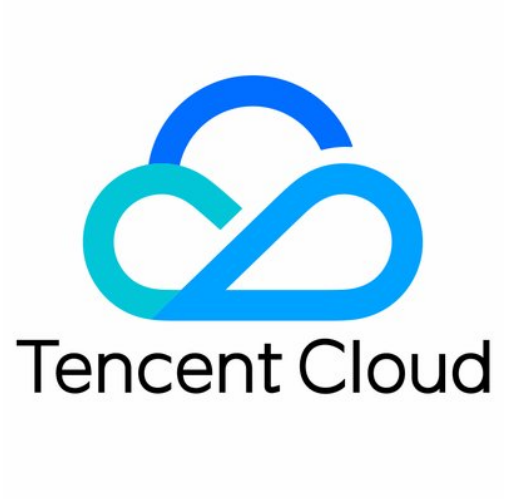
Cloud Object Storage

Best Practices

Product Documentation



Contents

Best Practices

Overview

Last updated：2022-12-19 12:10:40

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

Best Practice	Description
Access control and permission management	Access control and permission management is one of the most practical features of COS. Best practices are outlined in the following documents: • ACL Practices • CAM Practices • Granting Sub-accounts Access to COS • Authorization Cases • Working with COS API Access Policies • Security Guidelines for Using Temporary Credentials for Direct Upload from Frontend to COS • Generating and Using Temporary Keys • Authorizing Sub-Account to Get Buckets by Tag • Descriptions and Use Cases of Condition Keys • Granting Sub-account Under One Root Account Permission to Manipulate Buckets Under Another Root Account
Performance optimization	• COS supports performance expansion to achieve a higher request rate. For detailed directions, see Request Rate and Performance Optimization. • COS can dynamically adjust the part size of a multipart upload to improve the upload success rate for mobile devices under poor network conditions. For more information, see Multipart Upload Resumption in a Weak Network Environment.
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.
Disaster recovery and backup	Best practices and applicable solutions for disaster recovery and backup are outlined in the following three scenarios. • High-Availability Disaster Recovery Architecture Based on Cross-Region Replication • Cloud Data Backup • Local Data Backup

Best Practice	Description
Domain name management	• You can configure an HTTPS custom domain name to access COS. For more information, see Supporting HTTPS for Custom Endpoints. • You can configure CORS rules in COS. For more information, see Setting CORS. • You can host static websites in COS. For more information, see Hosting Static Website. • You can build frontend single-page application in COS. For more information, see Building a Frontend Single-Page Application with COS's Static Website Feature.
Direct data upload	Below are the best practices of direct data upload: • Practice of Direct Transfer for Web End • Practice of Direct Upload Through WeChat Mini Program • Practice of Direct Upload for Mobile Apps • uni-app Direct Upload Practice
Data security	• This practice describes the data security solution in terms of pre-event prevention, mid-event monitoring, and post-event backtracking. For more information, see Introduction to COS Data Security Solution. • You can configure hotlink protection in COS to control access sources. For more information, see Hotlink Protection Practice.
Data verification	• You can ensure the integrity of data uploaded to COS by using an MD5 checksum. For more information, see MD5 Verification. • You can verify data by using a CRC-64 checksum. For more information, see CRC64 Check.
Big data	• You can use COS as the deep storage of Druid. For more information, see Using COS as Deep Storage of Druid. • You can use Terraform to manage COS. • You can use DataX to import or export data to or from COS. For more information, see Importing/Exporting COS Using DataX. • You can configure COSN by using CDH. For more information, see Configuring COSN for CDH. • You can use COS Ranger to control permissions. For more information, see COS Ranger Permission System Solution. • You can connect Oceanus to COS. For more information, see Connecting Oceanus to COS.
Using COS in third-party applications	• You can store data of S3-compatible third-party applications to COS by using COS general settings. For more information, see Use the general configuration of COS in third-party applications compatible with S3. • You can use the remote attachment feature to store forum attachments in COS. • You can store multimedia content in COS by using the WordPress plugin. For more information, see Storing Remote WordPress Attachments to COS.

Best Practice	Description
Using APIs to zip files	You can package multiple files through an API.

Access Control and Permission Management
ACL Practices

Last updated：2021-09-27 14:14:57

ACL Overview

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

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

- Only supports permission grants to Tencent Cloud accounts.
- Only supports five operation sets: Read Objects, Write to Objects, Read ACLs, Write to ACLs, and All Permissions.
- Does not support additional conditions for the ACL rules to take effect.
- Does not support explicit deny.

Control granularities supported by ACLs

- Bucket
- Object Key Prefix
- Object

Control Elements of ACLs

When a bucket or object is created, the root account to which the bucket or object's resources belong has full access to these resources, and the access cannot be modified or deleted. You can use ACLs to grant other Tencent Cloud accounts the access permissions.
The following is an ACL example for a bucket. 100000000001 is the root account, 100000000011 is its sub-account, and 100000000002 is another root account. The ACL contains an Owner element that identifies the bucket owner, who has full access to the bucket. The Grant element grants anonymous read permission, which is expressed as READ permission of `http://cam.qcloud.com/groups/global/AllUsers` .

```
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Owner>
```

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

Grantees

Root account

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

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

Sub-account

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

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

Anonymous user

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

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

Operation Set

The operation sets supported by ACLs are listed below.

Operation Set	Access to Bucket	Access to Prefix	Access to Object

Operation Set	Access to Bucket	Access to Prefix	Access to Object
READ	Lists and reads objects in the bucket	Lists and reads objects in the directory	Reads objects
WRITE	Creates, overwrites and deletes any object in the bucket	Creates, overwrites and deletes any object in the directory	Not supported
READ_ACP	Reads the ACL of bucket	Reads the ACL in the directory	Reads the ACL of object
WRITE_ACP	Modifies the ACL of bucket	Modifies the ACL in the directory	Modifies the ACL of object
FULL_CONTROL	Any operations on the bucket and objects	Any operations on the objects in the directory	Any operations on the objects

Standard ACL

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

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

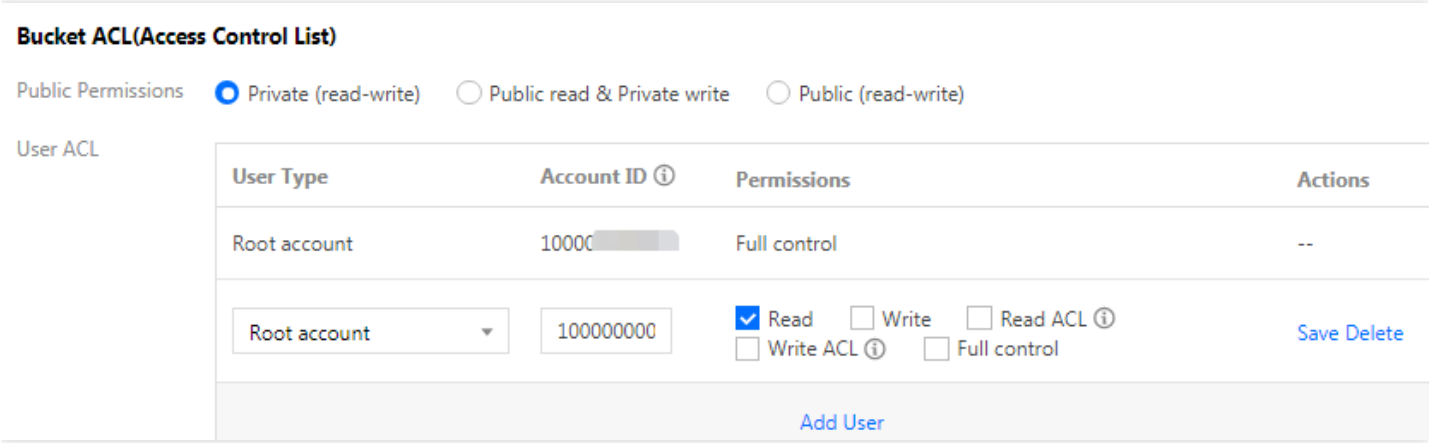
Standard ACL	Description
(Null)	This is the default policy. Others do not have permissions. The permissions for resources are inherited from upper level.
private	Other users do not have permissions
public-read	Anonymous user group has the READ permission
public-read-write	Anonymous user group has the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

Directions

Using ACLs in the console

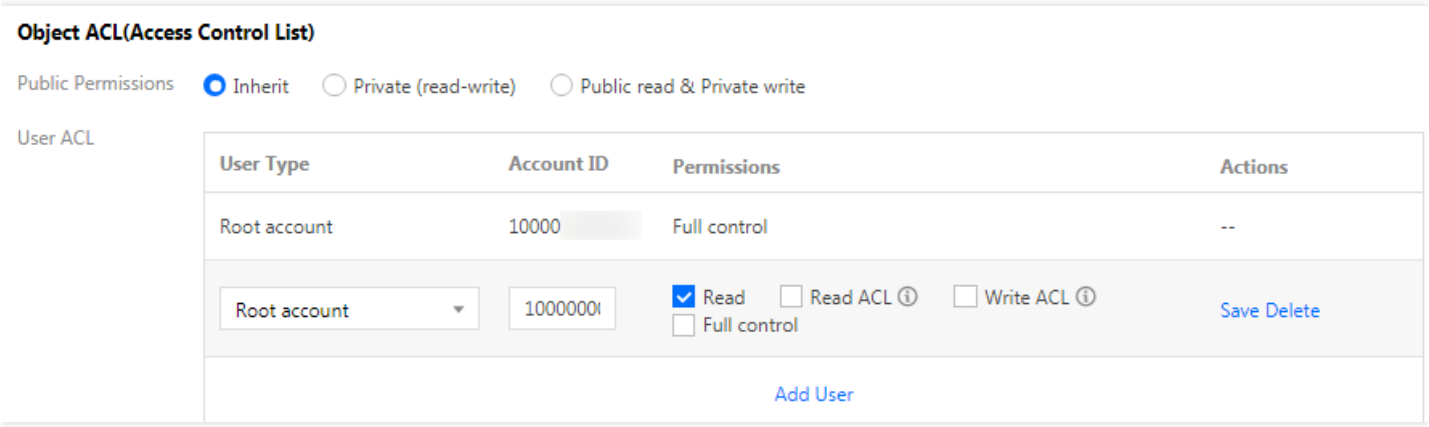
Set an ACL for a bucket

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



Set an ACL for an object

The following example grants another root account the read access to an **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 ACLs via APIs

Bucket ACL

API	Operation	Description

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

CAM Practices

Last updated：2022-10-26 14:52:02

Overview

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

Features

Granting access to resources under the root account

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

Granular permission management

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

Eventual consistency

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

Use Cases

Enterprise sub-account permission management

Enterprises may need to grant the minimal access permission of their cloud resources to all kinds of employees. Assume that an enterprise owns many cloud resources (CVM/VPC/CDN instances, COS buckets/objects, etc.) and many employees (developers, testers, Ops engineers, etc.).
Developers and testers need the read/write permissions of project-specific cloud resources on development servers

and test servers respectively, while Ops engineers are responsible for the purchase and daily operations of such servers. If the duty or project of an employee changes, the corresponding permissions should be terminated.

Cross-enterprise permission management

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

Policy Syntax

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

Note：

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

Core elements

Core Element	Description	Required
version	Specifies the version of the policy syntax. Valid value: `2.0` .	Yes
principal	Describes the entity to be authorized by the policy, including users (developers, sub-accounts, anonymous users) and user groups	This element can be used only in policy management APIs and policy syntax parameters.
statement	Describes the details of a permission or a permission set defined by other elements including effect, action, resource, and condition. One policy has only one statement.	Yes

Core Element	Description	Required
action	Describes the action to be allowed or denied, which can be an API operation or a set of API operations. This element is case-sensitive, such as `name/cos:GetService` .	Yes
resource	Describes the resource to which the permission applies. A resource is described in a six-segment format. Detailed resource definitions vary by product. For more information on how to specify a resource, see the documentation of the product for which you are writing a resource statement.	Yes
condition	Describes the condition for the policy to take effect. A condition consists of the operator, action key, and action value. A condition value may be time, IP address, etc. Some services allow you to specify additional values in a condition.	No
effect	Describes whether the statement result is "allow" or "deny".	Yes

Policy limits

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

Sample policy

The policy in this example allows the sub-account with ID 100000000011 under the root account with ID 100000000001 (APPID: 1250000000) to **upload** and **download** objects regarding the `examplebucket-bj`

