, its traffic fees will mainly consist of CDN back-to-origin traffic fees.

## Optimization 1: Select the appropriate storage type and business region.

Choosing the appropriate storage type and business region based on the business model can significantly optimize an enterprise's storage costs.
Object storage offers a diverse range of storage types for enterprises, allowing them to choose different storage types based on their performance, data durability, and business availability requirements, while incurring varying costs. Standard Storage has relatively higher storage fees but provides the lowest read latency. Infrequent Access Storage, Archive Storage, and Deep Archive Storage have lower storage capacity fees, but when downloading data, additional data retrieval fees are incurred, and longer retrieval times are required. Therefore, these storage types are more suitable for scenarios with infrequent data access.
The following table illustrates the storage costs for storing 100 TB of business data in the Guangzhou region for one month using different storage types:

| Comparison Item | STANDARD | Infrequent Access Storage | ARCHIVE | DEEP ARCHIVE | MAZ_STANDARD | MAZ_STANDARD_IA |
|---|---|---|---|---|---|---|
| Storage unit price (CNY/GB/month) | 0.118 | 0.08 | 0.033 | 0.01 | 0.15 | 0.1 |
| Traffic unit price (CNY/GB) | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| Request Unit Price (CNY/10k times, taking 1 million times as an example) | 0.01 | 0.05 | 0.01 | Read-write Requests: 0.5 Standard retrieval request: 7 | 0.01 | 0.05 |
| Retrieval price (CNY/GB) | 0 | 0.02 | Standard Retrieval: 0.06 | Standard retrieval: 0.14 | 0 | 0.02 |
| Total cost (100TB storage + no downloads) | 12083.20 | 8192.00 | 3379.20 | 1024 | 15360.00 | 10240.00 |
| Total cost (100TB storage + 100TB download + 1 million requests + 100TB retrieval) | 62084.20 | 60245.00 | 59524.20 | 66110.00 | 65361.00 | 62293.00 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Total cost (100TB storage + 500TB download + 1 million requests + 500TB retrieval) | 262084.20 | 268437.00 | 284100.20 | 323454.00 | 265361.00 | 270485.00 |

**Note**
To learn about the unit prices for other regions and storage types, please refer to the COS Pricing page.

As seen in the table, **if the business data download volume is low, choosing Archive Storage or even Deep Archive Storage can effectively reduce storage costs, with the coldest Deep Archive Storage saving up to 90% of storage fees compared to Standard Storage. However, if the business data requires frequent downloads, the retrieval fees of Infrequent Access Storage, Archive Storage, and Deep Archive Storage will result in additional cost overheads, leading to higher overall expenses.**
In specific business scenarios, we recommend:

1. **Frequent read-write scenarios**: For businesses with more reads than writes, such as UGC scenarios and e-commerce images, Standard Storage can be used. If the business requires high availability and data durability, consider using Multi-AZ Standard Storage.
2. **Low-frequency read scenarios (once a month)**: For businesses such as log data analysis and cloud storage data, where the read frequency is low but high performance is required during reads, Infrequent Access Storage can be used. For businesses with high requirements for availability and data durability, Infrequent Access Storage (multi-AZ) is recommended.
3. **Very infrequent read scenarios (once every three months)**: For businesses such as video surveillance and log data archiving, where the read frequency is extremely low and read performance requirements are minimal, Archive Storage can be used.
4. **Infrequent access scenarios (accessed once every six months)**: For businesses such as medical imaging and archival materials that only require long-term backups and have virtually no demand for read performance, Deep Archive Storage can be used.

Additionally, when selecting different storage types, we recommend that enterprises pay attention to the minimum storage duration and minimum storage unit restrictions for some storage types, as well as the performance differences between them. The following table provides a simple comparison.

| Comparison Item | STANDARD | Infrequent Access Storage | ARCHIVE | DEEP ARCHIVE | MAZ_STANDARD | MAZ_STANDARD_IA |
|---|---|---|---|---|---|---|
| Time to First Byte (TTFB) | Milliseconds | Milliseconds | Minimum 1-minute recovery time. | Restore within a minimum of 12 hours. | Milliseconds | Milliseconds |
| Minimum Storage Unit | No limit | 64KB | 64KB | 64KB | No limit | No limit |
| Minimum Storage Time | No limit | 30 days | 90 days | 180 days | No limit | 30 days |
| Data durability | Eleven nines | Eleven nines | Eleven nines | Eleven nines | Twelve nines | Twelve nines |
| Designing for availability in business operations. | 99.95% | 99.95% | 99.95% | 99.95% | 99.995% | 99.995% |

**Note**
- Minimum storage duration: The shortest time a file must be stored in a specific storage type. If the actual storage time is shorter than the minimum duration, the fee will be calculated based on the minimum duration. For example, if a file in Infrequent Access Storage is stored for only 1 day and then deleted, the fee will still be calculated based on a 30-day storage period, as the minimum storage duration for Infrequent Access Storage is 30 days.
- Minimum Storage Unit: The minimum file capacity required when a file is stored in a specific storage type. If the file capacity does not meet the minimum requirement, it will be calculated based on the minimum file capacity. For example, Infrequent Access Storage requires a minimum of 64KB. If a file in this storage type only occupies 1KB, the fee will still be calculated based on 64KB.