bucket in Beijing region and the `exampleobject` object in the `examplebucket-gz` bucket in Guangzhou region, on condition that the access IP falls within the IP range `10.*.*.10/24` .

```
{
"version": "2.0",
"principal": {
"qcs": ["qcs::cam::uin/100000000001:uin/100000000011"]
},
"statement": [{
"effect": "allow",
"action": ["name/cos:PutObject", "name/cos:GetObject"],
"resource": ["qcs::cos:ap-beijing:uid/1250000000:examplebucket-bj-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：2022-12-19 12:12:58

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, see CAM Glossary.

Root account

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

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

Sub-account (user) and user group

- A sub-account is an entity created by the root account. It has an ID and identity credentials, as well as permission to log in to the Tencent Cloud console.
- By default, a sub-account does not own resources. It needs to be authorized by a root account.
- One root account can create multiple sub-accounts (users).
- One sub-account can belong to multiple root accounts to assist them in managing their Tencent Cloud resources. However, at any specific time point, one sub-account can only log in under one root account to manage its Tencent Cloud resources.
- A sub-account can switch between developers (root accounts) in the console.
- When a sub-account logs in to the console, it is under its default root account and has the access permissions granted by the root account.
- After a sub-account switches from one root account to another, it will only have the access permissions granted by the current root account, but not the permissions granted by the previous one.
- A user group is a collection of multiple users (sub-accounts) with the same functions. You can create different user groups based on your business needs and associate them with appropriate policies to grant them different permissions.

Directions

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

Step 1. Create a sub-account

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

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

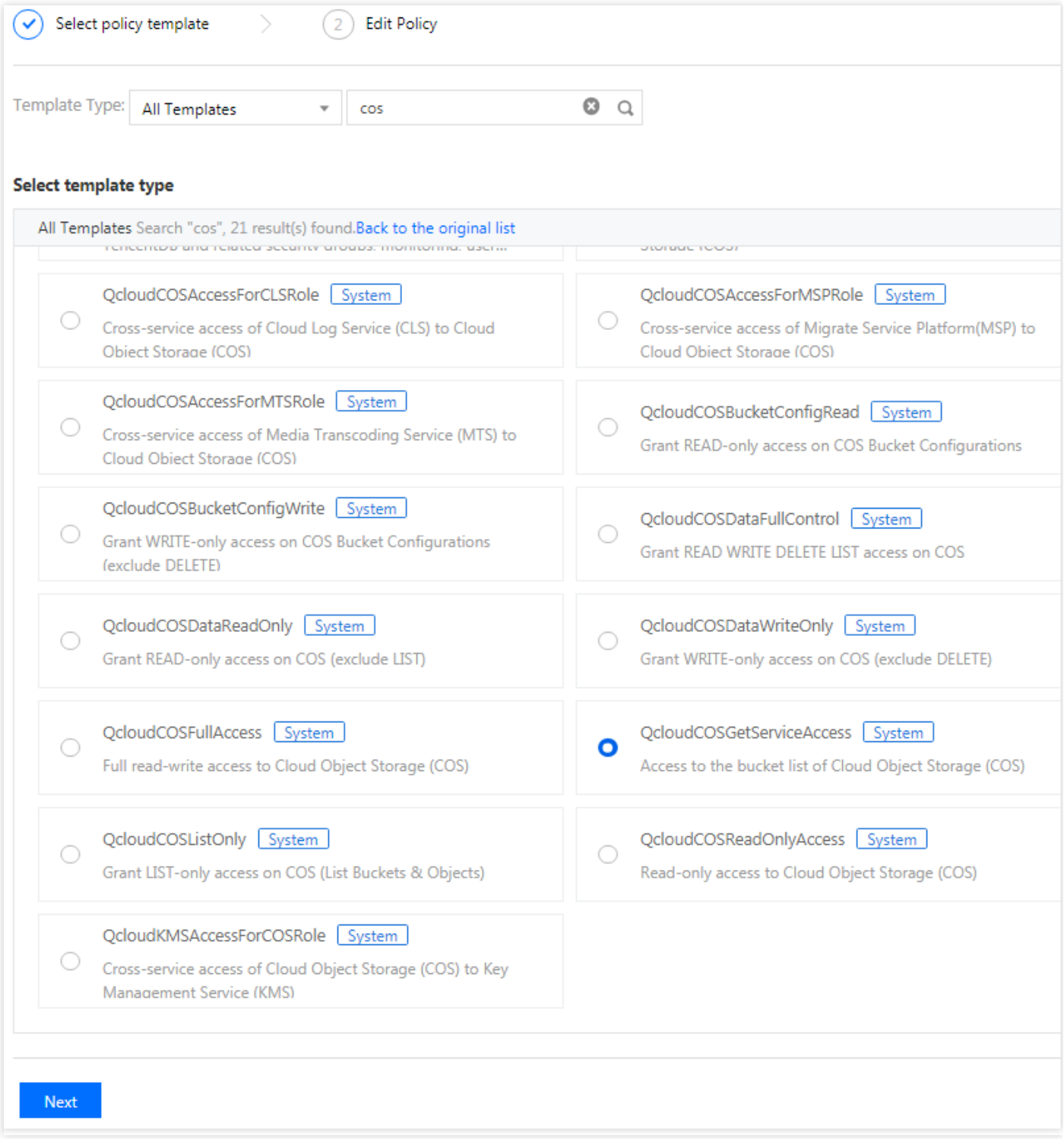
- **User Information**: Set the username (for example, Sub_user) and email address of the sub-account. The email address is needed to receive the email sent by Tencent Cloud to bind the sub-account with his/her WeChat account.
- **Access Method**: Select **Programming access** and **Tencent Cloud console access**. Other configuration items can be set as needed.

4. Click **Next** and start identity verification.
5. 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.
6. Click **Complete**. In this way, the sub-account is created.

Step 2. Grant permissions to the sub-account

You can grant permissions to the sub-account by configuring policies for the sub-account (user) or user group in CAM.

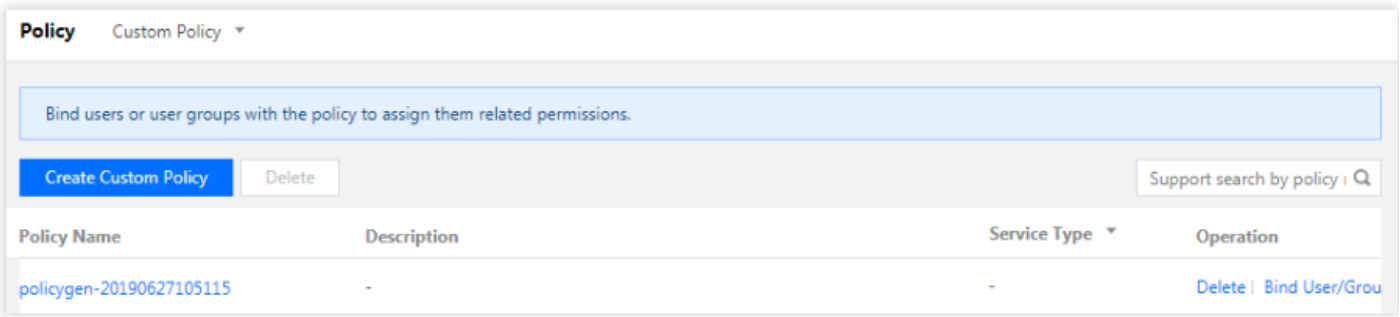
1. Log in to the CAM console.
2. Choose **Policy** > **Create Custom Policy** > **Create by Policy Syntax** to go to the policy creation page.
3. You can select **Blank Template** to customize a permission policy as needed. You can also select a COS-associated **System Template**. Then, click **Next**.



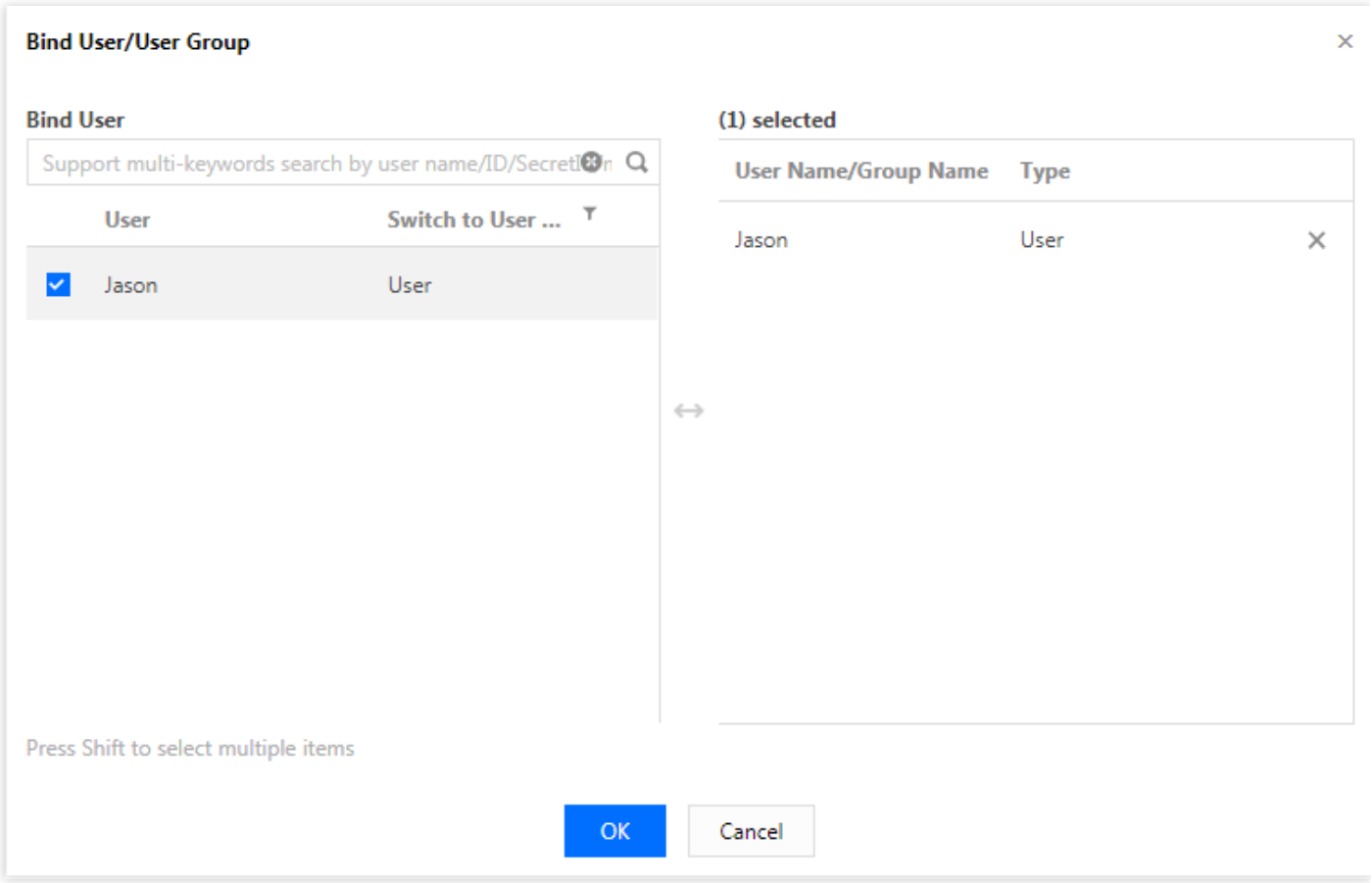
4. Enter an easy-to-remember policy name. If you have selected **Blank Template**, enter the policy syntax. For more information, see [Sample Policies](#Sample policies). You can copy and paste the policy content into the **Policy**

Content box and click **Complete**.

5. After you create a policy, associate it with the sub-account.



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



Step 3. Access COS resources using the sub-account

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

(1) Programming access

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

1. Log in to the CAM console with the root account.
2. Click **User** > **User List**.
3. Click the name of the sub-account to go to the **User Details** page of the sub-account.
4. Click the **API Key** tab. Then, click **Create Key** to create `SecretId` and `SecretKey` for the sub-account.

After this, you can use this sub-account's `SecretId` and `SecretKey`, as well as the root account's `APPID` to access COS resources.

Note：

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

Sample of access using XML Java SDK

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

```
// 1. Initialize identity information
COSCredentials cred = new BasicCOSCredentials("<root account's APPID>", "<sub-account's SecretId>", "<sub-account's SecretKey>");
```

Sample:

```
// 1. Initialize identity information
COSCredentials cred = new BasicCOSCredentials("1250000000", "AKIDasdfmRxHPa9oLhJp****", "e8Sdeasdfas2238Vi****");
```

Sample of access using COSCMD command line tool

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

```
coscmd config -u <root account's APPID> -a <sub-account's SecretId> -s <sub-account's SecretKey> -b <root account's bucketname> -r <root account's bucket region>
```

Sample:

```
coscmd config -u 1250000000 -a AKIDasdfmRxHPa9oLhJp**** -s e8Sdeasdfas2238Vi**** -b examplebucket -r ap-beijing
```

(2) Tencent Cloud console access

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

Sample Policies

The following typical sample policies are provided herein. When configuring a policy, you can refer to the following code, copy and paste it into the **Policy Content** box, and modify it as needed. For more policy syntax for other

common COS scenarios, see Overview or the business use cases parts of the CAM 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. Configure it with caution.

The policy is as follows:

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

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

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

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:List*",
"name/cos:Get*",
"name/cos:Head*",
"name/cos:OptionsObject"
],
"resource": "*",
"effect": "allow"
```

```
},
{
"effect": "allow",
"action": "monitor:*",
"resource": "*"
}
]
}
```

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

The policy is as follows:

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

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

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

To enter more conditions, see Conditions.

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

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

Authorization Cases

Last updated：2022-08-11 15:04:53

Granting Permission via Bucket Policy

Preparations

1. Create a bucket.
   Granting permissions through a bucket policy involves specific buckets, so you need to create a bucket first. For authorization at the account level, see Granting Permission via CAM in this document.
2. Prepare the UIN of the account to be authorized.
   This document assumes that the root account that owns the target bucket has a UIN of 100000000001 and its sub-account has a UIN of 100000000011. The sub-account needs to be authorized to access the destination bucket.

   Note：
   - To query sub-accounts created under the root account, log in to the CAM console and view them in the User List.
   - To create a sub-account, see Creating Sub-user.

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

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

Authorization cases

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

The configuration information is as follows:

Configuration Item	Description
Effect	Select **Allow**.

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

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

The configuration information is as follows:

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

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

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

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

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

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

Configuration Item	Description
Effect	Select **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`.
Resource	Specify the object key to be denied access, such as `folder/sub-folder/privateobject`.
Actions	Select **All Actions**.

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

The configuration information is as follows:

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

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：2022-09-19 17:30:44

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

Overview

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

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

```
{
"version": "2.0",
"statement": [{
"action": [
// Upload an object by using simple upload
"name/cos:PutObject",
// Upload an object by using an HTML form
"name/cos:PostObject",
// Initialize a multipart upload
"name/cos:InitiateMultipartUpload",
// List all ongoing multipart uploads
"name/cos:ListMultipartUploads",
// List uploaded parts
"name/cos:ListParts",
// Upload parts
"name/cos:UploadPart",
// Complete a multipart upload
"name/cos:CompleteMultipartUpload",
// Abort a multipart upload
"name/cos:AbortMultipartUpload"
```

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

Authorization Policy Elements

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

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

Service APIs

Querying bucket list

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

Sample

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

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

Bucket APIs

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

- To allow access to buckets in all regions
  The `resource` field should be set to `*`. **Use this option with caution as it may present data security risks due to excessive permissions.**
- To allow access only to buckets in a specified region
  For example, to grant access to `examplebucket-1250000000` under the APPID `1250000000` in the region `ap-beijing`, the `resource` field should be set to `qcs::cos:ap-beijing:uid/1250000000:*`.
- To allow access only to a bucket with a specified name in a specified region
- For example, to grant access to the bucket named `examplebucket-1250000000` under the APPID `1250000000` in the region `ap-beijing`, the `resource` field should be set to `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*`.

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

Creating bucket

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

Sample

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

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

Note：
For bucket naming rules, see Bucket Overview.

Extracting bucket and its permissions

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

Sample

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

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:HeadBucket"
```

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

Querying object list

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

Sample

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

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

Deleting bucket

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

Sample

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

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

Setting bucket ACL

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

Sample

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

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

Querying bucket ACL

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

Sample

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

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

Setting CORS configuration

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

Sample

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

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

Querying CORS configuration

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

Sample

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

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

Deleting CORS configuration

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

Sample

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

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

```
}
]
}
```

Setting lifecycle configuration

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

Sample

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

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

Querying lifecycle configuration

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

Sample

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

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetBucketLifecycle"
```

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

Deleting lifecycle configuration

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

Sample

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

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

Object APIs

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

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

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

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

Uploading object by using simple upload

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

Sample

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

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

Multipart upload

Multipart upload APIs include `Initiate Multipart Upload`, `List Multipart Uploads`, `List Parts`, `Upload Part`, `Complete Multipart Upload`, and `Abort Multipart Upload`. To grant access to these APIs, the `action` field in the policy should be a collection of `"name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListPa`

`rts","name/cos:UploadPart","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload"` .

Sample

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

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

Querying multipart upload

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

Sample

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

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:ListMultipartUploads"
],
```

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

Uploading object by using HTML form

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

Sample

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

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

Appending parts

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

Sample

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

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

Querying object metadata

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

Sample

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

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

Downloading object

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

Sample

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

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

Copying object

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

Sample

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

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

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

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

Copying part

To grant access to the `Upload Part - Copy` API, the `action` field for the destination object should be a collection of `"name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListParts","name/cos:PutObject","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload"`, and the `action` field for the source object should be set to `name/cos:GetObject`.

Sample

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

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

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

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

Setting object ACL

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

Sample

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

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

Querying object ACL

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

Sample

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

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

Checking CORS configuration

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

Sample

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

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

Restoring archived object

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

Sample

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

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

Deleting object

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

Sample

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

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

```
]
}
```

Deleting multiple objects

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

Sample

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

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

Authorization Policies for Common Scenarios

Granting full access to all resources

The following policy grants full access to all resources:

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

```
"resource": [
"*"
]
}
]
}
```

Granting read-only access to all resources

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

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

Granting read-write access to resources with specified path prefix

The following policy grants the permission to access only files with the path prefix `doc` in the bucket `examplebucket-1250000000` and does not allow any operations on files in other paths:

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

```
}
]
}
```

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

Last updated：2022-05-31 15:36:11

Overview

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

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

Prerequisites

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

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

Bad Examples and Security Regulations

Bad example 1. Excessive `resource`

Application A uses COS when registered users upload profile photos. The profile photo of each user has a fixed object key `app/avatar/<username>.jpg` and object keys `app/avatar/<username>_m.jpg` and `app/avatar/<username>_s.jpg` for different photo sizes. When you generate a temporary key on the backend, you specify `resource` as `qcs::cos:<region>:uid/<appid>:<bucketname-appid>/app/avatar/*` for convenience. In this case, when a malicious user gets the generated temporary key through methods such as packet capture, they can upload an image to overwrite any user's profile photo and thus gain unauthorized access, and the user's valid profile photo will be overwritten and lost.

Security regulation

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

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

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

Bad example 2. Excessive `action`

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

Security regulation

`action` indicates operations allowed for the temporary key. In principle, a temporary key with `name/cos:*` that allows all operations should not be distributed to the frontend; instead, all needed operations must be explicitly listed. If

each operation needs different resource paths, you should match the **operation ** and **resource** path separately rather than specifying them in batches.

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

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

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

Security regulation

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

In this example, you should use the following code:

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

Bad example 4. Unauthorized access to temporary key

Application D provides a forum service, and attachments to posts in the forum are stored in COS. The forum application has different user levels, and only active users with a certain number of posts can view posts and attachments in certain subforums. For some public subforums, all users can view the posts and attachments.
The temporary key generation API on the COS backend will generate a temporary key that allows downloading attachments in the corresponding subforum based on the subforum ID passed in by the frontend. However, in the implementation, the backend does not check whether the requesting user has access to the subforum of the specified ID; therefore, anyone can request this API to get a temporary key that allows access to attachments in private subforums and thus gain unauthorized access. The limitation on access to private resources does not take effect, leading to potential data leakage.

Security regulation

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

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

Summary

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

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

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

Generating and Using Temporary Keys

Last updated：2022-02-26 09:45:52

Note：

- When authorizing access with a temporary key, ensure that you follow the principle of the least privilege as needed. If you grant excessive permissions, such as granting permissions to all resources `(resource: *)` or all operations `(action: *)`, data security risks may arise.
- If you have specified a permission scope for the temporary key when applying for it, you can only use the key within the scope. For example, if you have limited the permissions to only uploading objects to the `examplebucket-1-1250000000`, you **can't** upload objects to the `examplebucket-2-1250000000` bucket or download objects from the `examplebucket-1-1250000000` bucket.

Temporary Key

The temporary key (access credential) obtains limited permissions using Tencent Cloud CAM APIs.
To call the COS API, you use temporary keys to calculate a signature for identity authentication.
When a COS API request uses a temporary key to calculate the signature for authentication, the following three fields in the message returned by the temporary key API are required:

- TmpSecretId
- TmpSecretKey
- Token

Benefits to Use a Temporary Key

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

For COS API authorization policies, see:

- Working with COS API Access Policies
- Examples of Temporary Key Authorization Policies in Common Scenarios.

Getting a Temporary Key

You can get a temporary key via COS STS SDK, or by calling the STS API directly.

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

COS STS SDK

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

Language	Download Address	Sample
Java	Download	View Sample
.NET	Download	View Sample
Go	Download	View Sample
Node.js	Download	View Sample
PHP	Download	View Sample
Python	Download	View Sample

> Note：
> To avoid the differences between versions of the STS API, STS SDKs take a return parameters structure that may be different from that of the STS API. For more information, see Java SDK documentation.

Here is an example to obtain a temporary key using the downloaded Java SDK:

Sample codes

```
// Import `java sts sdk` using the integration method with Maven as described on GitHub, with v3.1.0 or later required.
public class Demo {
```

```
public static void main(String[] args) {
TreeMap config = new TreeMap();
try {
// `SecretId` and `SecretKey` represent permanent identities (root account, sub-account) for applying for a temporary key. If it is a sub-account, it must have permission to operate buckets.
// Replace it with your Cloud API key SecretId
config.put("secretId", "SecretId");
// Replace it with your Cloud API key SecretKey
config.put("secretKey", "SecretKey");
// Set a domain:
// If you use Tencent Cloud CVMs, you can set an internal domain.
//config.put("host", "sts.internal.tencentcloudapi.com");
// Validity period of the key, in seconds (default: 1800). The value can be up to 7200 (2 hours) for the root account, and 129600 (36 hours) for a sub-account.
config.put("durationSeconds", 1800);
// Replace it with your own bucket
config.put("bucket", "examplebucket-1250000000");
// Replace it with the region where your bucket resides
config.put("region", "ap-guangzhou");
// Change it to an allowed path prefix. You can determine the upload path based on your login status.
// Examples of several typical prefix authorization scenarios:
// 1. Allow access to all objects: "*"
// 2. Allow access to specified objects: "a/a1.txt", "b/b1.txt"
// 3. Allow access to objects with specified prefixes: "a*", "a/*", "b/*"
// If "*" is entered, you allow the user to access all resources. Unless otherwise necessary, grant the user only the limited permissions that are needed following the principle of least privilege.
config.put("allowPrefixes", new String[] {
"exampleobject",
"exampleobject2"
});
// A list of permissions needed for the key (required)
// The following permissions are required for simple upload, upload using a form, and multipart upload. For other permissions, visit https://intl.cloud.tencent.com.cn/document/product/436/30580
String[] allowActions = new String[] {
// Simple upload
"name/cos:PutObject",
// Upload using a form or WeChat Mini Program
"name/cos:PostObject",
// Multipart upload
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
```

```
"name/cos:CompleteMultipartUpload"
};
config.put("allowActions", allowActions);
Response response = CosStsClient.getCredential(config);
System.out.println(response.credentials.tmpSecretId);
System.out.println(response.credentials.tmpSecretKey);
System.out.println(response.credentials.sessionToken);
} catch (Exception e) {
e.printStackTrace();
throw new IllegalArgumentException("no valid secret !");
}
}
}
```

FAQs

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

Use 3.1.0 or a later version.

Accessing COS using a temporary key

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

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

> Note：
> Before you run the sample, go to the GitHub project to download the Java SDK installation package.

```
// Import `cos xml java sdk` using the integration method with Maven as described 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 {
// Basic user information
String tmpSecretId = "COS_SECRETID"; // Replace it with the temporary SecretId returned by the STS API.
String tmpSecretKey = "COS_SECRETKEY"; // Replace it with the temporary SecretKey returned by the STS API.
```

```
String sessionToken = "Token"; // Replace it with the temporary token returned by
the STS API.
// 1. Initialize user authentication information (secretId, secretKey).
COSCredentials cred = new BasicCOSCredentials(tmpSecretId, tmpSecretKey);
// 2. Set the bucket region. For COS regions, see https://cloud.tencent.com/docum
ent/product/436/6224.
ClientConfig clientConfig = new ClientConfig(new Region("ap-guangzhou"));
// 3. Generate a COS client.
COSClient cosclient = new COSClient(cred, clientConfig);
// The bucket name must contain `appid`.
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
// Upload an object. You are advised to call this API to upload objects smaller t
han 20 MB.
File localFile = new File("src/test/resources/text.txt");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFi
le);
// Set the `x-cos-security-token` header field.
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setSecurityToken(sessionToken);
putObjectRequest.setMetadata(objectMetadata);
try {
PutObjectResult putObjectResult = cosclient.putObject(putObjectRequest);
// Success: PutObjectResult returns the file ETag.
String etag = putObjectResult.getETag();
} catch (CosServiceException e) {
//Failure: CosServiceException is thrown.
e.printStackTrace();
} catch (CosClientException e) {
//Failure: CosClientException is thrown.
e.printStackTrace();
}
// Shut down the client.
cosclient.shutdown();
}
}
```

Authorizing Sub-Account to Get Buckets by Tag

Last updated：2022-12-29 10:43:01

Overview

COS allows you to filter buckets by tag in the console, which is powered by the Tencent Cloud Tag service.

Suppose the enterprise account `CompanyExample` (OwnerUin: 100000000001, Owner_appid: 1250000000) has a sub-account `Developer`, and `CompanyExample` needs to grant the sub-account permissions to get the list of tagged objects. The following describes how to grant the permissions.

Notes

If you want to allow the sub-account to get the list of buckets filtered by tag in the console, you need to grant it the required permissions `GetResourceTags`, `GetResourcesByTags`, and `GetTags` by creating custom policies in the CAM console.

Directions

1. Log in to the CAM console as the enterprise account `CompanyExample` and enter the policy configuration page.

2. Grant the sub-account `Developer` permissions to pull the list of tagged objects through the **policy generator** or **policy syntax**.

   - Policy generator
   - Policy syntax

   i. Go to the Policies page in the CAM console.
   ii. Click **Create Custom Policy** > **Create by Policy Generator**.
   iii. On the permission configuration page, configure the following:

   - **Tag**:
     - **Effect**: Use the default option **Allow**.
     - **Service**: Select **Tag**.
     - **Action**: Select actions based on your business needs. Here, select `GetResourceTags`, `GetResourcesByTags`, and `GetTags`.

- **Resource**: Select **All resources**.
- **Add Permissions**: Configure based on your business needs.

v. Click **Next**, enter the policy name, and associate users/user groups.

vi. Click **Complete**.

3. Associate the policy with the sub-account `Developer` by locating the policy created in step 2 on the **Policies** page and clicking **Associate Users/Groups** on the right.

4. In the **Associate Users/Groups** window, select the sub-account `Developer` and click **OK**.

5. Log in to the console with the sub-account `Developer` . On the Bucket List page, select **Tag** and enter a **tag key** to search for buckets with the specified tag.

Descriptions and Use Cases of Condition Keys

Last updated：2022-10-09 16:40:52

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 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 in Conditions.

Note：

- When using condition keys to write a bucket policy, follow the principle of least privilege, add the corresponding condition keys only to applicable requests (actions), and avoid using the * wildcard when specifying the actions. Using the wildcard will cause the requests to fail. For more information on condition keys, see Conditions.
- When you create a policy in the CAM console, pay attention to the syntax format. The syntax elements of `version`, `principal`, `statement`, `effect`, `action`, `resource`, and `condition` must all begin with a letter in the same letter case.

Use Cases of Condition Keys

Restricting user access IPs (qcs:ip)

Condition key `qcs:ip`

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

Sample: Allowing access by users from specified IPs only

The policy in this example allows the sub-account with ID 100000000002 under the root account with ID 100000000001 (APPID: 1250000000) to upload and download objects regarding the `examplebucket-bj` bucket in Beijing region and the `exampleobject` object in the `examplebucket-gz` bucket in Guangzhou region, on condition that the access IP falls within the IP range `10.*.*.10/24`.

```
{
"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": "10.*.*.10/24"
}
}
}]
}
```

Restricting `vpcid` (vpc:requester_vpc)

Condition key `vpc:requester_vpc`

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

Sample: Restricting the `vpcid` to `aqp5jrc1`

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

```
{
"statement": [
{
"action": [
"name/cos:*"
],
"condition": {
"string_equal": {
"vpc:requester_vpc": [
"vpc-aqp5jrc1"
]
}
},
"effect": "allow",
"principal": {
"qcs": [
"qcs::cam::uin/100000000001:uin/100000000002"
```

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

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

Request parameter `versionid`

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

- If `versionid` is not carried, requests will apply to the latest version of the object by default.
- If `versionid` is an empty string, this is equivalent to the case where `versionId` is not carried.
- If `versionid` is `"null"` , for objects that are uploaded before versioning is enabled for a bucket, their version numbers will become the `"null"` string after versioning is enabled.

Condition key `cos:versionid`

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

Sample 1: Allowing users to get objects of a specified version

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

According to the policy, object download requests sent by the sub-account with UIN 100000000002 can succeed only when they carry the `versionid` parameter and the value of `versionid` is the version number `MTg0NDUxNTc1NjIzMTQ1MDAwODg` .

```
{
"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-account permissions, and the sub-account obtains the same permissions without any conditions attached through other means, the broader authorization policy will take effect. For example, if a sub-account is in a user group and the root account grants the `GetObject` permission to the user group without any conditions attached, then the restriction on the version number in the above policy will not take effect.

To cope with this, you can add an explicit deny policy based on the above policy to achieve tighter permission control. The following deny policy specifies that, when a sub-account initiates an object download request that does not carry the `versionid` parameter or carries `versionid` with a value other than `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"
}
```

Sample 2: Allowing users to get objects of the latest version only

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

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

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

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

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

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

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

Sample 3: Forbidding users to delete objects uploaded before versioning is enabled

Your bucket may have some objects uploaded before versioning is enabled, and the version numbers of these objects will become "null" after versioning is enabled. Sometimes, you may want to enable additional protection for these objects, for example, to prevent users from permanently deleting these objects. This involves denying the 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 `versionid` request parameter with the value `"null"`, the request will be denied.

Therefore, if object A is uploaded to the `examplebucket-1250000000` bucket before versioning is enabled, the version number of object A will become 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-account 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 either 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 uploaded files (cos:content-length)

Request header `Content-Length`

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

Condition key `cos:content-length`

When uploading an object, you can use the `cos:content-length` condition key to restrict the `Content-Length` request header to limit the file size of the uploaded object. In this way, you can flexibly manage storage space and avoid wasting storage space and network bandwidth by uploading files that are too large or too small.

In the two examples below, the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the `cos:content-length` condition key to restrict the value of the `Content-Length` header in upload requests initiated by the sub-account with UIN 100000000002.

Sample 1: Restricting the maximum value of the `Content-Length` request header

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

```
{
"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
}
}
}
]
}
```

Sample 2: Restricting the minimum value of the `Content-Length` request header

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

```
{
"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 uploaded files (cos:content-type)

Request header `Content-Type`

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

Condition key `cos:content-type`

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

Sample 1: Restricting the `Content-Type` of `PutObject` requests to "image/jpeg"

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

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

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

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

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

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

Request parameter `response-content-type`

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

Condition key `cos:response-content-type`

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

Sample 1: Restricting the `response-content-type` of `GetObject` requests to "image/jpeg"

Assume that the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the following bucket policy to require that `GetObject` requests initiated by the sub-account with UIN 100000000002 carry the `response-content-type` request parameter with the value `image/jpeg`.

`response-content-type` is a request parameter and needs to be URL-encoded when the request is initiated (encoded value: `response-content-type=image%2Fjpeg`). Therefore, when you set the policy, "image/jpeg" also needs to be URL-encoded, that is, `image%2Fjpeg` needs to be entered.

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

```
]
}
```

Allowing HTTPS requests only (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.