## Optimization 2: Analyze access patterns and implement data tiering.

### 1. Regularly analyze data access patterns using inventory and access log features.

Analyzing data access patterns can provide data support for selecting a reasonable storage type. Tencent Cloud Object Storage (COS) offers inventory and access log features, which record file metadata information and file access records, respectively, and transfer this information to the user's storage bucket.

> **Note**
> - For a detailed introduction to the inventory feature, see Inventory Feature Overview.
> - For a detailed introduction to the access log management feature, please refer to Log Management Overview.

Object Storage offers the COS Select feature, which allows you to search the contents of files. If you have generated a large number of inventory files or log records, you can also purchase an Elastic Map Reduce cluster and set up a Presto cluster for data analysis.

> **Note**
> - For an overview of the COS Select feature, see Select Overview.
> - For information on using EMR for analysis, please refer to Analyzing Data in COS with Presto.

Taking the example of retrieving and analyzing data from an inventory file, once the inventory report is delivered to the specified bucket, you can access the console to analyze the designated report. The analysis operation guide is as follows:

> **Note**
> - For instructions on generating inventory reports, please refer to Enabling Inventory Feature.
> - The console only supports searching for files smaller than 128MB. If the inventory report has a large capacity or a high number of reports, you can opt for using tools, SDKs, or APIs to make the call.

1. Log in to the COS Console.
2. Click the name of the bucket you wish to configure to enter the bucket list page.
3. Locate the corresponding report list, and in the Operation column on the right, select **More > Retrieve**.
4. On the object retrieval page, configure the corresponding input parameters, enter the retrieval statement, and click **Run SQL** to view the retrieval results on the results card page.
   Here are some common retrieval statements for inventory reports:
   - Query the number of files for a specific storage type on a particular day:

```
select count(*) from cosobject s where s._7 = <storage_class>
select count(*) from cosobject s where s._7 = 'Standard'
```

   - Query the storage capacity in MB for a specific storage type on a particular day:

```
select SUM(CAST(s._4 AS FLOAT))/1024/1024 from cosobject s where s._7 = <storage_class>
select SUM(CAST(s._4 AS FLOAT))/1024/1024 from cosobject s where s._7 = 'Standard'
```

   - Query the number of files smaller than 64KB for a specific storage type:

```
select count(*) from cosobject s where s._7 = <storage_class>  and CAST(s._4 AS FLOAT) < <SIZE>
select count(*) from cosobject s where s._7 = 'Standard_IA' and s._4 < 64*1024
```

   - Query the number of files with cross-region replication failures within the bucket:

```
select count(*) from cosobject s where s._9 = 'Failed'
```

> **Note**

The inventory report does not include header information, so you can only search by entering the corresponding field's serial number. The header and serial number correspondence information for the inventory report is as follows:

| Appid | Bucket | Key | Size | LastModifiedDate | ETag | StorageClass | IsMultipartUploaded | Replication Status |
|---|---|---|---|---|---|---|---|---|
| s._1 | s._2 | s._3 | s._4 | s._5 | s._6 | s._7 | s._8 | s._9 |

## 2. Transition data using lifecycle policies and batch processing.

During business development, data access patterns are constantly changing. By periodically analyzing data access patterns using inventory and access log features, you can "cool down" business data based on the analysis report.
For most data, the access frequency tends to decrease as the storage duration increases. Therefore, enterprises need to adjust their data storage types based on the changing access patterns of their business data to achieve optimal cost control.
Object storage offers a lifecycle feature that helps enterprises periodically transition storage types. Enterprises can analyze their business data access patterns using inventory and access logs, and establish reasonable lifecycle transition rules based on these access patterns.
Taking a customer from a community platform as an example, they use object storage services to store user-uploaded image data. Generally, image data is frequently accessed shortly after being uploaded, and after some time, most data gradually "cools down," with access frequency decreasing. Assuming that for this customer, the access frequency of most image data drops below once per month after 90 days and is virtually unaccessed after 365 days, we can compare the cost scenarios when setting a **lifecycle policy** and **not setting a lifecycle policy**:

| Comparison Item | Use Standard Storage only. | Use Standard Storage for hot data, and transition to Infrequent Access Storage after 90 days. | Utilize Standard Storage for hot data, transition to Infrequent Access Storage after 90 days, and move to Archive Storage after 365 days. |
|---|---|---|---|
| Storage unit price (CNY/GB/month, using Guangzhou region as an example) | Standard Storage: 0.118 | Standard Storage: 0.118<br>Infrequent Access Storage: 0.08 | Standard Storage: 0.118<br>Infrequent Access Storage: 0.08<br>Archive Storage: 0.033 |
| Storage Time | Twenty-four months | Twenty-four months<br>(3 months of Standard Storage + 21 months of Infrequent Access Storage) | Twenty-four months<br>(3 months of Standard Storage + 9 months of Infrequent Access Storage + 12 months of Archive Storage) |
| Total Storage Fees (CNY) | 289996.80 | 208281.60 | 150528.00 |