Sample 1: Requiring 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"
}
}
}
]
}
```

Sample 2: Denying all non-HTTPS requests

Assume that the root account with UIN 100000000001 that owns the `examplebucket-1250000000` bucket uses the following bucket policy to deny all 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 specified storage classes only (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 storage class fields include `STANDARD`, `STANDARD_IA`, `INTELLIGENT_TIERING`, `ARCHIVE`, and `DEEP_ARCHIVE`.

Sample 1: Requiring `PutObject` requests to set the storage class to `STANDARD`

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

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

Allowing setting specified bucket/object ACLs only (cos:x-cos-acl)

Request header `x-cos-acl`

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

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

Condition key `cos:x-cos-acl`

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

Sample 1: Requiring `PutObject` requests to set the object ACL to `private`

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

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

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

Allowing listing objects in specified directories only (cos:prefix)

Condition key `cos:prefix`

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

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

Sample 1: Allowing listing objects in a specified directory of the bucket only

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

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

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

Granting Sub-account Under One Root Account Permission to Manipulate Buckets Under Another Root Account

Last updated：2022-06-20 14:23:24

Overview

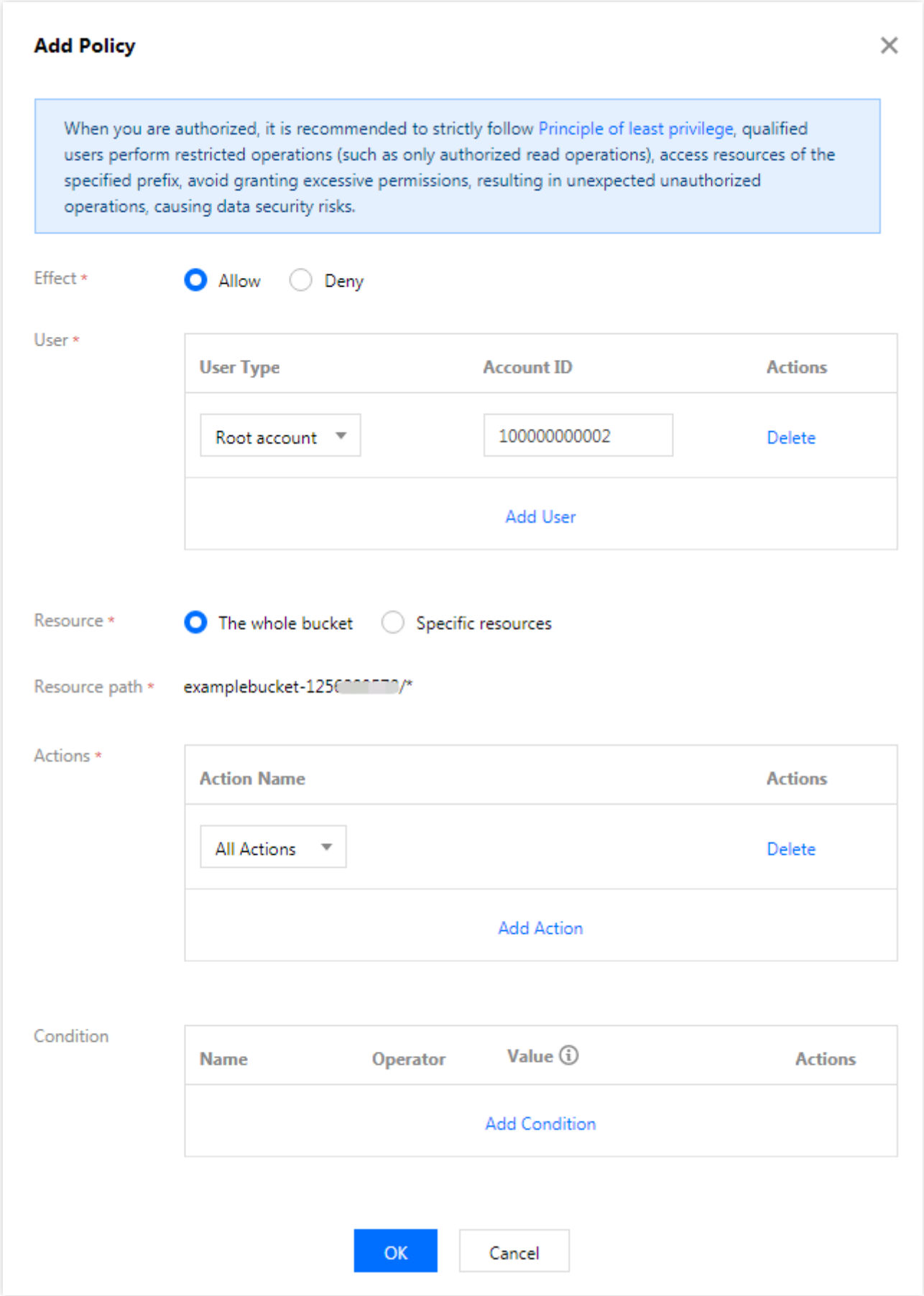
There are two buckets under root account A (APPID: `1250000000`): `examplebucket1-1250000000` and `examplebucket2-1250000000`, which sub-account B0 under root account B wants to manipulate to meet its business needs. This document describes how to authorize it to do so.

Directions

Authorizing root account B to manipulate buckets under root account A

1. Log in to the COS Console with root account A.
2. Click **Bucket List**, find the bucket to be authorized, and click its name to enter the bucket details page.
3. On the left sidebar, click **Permission Management** to enter the bucket's permission management page.
4. Locate **Permission Policy Settings**, click **Add Policy**, and select or enter the items as shown below:
   - **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.



5. Click **OK** to grant root account B specified permissions to the bucket.
6. 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

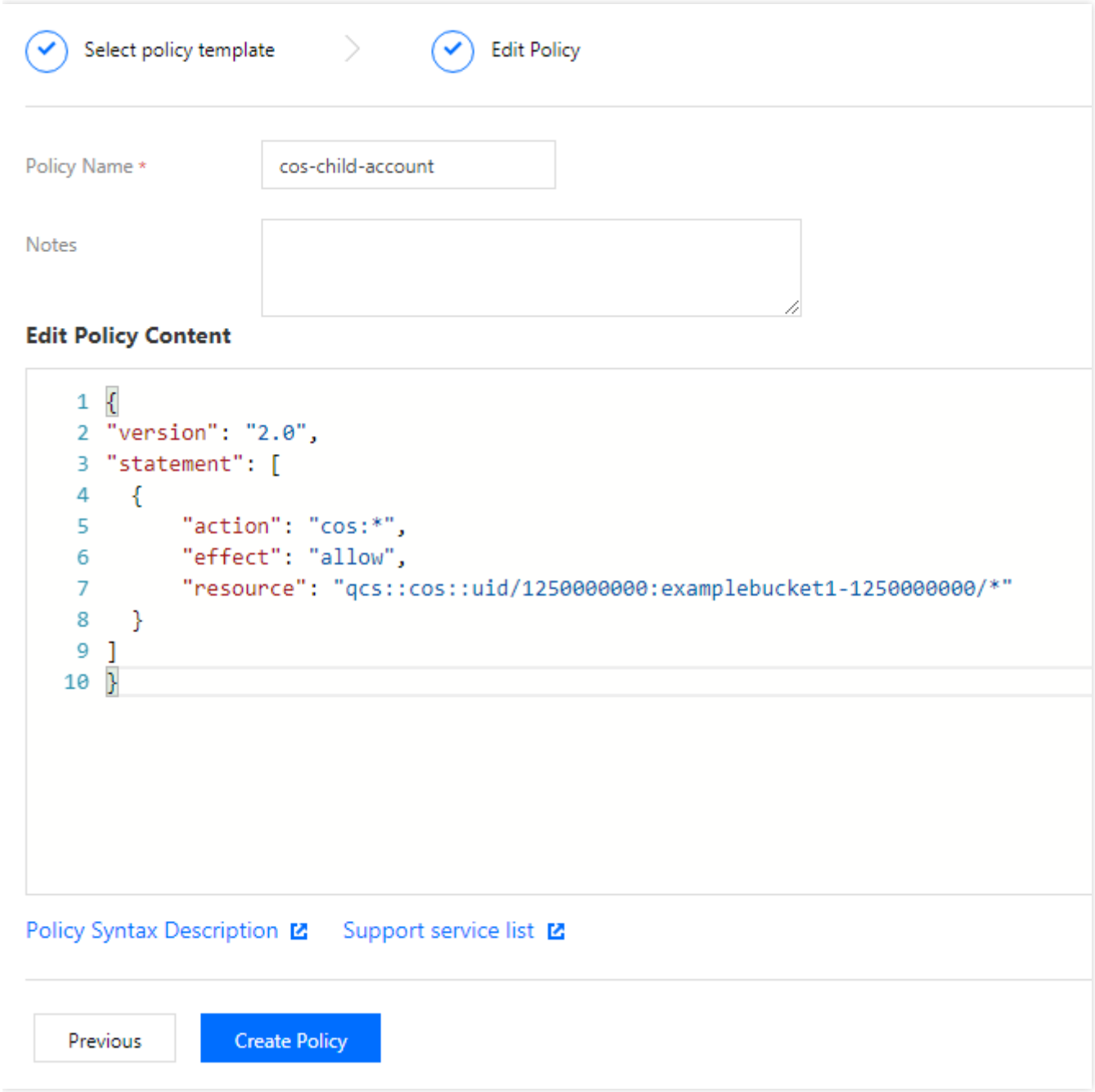
1. Log in to the CAM Console with root account B and go to the Policy page.
2. Click **Create Custom Policy** > **Create by Policy Syntax**, select a blank template, and click **Next**.

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

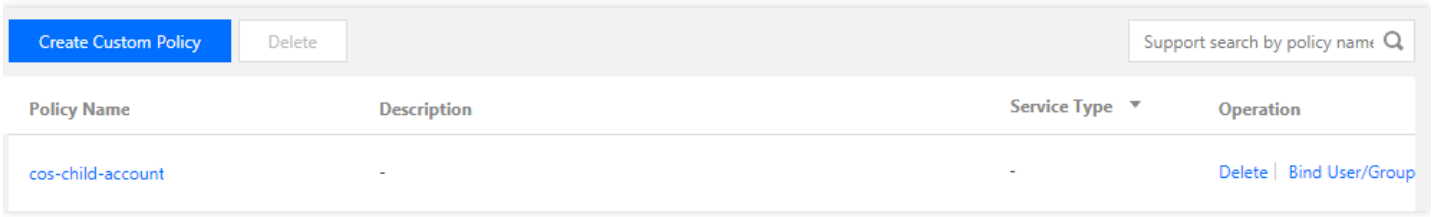
3. Fill in the form as shown below:
   - **Policy Name**: Designate a unique and meaningful name for the policy, such as `cos-child-account` .
   - **Remarks**: Optional; add remarks as needed.
   - **Policy Content**:

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

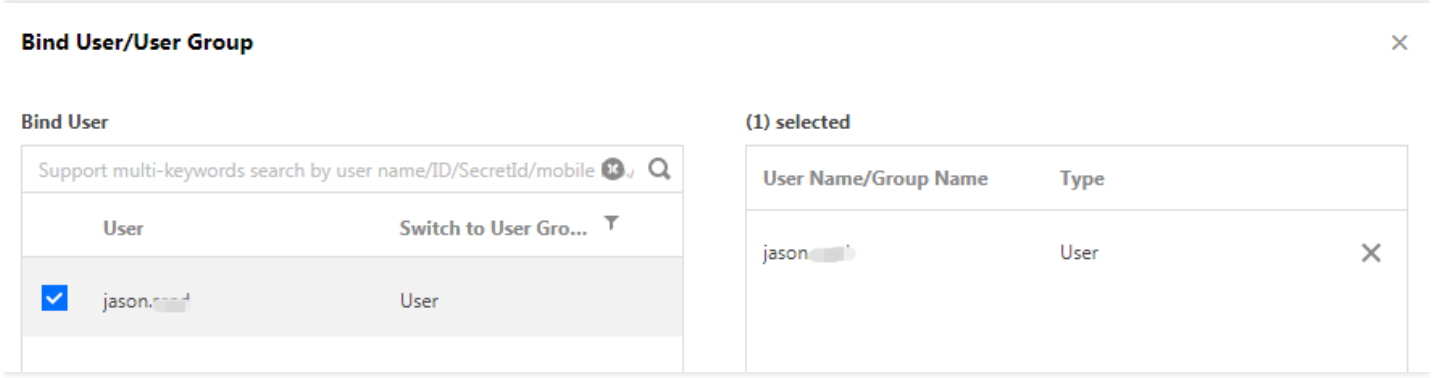
`1250000000` in `uid/1250000000` is the `APPID` of root account A, and `examplebucket1-1250000000` is the name of the bucket to authorize. `examplebucket1-1250000000/*` means that all buckets under root account A that have been authorized to root account B will be authorized to its sub-account B0.



4. Click **Done**.
5. Locate the created policy in the **policy list** and click **Bind User/User Group** on the right.



6. In the **Bind User/User Group** 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：2022-10-26 14:52:03

Note：

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

Overview

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

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

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

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

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

Directions

Mixed request load

When a large number of objects need to be uploaded, the object key you select may cause performance issues. Below is a brief description of how COS stores object key values.

Tencent Cloud maintains bucket and object key values as indexes in each service region of COS. Object keys are stored in the UTF-8 binary order in multiple index partitions. Due to such a large number of key values, using timestamps or alphabetical order, for example, may exhaust the read/write performance capacity of the partition where the key values are located. Taking the bucket path `examplebucket-1250000000.cos.ap-beijing.myqcloud.com` as an example, below are some cases that may exhaust the index performance capacity:

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

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

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

Adding hexadecimal hash prefixes

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

```
faf1-20170701/log120000.tar.gz
e073-20170701/log120500.tar.gz
```

```
333c-20170701/log121000.tar.gz
2c32-20170701/log121500.tar.gz
```

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

Adding enumerated value prefixes

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

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

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

```
logs/faf1-20170701/log120000.tar.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.

Multipart Upload Resumption in a Weak Network Environment

Last updated：2022-10-11 12:20:42

Background

This document describes how to dynamically adjust the part size during a multipart upload to COS to improve the upload success rate on poor network connections on mobile devices.

> Note：
> This feature is currently during beta test. To enable it, submit a ticket for application.

Notes

1. Currently, the minimum part size supported by COS is 100 KB.
2. The smaller the part, the more the requests. Tests show that the time consumed by requests with a part size of 100 KB is about twice that when the part size is 1 MB.
3. If a request is initiated on a normal network connection, it is recommended to set the part size to 1 MB or higher.
4. Uploading an object in environments with weak signals may fail. If this is the case, you can consider setting the part size to 100 KB and retry the upload.
5. The smaller the part, the slower the file download in the future.

Samples

SDK for Android

Below is sample code for uploading an object in multiple parts on a poor network connection:

```
import android.content.Context;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import com.tencent.cos.xml.CosXmlService;
import com.tencent.cos.xml.CosXmlServiceConfig;
```

```
import com.tencent.cos.xml.common.ClientErrorCode;
import com.tencent.cos.xml.common.Region;
import com.tencent.cos.xml.exception.CosXmlClientException;
import com.tencent.cos.xml.exception.CosXmlServiceException;
import com.tencent.cos.xml.listener.CosXmlProgressListener;
import com.tencent.cos.xml.listener.CosXmlResultListener;
import com.tencent.cos.xml.model.CosXmlRequest;
import com.tencent.cos.xml.model.CosXmlResult;
import com.tencent.cos.xml.transfer.COSXMLUploadTask;
import com.tencent.cos.xml.transfer.TransferConfig;
import com.tencent.cos.xml.transfer.TransferManager;
import com.tencent.cos.xml.transfer.TransferState;
import com.tencent.cos.xml.transfer.TransferStateListener;
import com.tencent.qcloud.core.auth.QCloudCredentialProvider;
import com.tencent.qcloud.core.auth.ShortTimeCredentialProvider;
public class ResumeHelper {
private final static String TAG = ResumeHelper.class.getSimpleName();
private Context context;
String bucket = "examplebucket-1250000000"; // Bucket in the format of BucketName
-APPID, which should be replaced with your real bucket
String region = Region.AP_Guangzhou.getRegion(); // Region of the bucket. Enter t
he region where your bucket is located
private CosXmlService cosXmlService;
private Handler mainHandler;
private boolean isTriedOnce = false; // Whether the object has been re-uploaded.
This is used to avoid unlimited re-uploads
private final int MESSAGE_RETRY = 1;
public ResumeHelper(Context context){
this.context = context;
this.mainHandler = new Handler(context.getMainLooper()){
@Override
public void handleMessage(Message msg) {
super.handleMessage(msg);
switch (msg.what){
case MESSAGE_RETRY:
/** Re-uploaded */
if(isTriedOnce)return;
isTriedOnce = true;
Parameter parameter = (Parameter) msg.obj;
/** Re-upload */
upload(parameter.srcPath, parameter.cosPath, parameter.uploadId, parameter.sliceS
ize);
break;
}
}
};
initCosXml();
```

```
}
/**
* Initialize CosXmlService
*/
private void initCosXml(){
/** Initialize CosXmlServiceConfig */
CosXmlServiceConfig cosXmlServiceConfig = new CosXmlServiceConfig.Builder()
.setDebuggable(true)
.isHttps(true)
.setRegion(region)
.builder();
// A permanent key is used here for the convenience of testing. In actual usage, it is recommended to use a temporary key
String secretId = "COS_SECRETID"; // Enter the SecretId of your TencentCloud API key
String secretKey = "COS_SECRETKEY"; // Enter the SecretKey of your TencentCloud API key
/** Initialize key information */
QCloudCredentialProvider qCloudCredentialProvider = new ShortTimeCredentialProvider(secretId, secretKey, 6000);
/** Initialize CosXmlService */
cosXmlService = new CosXmlService(context, cosXmlServiceConfig, qCloudCredentialProvider);
}
/**
* Upload
* @param srcPath Local file path
* @param cosPath COS path
* @param uploadId Whether the upload is resumed. If not, this parameter should be null
* @param sliceSize Part size set for multipart upload
*/
public void upload(final String srcPath, final String cosPath, final String uploadId, long sliceSize){
/** Set the part size for multipart upload */
TransferConfig transferConfig = new TransferConfig.Builder()
.setSliceSizeForUpload(sliceSize)
.build();
/** Initialize TransferManager */
TransferManager transferManager = new TransferManager(cosXmlService, transferConfig);
/** Start uploading: If uploadId != null, interrupted upload can be resumed */
final COSXMLUploadTask uploadTask = transferManager.upload(bucket, cosPath, srcPath, uploadId);
/** Display task status information */
uploadTask.setTransferStateListener(new TransferStateListener() {
@Override
```

```
public void onStateChanged(TransferState state) {
Log.d(TAG, "upload task state: " + state.name());
}
});
/** Display upload progress */
uploadTask.setCosXmlProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long complete, long target) {
Log.d(TAG, "upload task progress: " + complete + "/" + target);
}
});
/** Display upload result */
uploadTask.setCosXmlResultListener(new CosXmlResultListener() {
/** Upload succeeded */
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {
COSXMLUploadTask.COSXMLUploadTaskResult uploadTaskResult = (COSXMLUploadTask.COSX
MLUploadTaskResult) result;
Log.d(TAG, "upload task success: " + uploadTaskResult.printResult());
}
/** Upload failed */
@Override
public void onFail(CosXmlRequest request, CosXmlClientException exception, CosXml
ServiceException serviceException) {
Log.d(TAG, "upload task failed: " + (exception == null ? serviceException.getMess
age() :
(exception.errorCode + "," + exception.getMessage())));
if(exception != null){
/** If the failure is caused by a network reason, try setting the part size to 10
0 KB and re-run the task */
if(exception.errorCode == ClientErrorCode.IO_ERROR.getCode()
|| exception.errorCode == ClientErrorCode.POOR_NETWORK.getCode()){
Log.d(TAG, "upload task try again");
Message msg = mainHandler.obtainMessage();
msg.what = MESSAGE_RETRY;
Parameter parameter = new Parameter();
parameter.cosPath = cosPath;
parameter.srcPath = srcPath;
parameter.uploadId = uploadTask.getUploadId();
parameter.sliceSize = 100 * 1024L;
msg.obj = parameter;
mainHandler.sendMessage(msg);
}
}
}
});
}
```

```
private static class Parameter{
private String cosPath;
private String srcPath;
private String uploadId;
private long sliceSize;
}
}
```

SDK for iOS

Below is sample code for uploading an object in multiple parts on a poor network connection:

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];
NSURL* url = [NSURL fileURLWithPath:@"filePathURL"];
put.object = @"text.txt";
put.bucket = self.bucket;
put.body = url;
[put setSendProcessBlock:^(int64_t bytesSent, int64_t totalBytesSent, int64_t tot
alBytesExpectedToSend) {
NSLog(@"upload %lld totalSend %lld aim %lld", bytesSent, totalBytesSent, totalByt
esExpectedToSend);
}];
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
/**
1. Request timeout may be caused by a slow network connection. Upload resuming in
formation (resumeData) can be obtained in error (in the QCloudUploadResumeDataKey
field).
2. Call the requestWithRequestData of QCloudCOSXMLUploadObjectRequest to pass in
the resumeData and generate a new request.
3. Reset the part size through the sliceSize of QCloudCOSXMLUploadObjectRequest a
s needed.
4. Start uploading
*/
if (error.code == -1001) {
NSData *resumeData =error.userInfo[QCloudUploadResumeDataKey];
if (resumeData) {
QCloudCOSXMLUploadObjectRequest* request = [QCloudCOSXMLUploadObjectRequest reque
stWithRequestData:resumeData];
request.sliceSize = 100*1024;
[request setFinishBlock:^(QCloudUploadObjectResult* outputObject, NSError *error)
{
NSLog(@"result: %@",outputObject);
}];

[request setSendProcessBlock:^(int64_t bytesSent, int64_t totalBytesSent, int64_t
totalBytesExpectedToSend) {
```

```
NSLog(@"upload %lld totalSend %lld aim %lld", bytesSent, totalBytesSent, totalByt
esExpectedToSend);
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:reques
t];
}
}
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

COS Stress Test Guide

Last updated：2022-12-28 14:08:38

COSBench Overview

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

System Environment

We recommend you run COSBench on CentOS 7.0 or later. If you run it on Ubuntu, unexpected issues may occur.

Performance Factors

- **CPU cores**: A small number of CPU cores coupled with a large number of running workers is very likely to cause high overheads for context switching. Therefore, 32 or 64 cores are recommended for the test.
- **NIC**: The outbound traffic from the server is limited by the NIC. To test the traffic for large files, an NIC above 10 GbE is recommended.
- **Request network linkage**: Public network linkages vary in quality. Charges will incur for public network downstream traffic for downloads over the public network. Therefore, private network is recommended for access within the same region.
- **Test duration**: In order to get a reliable value, a longer test period is recommended.
- **Test environment**: The version of the JDK running on your program is also a key performance factor. For example, when testing on HTTPS, with an earlier JDK version, GCM bugs may occur for the encryption algorithm, as well as the locking issues for the random number generator.

Directions

1. Download COSBench 0.4.2.c4.zip from GitHub and decompress it on your server.
2. Install the COSBench dependency library and run the following command.

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

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

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

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

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

   i. init: Creates a bucket.
   ii. prepare: Uses worker threads to PUT (upload) objects in specified size for read in the main stage.
   iii. main: Uses worker threads to read and write objects for a specified period of time.
   iv. cleanup: Deletes the created objects.
   v. dispose: Deletes the bucket.

   The sample configuration is as shown below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<workload name="s3-50M-sample" description="sample benchmark for s3">
<storage type="s3" config="accesskey=AKIDHZRLB9Ibhdp7Y7gyQq6BOk1997xxxxxx;secre
tkey=YaWIuQmCSZ5ZMniUM6hiaLxHnxxxxxx;endpoint=http://cos.ap-beijing.myqcloud.co
m" />
<workflow>
<workstage name="init">
<work type="init" workers="10" config="cprefix=examplebucket;csuffix=-125000000
0;containers=r(1,10)" />
</workstage>
<workstage name="prepare">
<work type="prepare" workers="100" config="cprefix=examplebucket;csuffix=-12500
00000;containers=r(1,10);objects=r(1,1000);sizes=c(50)MB" />
</workstage>
<workstage name="main">
<work name="main" workers="100" runtime="300">
<operation type="read" ratio="50" config="cprefix=examplebucket;csuffix=-125000
0000;containers=u(1,10);objects=u(1,1000)" />
<operation type="write" ratio="50" config="cprefix=examplebucket;csuffix=-12500
00000;containers=u(1,10);objects=u(1000,2000);sizes=c(50)MB" />
</work>
</workstage>
<workstage name="cleanup">
<work type="cleanup" workers="10" config="cprefix=examplebucket;csuffix=-125000
0000;containers=r(1,10);objects=r(1,2000)" />
```

```
</workstage>
<workstage name="dispose">
<work type="dispose" workers="10" config="cprefix=examplebucket;csuffix=-125000
0000;containers=r(1,10)" />
</workstage>
</workflow>
</workload>
```

Parameter description

Parameter	Description
accesskey, secretkey	Key information. Replace them with your own `SecretId` and `SecretKey`.
cprefix	Bucket name prefix, such as `examplebucket`.
containers	Value range for bucket names. A bucket name is made up of `cprefix` and `containers`, such as `examplebucket1` or `examplebucket2`.
csuffix	Your `APPID`. Note that `APPID` is prefixed with a `-`, such as `-1250000000`.
runtime	Duration of the stress test
ratio	Ratio of reads to writes
workers	Number of stress test threads

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

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

5. Start the COSBench service.

- For CentOS, run the following command:

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

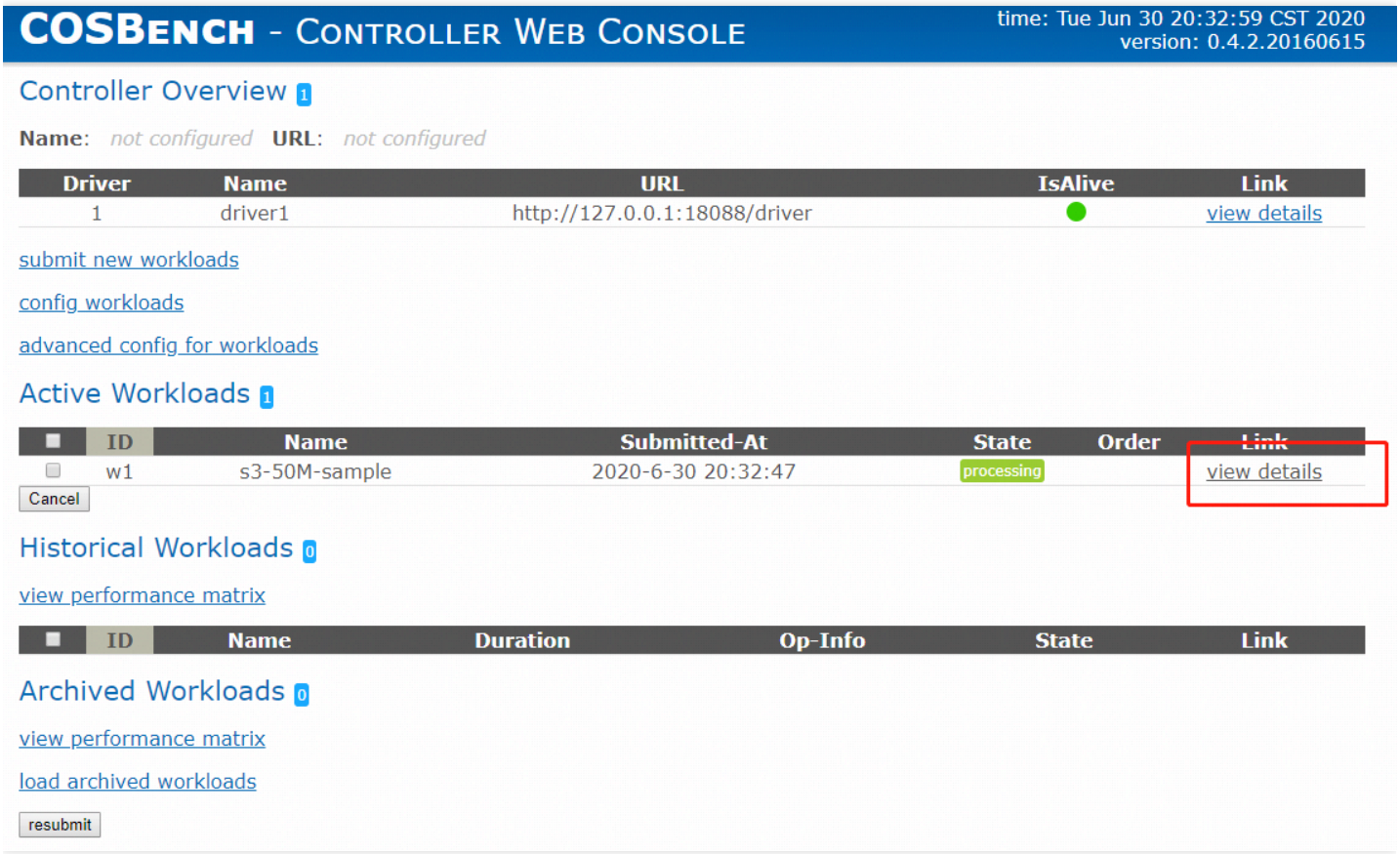
- For Ubuntu, run the following command:

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

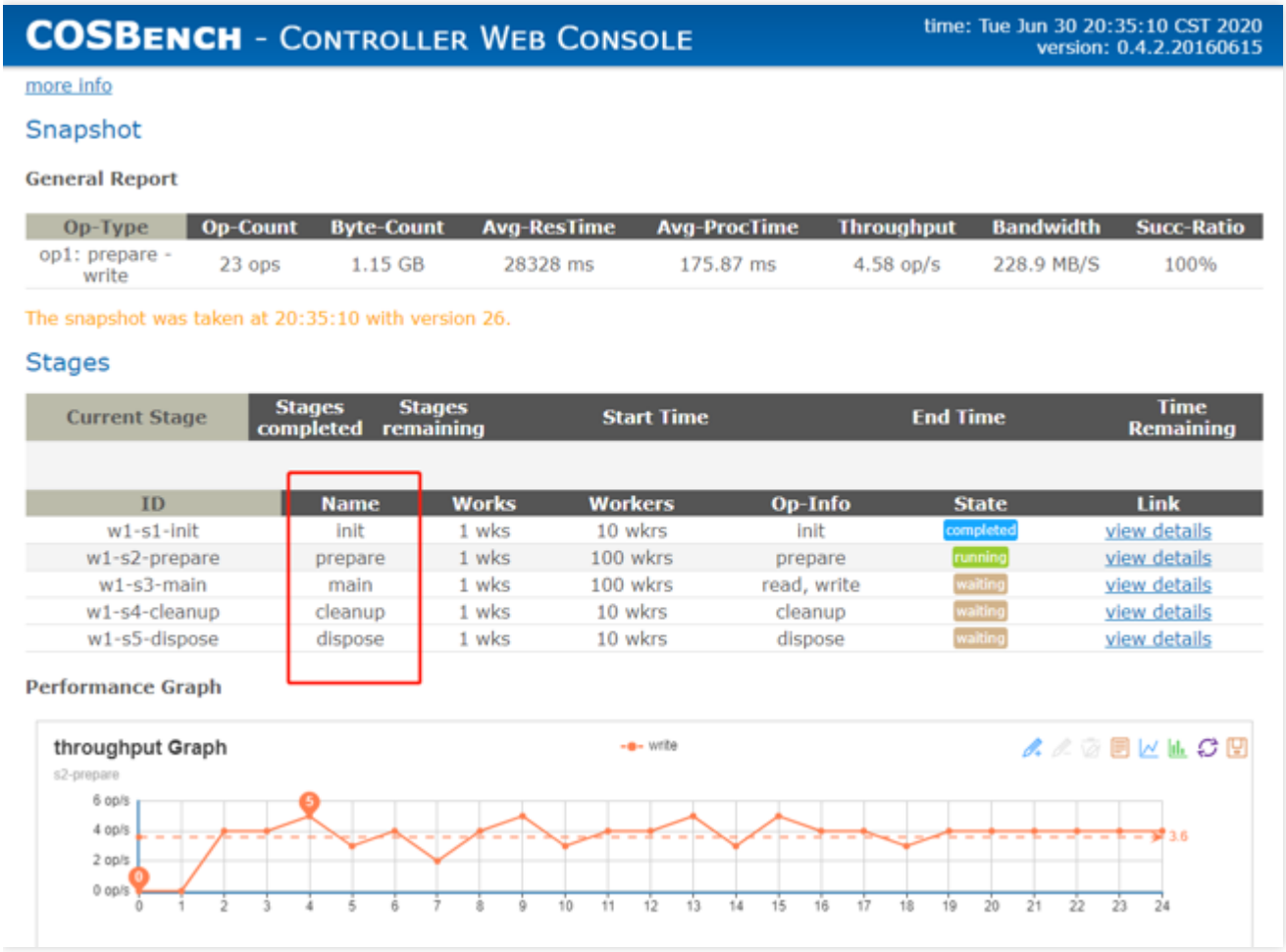
6. Run the following command to submit the job.

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

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



You can see the five stages as shown below:



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

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

2. main: 100 workers are run to read and write objects in parallel for 300 seconds.

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

Basic Info

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

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

more info

Final Result

General Report

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

show peformance details

8. Run the following command to stop the test.

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

Data Migration
Migrating Local Data to COS

Last updated：2020-01-06 14:15:27

Practice Scenario

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

Migration Method	Description
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 completing simple configurations.
Cloud Data Migration (CDM) (offline migration)	CDM uses a dedicated offline migration device provided by Tencent Cloud to help you migrate your local data to the cloud, solving the problems that arise from online migration from a local IDC to cloud, such as long time, high costs, and low security.

You can determine how to migrate data based on the data volume, IDC egress bandwidth, IDC idle server resources, acceptable completion time, and other factors. The estimated time needed for online migration is shown in the figure below. As can be seen, if data migration needs to take more than 10 days or the amount of data to be migrated

exceeds 50 TB, you are recommended to use CDM for offline migration; otherwise, online migration is a better choice.

Data volume \ Bandwidth	10Mbps	100Mbps	1Gbps	10Gbps
10GB	3 hours	17 minutes	2 minutes	10 seconds
100GB	30 hours	3 hours	17 minutes	2 minutes
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

Data migration may be affected if there is a high number of small files below 1 MB or the performance of disk IO is insufficient.

Migration Practice

COS Migration

The steps are as follows:

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

For more information, please see COS Migration Tool.

Tips

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

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

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

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

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 a device.
3. After receiving the device, copy your data to it as instructed in the migration device user manual.
4. Then, submit a return application in the console and wait for Tencent Cloud to migrate your data to COS.

For more information, please see Cloud Data Migration (CDM).

Tips

Below describes how to migrate data to COS offline efficiently and securely.

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 though CDM. You can monitor the CPU and memory utilization of the device and if the current migration speed is lower than expected, you can speed the migration up in the following ways:
   - Connect multiple devices to the same CDM device via different network interfaces.
   - Connect multiple devices to several CDM devices via different network interfaces.

Migrating Data from Third-Party Cloud Storage Service to COS

Last updated：2022-11-08 14:36:47

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
- Migrating from Huawei OBS
- Migrating from Qiniu KODO
- Migrating from UCloud UFile

- Migrating from Kingsoft Cloud KS3
- Migrating from Baidu AI Cloud BOS
- AWS S3 Migration Tutorial

Tips

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

Migrating Data from URL to COS

Last updated：2020-01-06 14:24:28

Background

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

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

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

Migration Practice

Migration Service Platform (MSP)

The steps are as follows:

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

For more information, please see Migrating from a URL List.

Tips

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

COS origin-pull

The steps are as follows:

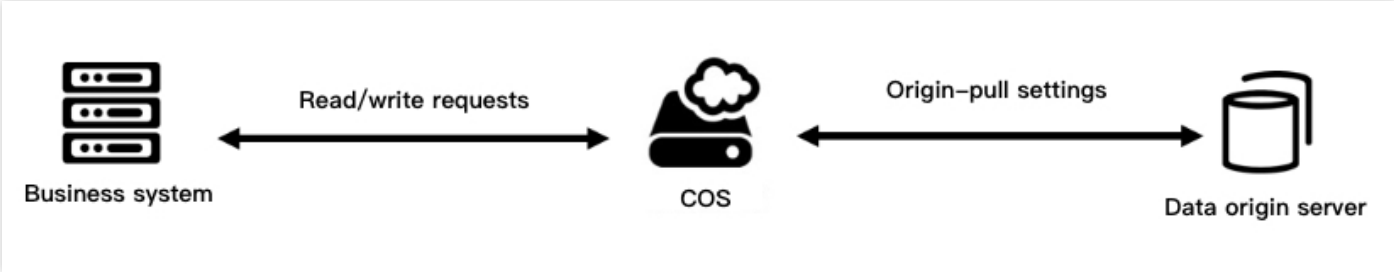
1. Log in to the COS Console and enable the origin-pull setting in the destination bucket.
2. Configure the bucket origin-pull address and save the configuration.
3. Redirect the read/write requests from your business system to COS.

For more information, please see Setting Origin-Pull.

Tips

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

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



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

COS Migration

The steps are as follows:

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

For more information, please see COS Migration Tool.

Tips

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

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

Parameter Name	Parameter Description
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 servers and perform migration tasks separately for different data sources.

Migrating Data Within COS

Last updated：2021-03-08 17:29:18

Background

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

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

Migration Practice

MSP

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

The steps are as follows:

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

For more information, please see Migrating Between Tencent Cloud COS.

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

Cross-Bucket replication

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

Migrating Data Between HDFS and COS

Last updated：2021-02-25 15:30:32

Overview

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

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

Prerequisites

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

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

   If the files in the COS bucket can be listed correctly, it means that Hadoop-COS has been installed and configured correctly, and the following steps can be performed.
2. The COS access account must have read and write permission for the destination path of the COS bucket.

⚠ Note：

- You can authorize sub-accounts read/write permissions for resources in the COS bucket as needed. You are advised to authorize by referring to Notes on Principle of Least Privilege and Setting Sub-user Permissions. The common preset policies are as follows:
  - DataFullControl: full control over data, including the permission to read, write, as well as delete files, and list the file list.
  - QcloudCOSDataReadOnly: read-only permission
  - QcloudCOSDataWriteOnly: write-only permission
- The custom monitoring feature requires Cloud Monitoring to have permission to report metrics and read API operations. Please grant the QcloudMonitorFullAccess permission with caution.

Directions

Copying files from HDFS to a COS bucket

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

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

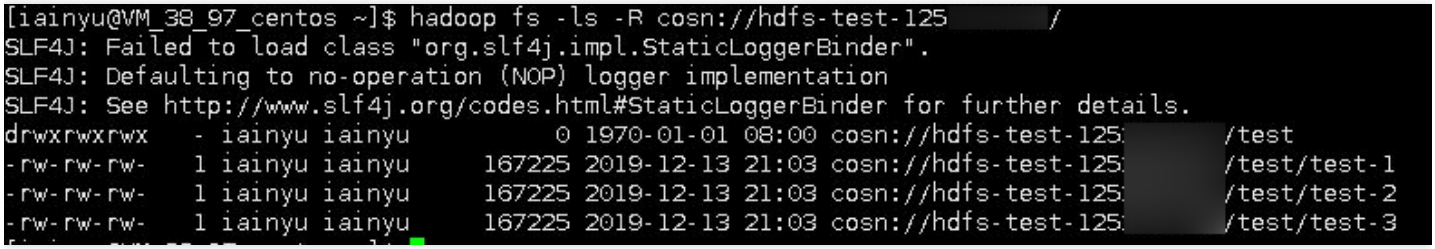
1. Run the following command to start the migration:

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

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