As observed, using lifecycle rules to manage objects in storage buckets can substantially reduce data storage costs. For long-term data storage, **properly configuring lifecycle rules can help businesses reduce storage costs by more than 50%.**
In addition to managing the storage type of business data, the lifecycle feature can also be used to manage file fragments and historical version files in storage buckets. File fragments are incomplete file chunks generated when the transmission of large files fails due to unexpected interruptions, such as network disconnections. If there are numerous file fragments in the business data, you can use lifecycle rules to delete expired fragments. Historical version files are old file information generated after enabling version control. These files can be used for data recovery and rollback in case of accidental deletion but will occupy storage space. Businesses can also set an expiration time for deleting historical version files that are no longer in use, achieving a balance between data security and cost.
Refer to Setting Lifecycle to add rules and enable **Manage Historical Version Files** for downgrading or deleting historical version files. Select **Delete Fragments** and set an expiration time to clean up any potential file fragments.

For specific businesses that require a one-time conversion of a large number of files to colder storage types without fixed rules (such as specified prefixes or tags), users can utilize the COS Batch Processing (Batch) feature. This feature allows for batch replication of data to other storage types. Additionally, you can add object tags to your business data to set lifecycle rules for batch deletion. The operation steps are as follows:
1. Export the list of files pending processing and consolidate them into a CSV format file.

2. Create a COSBatch bulk processing task and import the file list.
3. Initiate a batch processing task and wait for its completion.

For detailed instructions, refer to Batch Processing.

## Optimization 3: Reduce storage capacity by compressing files.

For image data, object storage also provides data compression features to help users reduce image size and storage costs. Currently, the available compression features are as follows:

1. Guetzli Compression: Guetzli compression is a visually lossless compression technique that takes advantage of the human eye's insensitivity to certain color ranges and image details. By selectively discarding detail information without affecting visual quality, it can save approximately 35% – 50% of image size compared to the original image under the same quality conditions.
2. TPG Compression: TPG is a proprietary image format developed by Tencent, which can convert JPG, PNG, GIF, and WEBP images into TPG format, significantly reducing image size with a compression ratio of over 35%.
3. HEIF Compression: For image usage scenarios in iOS environments, images in JPG, PNG, GIF, and WEBP formats can be converted to HEIF format. HEIF format boasts an ultra-high compression ratio, with a compression rate generally above 45%.

Taking a customer's image storage business as an example, the following table compares the cost expenditures for 100 TB of image storage using different compression methods:

| Comparison Item | Guetzli compression | TPG compression | WebP compression | HEIF compression |
|---|---|---|---|---|
| Original storage fees | 12083.20 | 12083.20 | 12083.20 | 12083.20 |
| Compression Ratio Example | 40% | 35% | 20% | 45% |
| Compressed Unit Price | 1 CNY/1,000 times | 0.1 USD/1,000 times | ￥0.025 CNY/GB | 0.1 USD/1,000 times |
| Storage fees after compression | 4833.28 | 4229.12 | 2416.64 | 5437.44 |
| Total cost before compression (100T storage + 500T download) | 262083.20 | 262083.20 | 262083.20 | 262083.20 |
| Total fees after compression | 117233.28 | 94829.12 | 56176.64 | 121637.44 |
| Compression Ratio Range | 30% – 50%, the larger the image, the better the effect. | Over 35%, the compression ratio remains stable. | Approximately 20%. | Above 45% |
| Visual loss | Visually lossless | Almost lossless | Some color gamut loss is noticeable. | Almost lossless |

As seen in the table, **using the image compression feature can significantly reduce storage costs, despite incurring some compression costs. Moreover, a large amount of traffic will be generated during subsequent business operations, and compression can greatly save traffic costs.**

## Conduct a cost review.

Cost optimization should be implemented throughout the entire business process, not just during the initial cloud migration planning. Businesses need to periodically review their costs. On the one hand, as the business grows, storage demands and data access patterns are constantly changing. On the other hand, Tencent Cloud Object Storage is committed to providing users with lower-cost storage services to help reduce costs and increase efficiency. Therefore, conducting regular cost reviews and planning the cloud storage architecture according to business needs is beneficial for reducing storage costs.

In addition, you can also:

1. Go to the Bill Download page in the Billing Center to download your Tencent Cloud billing statement and understand your cloud storage usage details. Analyze your cloud storage consumption and optimize it accordingly.
2. Follow the Tencent Cloud Storage WeChat Official Account or visit the Object Storage Console Overview page to stay informed about new product releases and cost optimization-related updates.