```
                Merged Map outputs=0
                GC time elapsed (ms)=0
                Total committed heap usage (bytes)=253755392
        File Input Format Counters
                Bytes Read=640
        File Output Format Counters
                Bytes Written=8
        DistCp Counters
                Bytes Copied=501675
                Bytes Expected=501675
                Files Copied=4
19/12/13 21:03:35 INFO mapred.LocalJobRunner: Finishing task: attempt_local1986740758_0001_m_000000_0
19/12/13 21:03:35 INFO mapred.LocalJobRunner: map task executor complete.
19/12/13 21:03:35 INFO mapred.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. Run the `hadoop fs -ls -R cosn://hdfs-test-1250000000/` command to list the directories and files that have just been migrated to the bucket `hdfs-test-1250000000`.



Copying files from a COS bucket to a local HDFS cluster

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

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

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

Note：

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

Run the following command:

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

Parameter description:

- Dfs.cosn.impl: Always set it to `org.apache.hadoop.fs.CosFileSystem`.
- Dfs.cosn.bucket.region: bucket region. You can view the region on the bucket list page of the COS console.
- Dfs.cosn.userinfo.secretId: Enter the `SecretId` of the bucket owner's account. The value can be obtained at Manage API Key.
- Dfs.cosn.userinfo.secretKey: Enter the `SecretKey` of the bucket owner's account. The value can be obtained at Manage API Key.

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

Note：

For other parameters, please see Hadoop-COS.

Additional Hadoop DistCp Parameters

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

Accessing COS with AWS S3 SDK

Last updated：2022-09-14 14:43:38

Overview

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

Preparations

1. 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.
2. You have a client application that has been integrated with the S3 SDK and runs properly.

Android

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

Initialization

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

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

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

iOS

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

1. Implement the `AWSCredentialsProvider` protocol

```
-(AWSTask<AWSCredentials *> *)credentials{
// Here, the backend requests for a temporary key from STS.
AWSCredentials *credential = [[AWSCredentials alloc]initWithAccessKey:@"<TempSecr
etID>" secretKey:@"<TempSecretKey>" sessionKey:@"<STSSessionToken>" expiration:[N
SDate 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 serv
ice:AWSServiceS3 URL:bucketURL];
AWSServiceConfiguration* configuration = [[AWSServiceConfiguration alloc]
initWithRegion:AWSRegionUSEast2 endpoint:endpoint
credentialsProvider:[MyCredentialProvider new]]; // `MyCredentialProvider` implem
ents the `AWSCredentialsProvider` protocol.

[[AWSServiceManager defaultServiceManager] setDefaultServiceConfiguration:configu
ration];
```

Node.js

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

Initialization

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

```
var AWS = require('aws-sdk');
AWS.config.update({
accessKeyId: "COS_SECRETID",
secretAccessKey: "COS_SECRETKEY",
region: "ap-guangzhou",
endpoint: 'https://cos.ap-guangzhou.myqcloud.com',
});
s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

Java

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

1. Modify the configuration and certificate files of AWS

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

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

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

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

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

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

2. Set the `Endpoint` in the code

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

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

Python

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

1. Modify the configuration and certificate files of AWS

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

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

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

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

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

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

2. Set the `Endpoint` in the code

Suppose the bucket region is `ap-guangzhou`:

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

PHP

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

1. Modify the configuration and certificate files of AWS

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

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

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

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

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

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

2. Set the `Endpoint` in the code

Suppose the bucket region is `ap-guangzhou`:

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

.NET

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

Initialization

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

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

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

Go

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

1. Create a session based on the key

Suppose the bucket region is `ap-guangzhou`:

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

2. Create a server initiation request based on the session

```
sess, _ := newSession()
service := s3.New(sess)
// Take file upload as an example.
fp, _ := os.Open("yourLocalFilePath")
defer fp.Close()
ctx, cancel := context.WithTimeout(context.Background(), time.Duration(30)*time.S
econd)
defer cancel()
service.PutObjectWithContext(ctx, &s3.PutObjectInput{
Bucket: aws.String("examplebucket-1250000000"),
Key: aws.String("exampleobject"),
Body: fp,
})
```

C++

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

1. Modify the configuration and certificate files of AWS

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

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

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

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

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

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

2. Set the `Endpoint` in the code

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

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

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

Last updated：2020-11-13 11:30:54

Overview

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

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

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

Backup and Disaster Recovery Solution Based on Cross-Bucket Replication

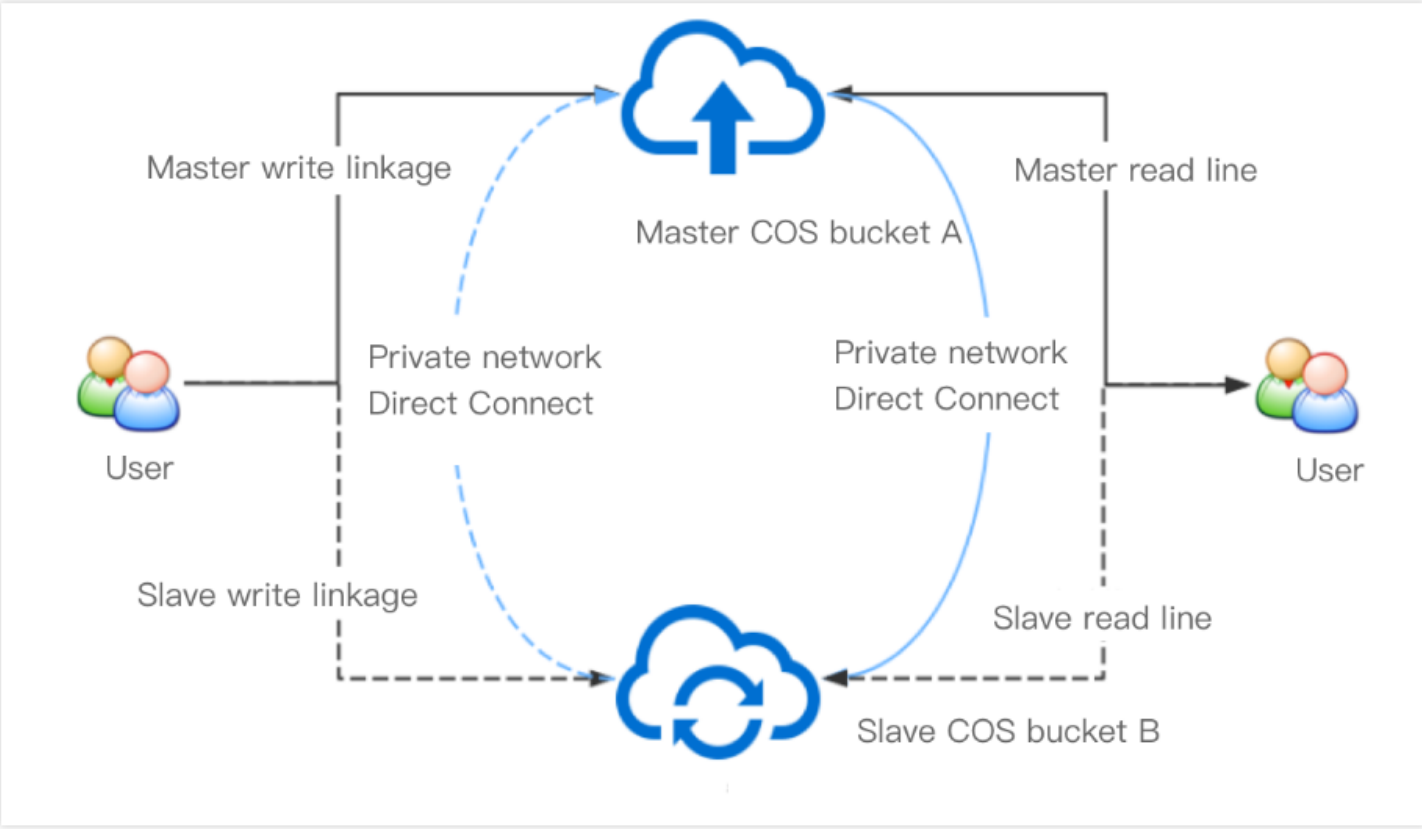
Disaster recovery entails three elements: redundancy, remote, and replication.

- Redundancy: data should be backed up simultaneously to another available system.
- Remote: data backups should be stored in another remote region, as disasters often extend geographically and only a long enough distance can guarantee the availability of redundant data.
- Replication: data loss during backup should be reduced down to zero.

COS cross-bucket replication enables cross-bucket syncing of incremental data. Data uploaded to a bucket can be replicated to another bucket in seconds or minutes, depending on file size and distance. Cross-bucket replication allows you to make remote redundant backups of your data for disaster recovery and business continuity. For more

information, please see Cross-Bucket Replication Overview. To enable this feature, you need to enable versioning first. For more information on versioning, please see Versioning Overview.

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



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

⚠ Note：

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

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

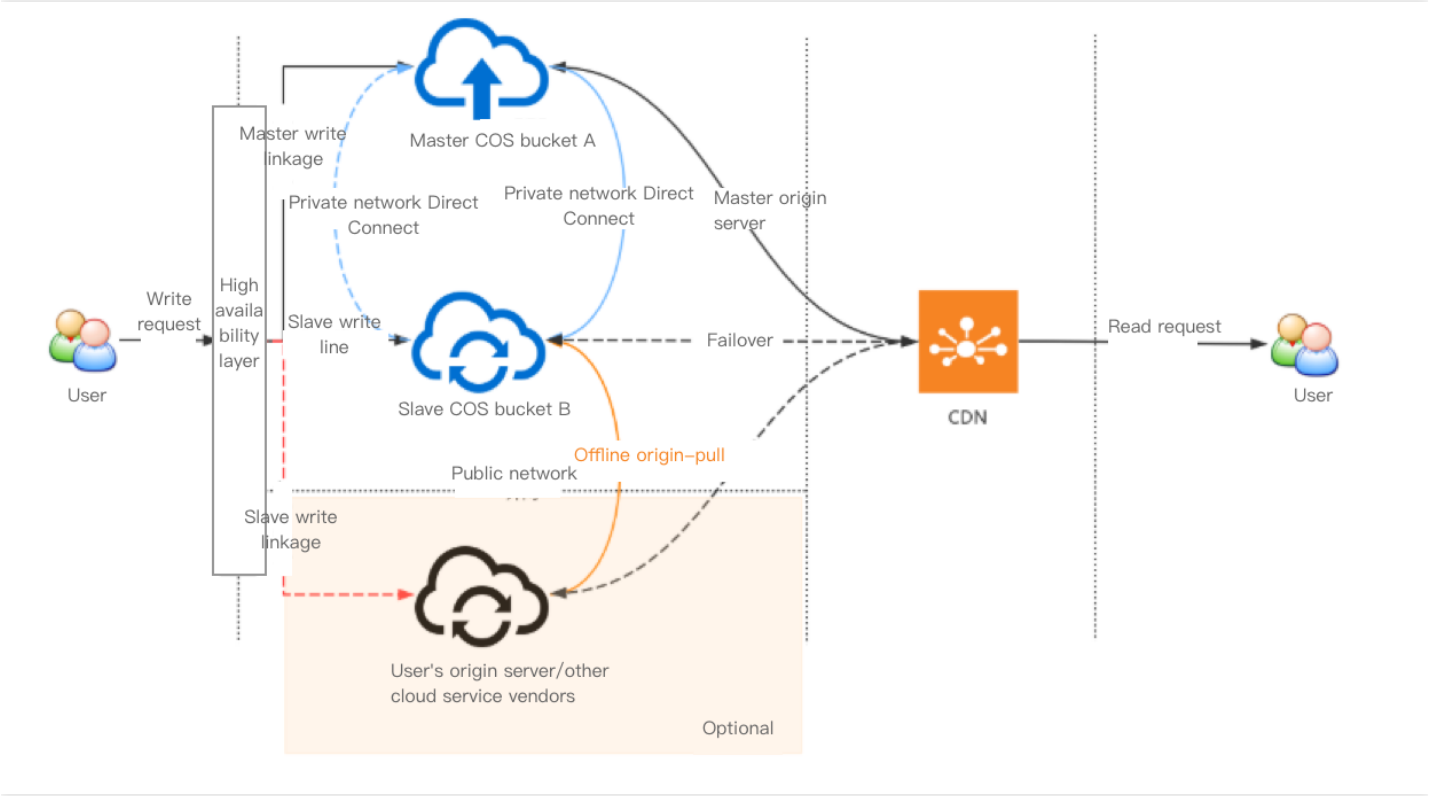
⚠ Note：

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

High-Availability Solution Based on Cross-Bucket Replication

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

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



This architecture consists of the following layers:

- **High availability layer**: integrates network detection and service scheduling and switches links based on metrics such as link connectivity, which can be implemented with the aid of SCF (as described in the previous section) or on the client side according to your business needs.
- **Storage layer**: typically consists of COS buckets in different regions. You can also **introduce buckets from external origin servers or other cloud vendors** by setting origin-pull policies so as to further guarantee data consistency.

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

The following explains how this architecture guarantees high availability:

1. Normally, all your write requests point to bucket A where all incremental data will be automatically replicated to bucket B as backups.
2. When the links to bucket A fail (for example, the quality of automated testing declines or an upload fails), the client can switch the links to bucket B. In this case, all incremental data in bucket B will also be automatically replicated to bucket A.
3. You can also choose to make a redundant backup of your data on an external origin server or in another cloud first and then configure an origin-pull policy for bucket B. If, in extreme cases, the links to both buckets A and B fail, bucket B can pull data from the origin server when the attempt to upload data to bucket B fails.

⚠ Note：

- As full redundant backups are costly, you can choose to make redundant backups of only hot data (such as files uploaded in just a few hours) so as to reduce data storage costs.
- If you choose an origin server as part of the high availability architecture, please be sure to assess the bandwidth of the origin server and the possible impact of the limit on it when designing the architecture.

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

ⓘ Note：

- If you want to read data from your bucket directly, your client should be able to follow 302 redirects in the HTTP protocol.
- Tencent Cloud CDN boasts nearly a thousand edge servers which provide adjacent access nodes to increase the data read speed. You can bind multiple origin servers to CDN as master and slave servers in order to ensure high availability. For more information, please see Origin Server Configuration.
- If you want to secure your origin servers as much as possible, you can set private-read/write permission for them and enable CDN origin-pull authentication so as to allow your end users to anonymously access the data cached on the CDN edge servers whiling protecting the security of the data on the origin servers.

References

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

- Versioning Overview
- Cross-Bucket Replication Overview
- Setting Origin-Pull
- Setting CDN Acceleration
- Origin Server Configuration
- Adding Domain Names

Cloud Data Backup

Last updated：2020-03-25 18:20:29

Overview

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

- Backup and disaster recovery scheme based on cross-region replication
- Batch data replication scheme (COS batch operation feature)
- Migration and backup scheme based on Migration Service Platform (MSP)

Backup and Disaster Recovery Scheme Based on Cross-region Replication

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

For more information, please see High-Availability Disaster Recovery Architecture Based on Cross-region Replication.

Use cases

- **Remote disaster recovery:** COS has 11 nines of availability for object data, but there is still a slight chance of data loss due to force majeure such as wars and natural disasters. If you want to avoid data loss by explicitly having a separate copy in a different region, you can use cross-region replication to achieve remote disaster recovery. In this way, when the IDC in one region is damaged due to force majeure, the IDC in the other region can still provide data copies for your use.
- **Compliance:** COS ensures data availability by providing multiple copies and erasure codes for data in physical disks by default. However, there may be compliance requirements in some industries stipulating that you keep copies in another region. Cross-region replication allows data to be replicated across regions to meet such requirements.
- **Access latency reduction:** when you have end users accessing objects from different regions, with cross-region replication, you can maintain object copies in the available storage regions closest to them, which can minimize access latency to deliver a better user experience.

- **Special technical requirements:** if you have computing clusters in two different regions and the clusters need to process the same set of data, with cross-region replication, you can maintain object copies in both regions.
- **Data migration and backup:** you can copy your business data from one availability region to another one as needed to implement data migration and backup.

Usage

For more information, please see Setting Cross-Region Replication.

Batch Data Replication Scheme

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

Use cases

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

Usage

For more information, please see Batch Operation.

Instructions

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

- All objects to be replicated must be in the same bucket.
- Only one destination bucket can be configured for a batch replication job.
- You need to have permission to read objects from the source bucket and write objects into the destination bucket.
- The total size of the objects cannot exceed 5 GB.
- Verification via ETags and server-side encryption using custom keys are not supported.
- If the destination bucket does not have versioning enabled and contains an object file with the same name as a file to be replicated, COS will overwrite the object file.

Data Migration and Backup Scheme

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

Local Data Backup

Last updated：2020-03-25 18:21:11

Overview

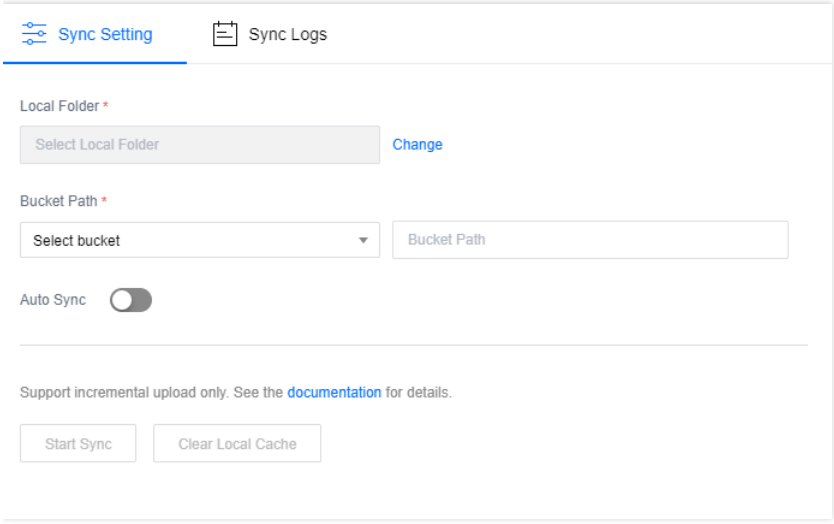
This document describes how to back up local data to the cloud. COS provides the following three methods to facilitate backup of local data to a COS bucket:

- Backup based on file sync through COSBrowser
- Backup based on online migration through COS Migration
- Backup based on offline migration through CDM

Backup Based on File Sync

COSBrowser is a visual cloud file manager launched by COS that allows you to easily view, transfer, and manage COS resources through simple interactions. Currently, COSBrowser is available in Desktop Edition and Mobile Edition. For more information, please see COSBrowser Overview.

COSBrowser Desktop Edition has a file sync feature that can be used to sync and upload local files to the cloud by associating local folders with buckets.



Usage

For more information, please see COSBrowser Instructions.

Online Migration Scheme

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

Use cases

If you have a local IDC, COS Migration can help you migrate massive amounts of local data to COS.

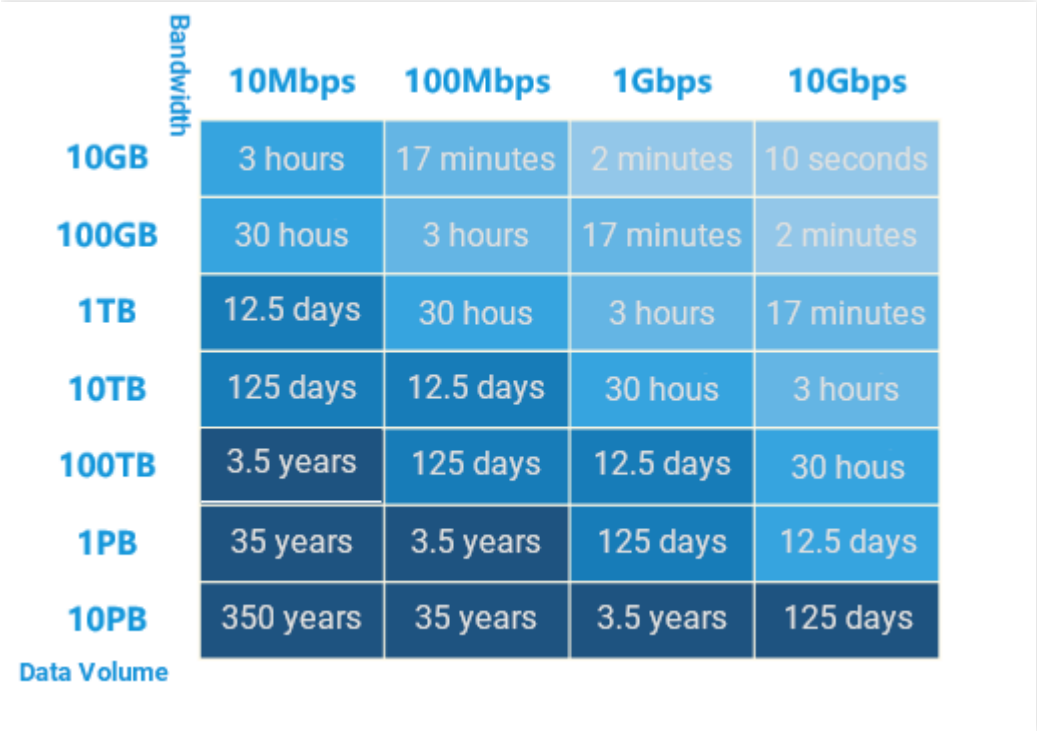
Usage

For more information, please see COS Migration.

Offline Migration Scheme

CDM uses a dedicated offline migration device provided by Tencent Cloud to help you migrate your local data to the cloud, solving the problems that arise from online migration from a local IDC to cloud, such as long time, high costs, and low security.

You can determine how to migrate data based on the data volume, IDC egress bandwidth, IDC idle server resources, acceptable completion time, and other factors. The estimated time needed for online migration is shown in the figure below. As can be seen, if data migration needs to take more than 10 days or the amount of data to be migrated exceeds 50 TB, you are recommended to use CDM for offline migration.



Usage

For more information, please see CDM

Direct Data Upload
Practice of Direct Transfer for Web End

Last updated：2022-08-23 14:25:04

Overview

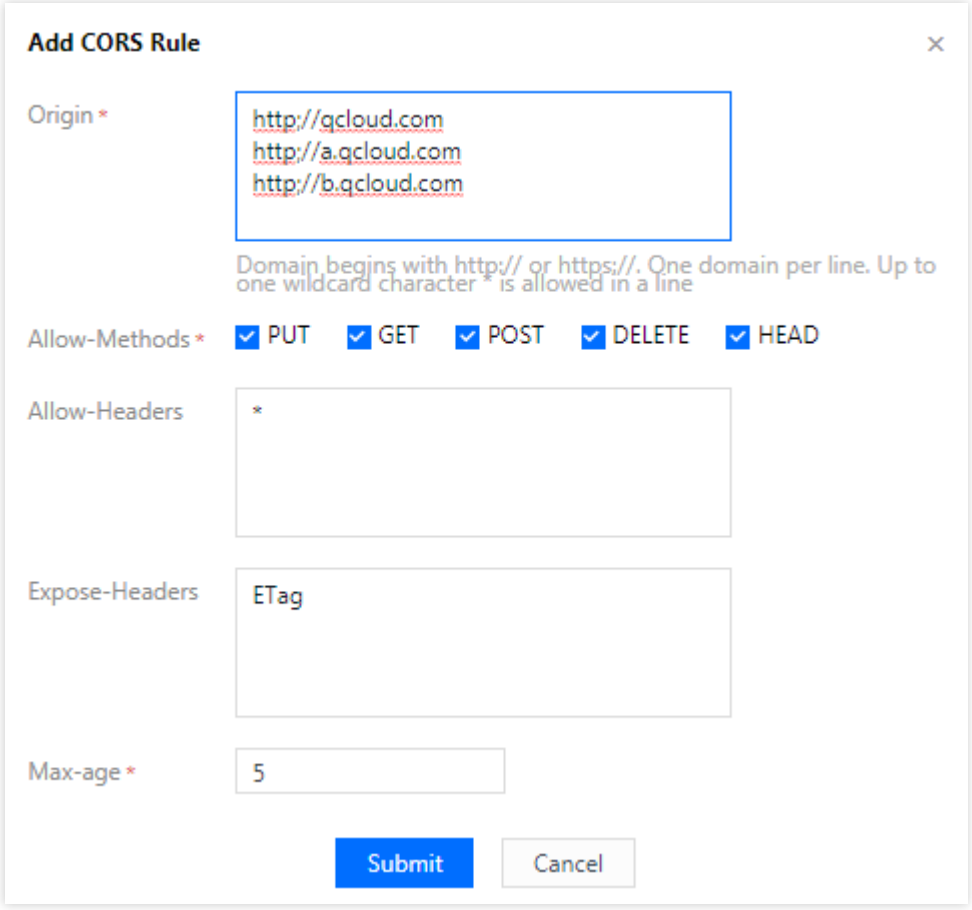
This document describes how to use simple code to upload files to a COS bucket directly from the web without using an SDK.

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

Prerequisites

1. Log in to the COS console and create a bucket to obtain the `Bucket` (bucket name) and `Region` (region name). For more information, please see Creating Buckets.
2. Go to the bucket detail page, choose the **Security Management** tab, and select **CORS (Cross-Origin Resource Sharing)** from the drop-down list. Then, click **Add a Rule**. A configuration example is shown in the following figure.

For more information, please see Setting Cross-Origin Access.



3. Log in to the CAM console and obtain the `SecretId` and `SecretKey` of your project.

Directions

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

Getting temporary key and calculating signature

For security reasons, the signature uses a temporary key. For building a temporary key service on the server, please see PHP Sample and Nodejs Sample.
To use other languages or implement it on your own, please take the following steps:

1. Obtain a temporary key from the server. The server first uses the `SecretId` and `SecretKey` of a fixed key to obtain the `tmpSecretId` , `tmpSecretKey` , and `sessionToken` of the temporary key from the STS service. For more information, please see Generating and Using Temporary Keys or cos-sts-sdk.

2. The frontend calculates the signature based on the `tmpSecretId`, `tmpSecretKey`, `method`, and `pathname`. You can use cos-auth.js to calculate the signature as described in this document. If required by the actual business, the signature can also be calculated at the backend.
3. If you use the `PutObject` API for the upload, you can specify the calculated signature and `sessionToken` in the `authorization` and `x-cos-security-token` fields, respectively, in the request header.
   If you use the `PostObject` API for the upload, you can specify the calculated signature and `sessionToken` in the `Signature` and `x-cos-security-token` fields, respectively, in the request form.

Frontend upload

Solution A: Upload with AJAX

To upload with AJAX, your browser needs to support the basic features of HTML5. You can implement this solution by referring to PUT Object and taking the steps below:

1. Obtain the bucket information by taking the steps in Prerequisites.
2. Create a `test.html` file. Then, modify the values `Bucket` and `Region` in the following code and copy the code to 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.

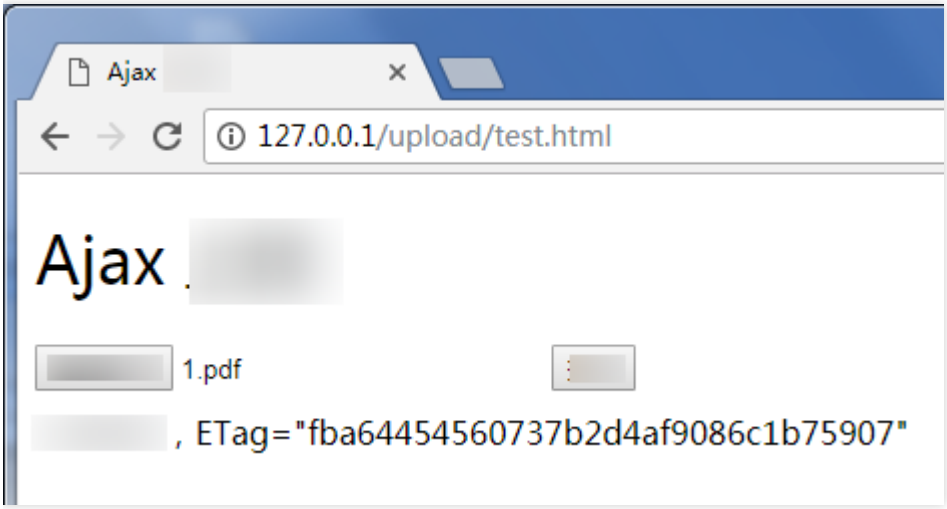
```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Ajax Put Upload</title>
<style>
h1, h2 {
font-weight: normal;
}
#msg {
margin-top: 10px;
}
</style>
</head>
<body>
<h1>Ajax Put Upload</h1>
<input id="fileSelector" type="file">
<input id="submitBtn" type="submit">
<div id="msg"></div>
<script src="https://unpkg.com/cos-js-sdk-v5/demo/common/cos-auth.min.js"></scrip
t>
<script>
```

```
(function () {
// Parameters used for the request
var Bucket = 'examplebucket-1250000000';
var Region = 'ap-guangzhou';
var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/'; // T
he prefix is used to concatenate the request URL. Use the default bucket endpoin
t.
// URL-encode more characters.
var camSafeUrlEncode = function (str) {
return encodeURIComponent(str)
.replace(/!/g, '%21')
.replace(/'/g, '%27')
.replace(/\(/g, '%28')
.replace(/\)/g, '%29')
.replace(/\*/g, '%2A');
};
// Calculate the signature.
var getAuthorization = function (options, callback) {
// var url = 'http://127.0.0.1:3000/sts-auth' +
var url = '../server/sts.php';
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.onload = function (e) {
var credentials;
try {
credentials = (new Function('return ' + xhr.responseText))().credentials;
} catch (e) {}
if (credentials) {
callback(null, {
SecurityToken: credentials.sessionToken,
Authorization: CosAuth({
SecretId: credentials.tmpSecretId,
SecretKey: credentials.tmpSecretKey,
Method: options.Method,
Pathname: options.Pathname,
})
});
} else {
console.error(xhr.responseText);
callback('Signature obtaining error');
}
};
xhr.onerror = function (e) {
callback('Signature obtaining error');
};
xhr.send();
```

```
};
// Upload a file.
var uploadFile = function (file, callback) {
var Key = 'dir/' + file.name; // File directory and filename
getAuthorization({Method: 'PUT', Pathname: '/' + Key}, function (err, info) {
if (err) {
alert(err);
return;
}
var auth = info.Authorization;
var SecurityToken = info.SecurityToken;
var url = prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/');
var 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)) {
var 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. Check whether the CORS rule has been se
t');
};
xhr.send(file);
});
};
// Listen on form submission.
document.getElementById('submitBtn').onclick = function (e) {
var file = document.getElementById('fileSelector').files[0];
if (!file) {
document.getElementById('msg').innerText = 'File to upload not selected';
return;
}
file && uploadFile(file, function (err, data) {
console.log(err || data);
document.getElementById('msg').innerText = err ? err : ('Upload successful, ETag
=' + data.ETag);
});
```

```
};
})();
</script>
</body>
</html>
```

The result is as follows:



Solution B: Upload with a form

HTML form supports uploading with a lower browser version (e.g., IE8). The current solution uses the Post Object API. The directions are described as follows:

1. Obtain the bucket information by taking the steps in Prerequisites.
2. Create a `test.html` file. Then, modify the values `Bucket` and `Region` in the following code and copy the code to 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.

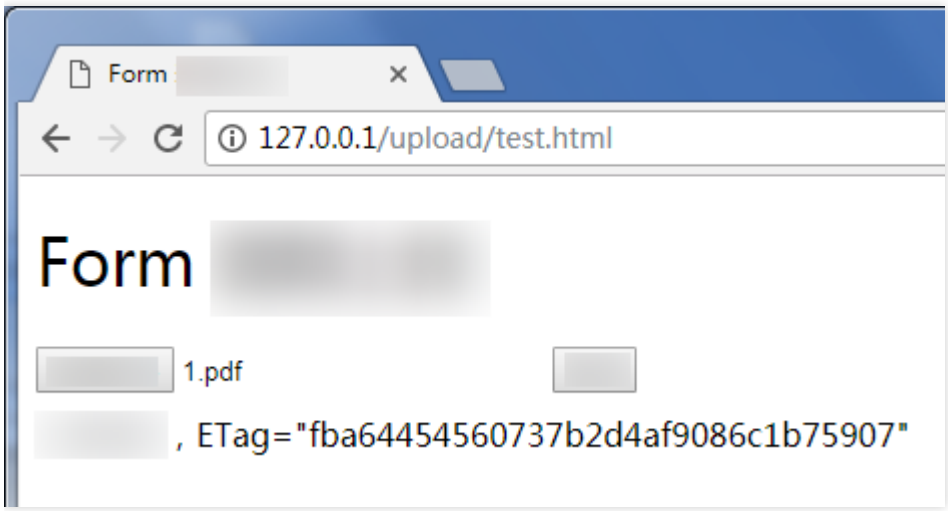
```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Simple Upload with a Form</title>
<style>h1, h2 {font-weight: normal;}#msg {margin-top:10px;}</style>
</head>
<body>
```

```
<h1>Simple Upload with a Form (IE8-Compatible)</h1>
<div>The browser should at least be IE6 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="Signature" name="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=
"">
<!-- Put the file field at the end of the form (in case the file content is too l
ong) to avoid affecting the signature field 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 src="https://unpkg.com/cos-js-sdk-v5/demo/common/cos-auth.min.js"></scrip
t>
<script>
(function () {
// Parameters used for the request
var Bucket = 'examplebucket-1250000000';
var Region = 'ap-guangzhou';
var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/'; // T
he prefix is used to concatenate the request URL. Use the default bucket endpoin
t.
var form = document.getElementById('form');
form.action = prefix;
// URL-encode more characters.
var camSafeUrlEncode = function (str) {
return encodeURIComponent(str)
.replace(/!/g, '%21')
.replace(/'/g, '%27')
.replace(/\(/g, '%28')
.replace(/\)/g, '%29')
.replace(/\*/g, '%2A');
};
// Calculate the signature.
var getAuthorization = function (options, callback) {
// var url = 'http://127.0.0.1:3000/sts' +
var url = '../server/sts.php';
```

```
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.onreadystatechange = function (e) {
if (xhr.readyState === 4) {
if (/^2\d\d$/.test('' + xhr.status)) {
var credentials;
try {
credentials = (new Function('return ' + xhr.responseText))().credentials;
} catch (e) {}
if (credentials) {
callback(null, {
SecurityToken: credentials.sessionToken,
Authorization: CosAuth({
SecretId: credentials.tmpSecretId,
SecretKey: credentials.tmpSecretKey,
Method: options.Method,
Pathname: options.Pathname,
})
});
} else {
console.error(xhr.responseText);
callback('Signature obtaining error');
}
} else {
callback('Signature obtaining error');
}
}
};
xhr.send();
};
// Listen on whether the upload is completed.
var Key;
var submitTarget = document.getElementById('submitTarget');
var showMessage = function (err, data) {
console.log(err || data);
document.getElementById('msg').innerText = err ? err : ('Upload successful, ETag
=' + data.ETag);
};
submitTarget.onload = function () {
var search;
try {
search = submitTarget.contentWindow.location.search.substr(1);
} catch (e) {
showMessage('File ' + Key + ' Upload failed');
}
if (search) {
var items = search.split('&');
```

```
var i, arr, 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 {
}
};
// Initiate an upload.
document.getElementById('submitBtn').onclick = function (e) {
var filePath = document.getElementById('fileSelector').value;
if (!filePath) {
document.getElementById('msg').innerText = 'File to upload not selected';
return;
}
Key = 'dir/' + filePath.match(/[\\\/]?([^\\\/]+)$/)[1]; // File directory and fil
ename
getAuthorization({Method: 'POST', Pathname: '/'}, function (err, AuthData) {
// Place an empty "empty.html" file in the current directory to be redirected bac
k when the upload is completed with the API.
document.getElementById('success_action_redirect').value = location.href.substr(0
, location.href.lastIndexOf('/') + 1) + 'empty.html';
document.getElementById('key').value = Key;
document.getElementById('Signature').value = AuthData.Authorization;
document.getElementById('x-cos-security-token').value = AuthData.SecurityToken ||
'';
form.submit();
});
};
})();
</script>
</body>
</html>
```

The result is shown as follows:



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：2022-11-30 14:59:51

Overview

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

> Note：
> The content of this document is based on the XML edition of APIs.

Prerequisites

1. Log in to the COS console and create a bucket with `BucketName` (bucket name) and `Region` (region name) specified. For more information, see Creating Buckets.
2. Log in to the CAM console and obtain your project's SecretId and SecretKey on the API key management page.

Directions

1. Configuring the WeChat Mini Program Allowlist

To request COS in your WeChat Mini Program, you need to log in to the WeChat 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.

For both methods, you need to configure COS domain names in corresponding allowlists. There are two forms of allowed domain names.

- If only one bucket is used, you can configure the bucket domain name as the allowed domain name, such as `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com`.

- If multiple buckets are used, you can choose to use suffixed COS requests by placing the buckets in the pathname. In this case, you need to configure the region domain name as the allowed domain name, such as `cos.ap-guangzhou.myqcloud.com` . For details, see the comments in the following code.

2. Getting Temporary Key and Calculating Signature

For security reasons, the signature uses a temporary key. For details on how to build a temporary key service on the server, see PHP Sample and Nodejs Sample.
If you use other languages or want to implement it on your own, follow the steps below:
(1) Obtain a temporary key from the server. The server first obtains the tmpSecretId, tmpSecretKey, and sessionToken of the temporary key from the STS service using the SecretId and SecretKey of the fixed key. For more information, see Temporary Key Generation and Usage Guidelines or cos-sts-sdk.

> Note：
> Based on whether the PUT or POST request is used, an action that "name/cos:PutObject" or "name/cos:PostObject" is allowed needs to be added to the policy action of the STS.

(2) The frontend calculates the signature based on the tmpSecretId, tmpSecretKey, method, and pathname. For more information about signature calculation, see cos-auth.js. If required by the actual business, the signature can also be calculated in the backend.
(3) Based on the used request, process the calculated signature and sessionToken as follows:

- For the POST request, put them in the `Signature` and `x-cos-security-token` fields of `formData` and send the upload request to COS API.
- For the PUT request, put them in the `Signature` and `x-cos-security-token` fields of `headers` and send the upload request to COS API.

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

3. Suffix Request

The general request format of COS API is similar to `POST http://examplebucket-1250000000.cos.ap-beijing.myqcloud.com/` . The requested domain name is a bucket domain name. In this way, if you use more than one bucket in the WeChat Mini Program, you need to configure the bucket domain name as an allowed domain name. The solution is as follows:

COS provides a suffix request format `POST http://cos.ap-beijing.myqcloud.com/examplebucket-1250000000/` . The requested domain name is a region domain name, and the bucket name is placed in the requested path. If you use multiple buckets in the same region in the WeChat Mini Program, you need to configure only one domain name `cos.ap-beijing.myqcloud.com` as an allowed domain name.

Note that, for the suffix request format, the path used during signing must be prefixed with the bucket name, for example, `/examplebucket-1250000000/` .

4. Sample Code for Direct Upload

The following code lists examples of both PUT Object API (recommended) and POST Object API. The operation guide is as follows:

```
var CosAuth = require('./cos-auth'); // cos-auth.js is referenced, and the downlo
ad address is https://unpkg.com/cos-js-sdk-v5/demo/common/cos-auth.min.js.
var Bucket = 'examplebucket-1250000000';
var Region = 'ap-shanghai';
var ForcePathStyle = false; // Whether to use a suffix, which involves signature
calculation and domain name allowlist configuration. For details about suffix req
uests, see the description above.
var uploadFile = function () {
// Parameters used for the request
var prefix = 'https://' + Bucket + '.cos.' + Region + '.myqcloud.com/';
if (ForcePathStyle) {
// The domain name used by the suffix request during signing is the region domain
name, not the bucket domain name. For more information, see "3. Suffix Request" d
escribed above.
prefix = 'https://cos.' + Region + '.myqcloud.com/' + Bucket + '/';
}
// URL-encode more characters.
var camSafeUrlEncode = function (str) {
return encodeURIComponent(str)
.replace(/!/g, '%21')
.replace(/'/g, '%27')
.replace(/\(/g, '%28')
.replace(/\)/g, '%29')
.replace(/\*/g, '%2A');
};
// Get the temporary key
var stsCache;
var getCredentials = function (callback) {
if (stsCache && Date.now() / 1000 + 30 < stsCache.expiredTime) {
callback(data.credentials);
return;
}
wx.request({
```

```
method: 'GET',
url: 'https://example.com/sts.php', // Server-side signature. For more informatio
n, see the instructions for getting the temporary key above.
dataType: 'json',
success: function (result) {
var data = result.data;
var credentials = data.credentials;
if (credentials) {
stsCache = data
} else {
wx.showModal({title: 'failed to get the temporary key', content: JSON.stringify(d
ata), showCancel: false});
}
callback(stsCache && stsCache.credentials);
},
error: function (err) {
wx.showModal({title: 'failed to get the temporary key', content: JSON.stringify(e
rr), showCancel: false});
}
});
};
// Calculate the signature.
var getAuthorization = function (options, callback) {
getCredentials(function (credentials) {
callback({
XCosSecurityToken: credentials.sessionToken,
Authorization: CosAuth({
SecretId: credentials.tmpSecretId,
SecretKey: credentials.tmpSecretKey,
Method: options.Method,
Pathname: options.Pathname,
})
});
});
};
// Upload the file through POST.
var postFile = function (filePath) {
var Key = filePath.substr(filePath.lastIndexOf('/') + 1); // The name of the file
to be uploaded is specified here.
var signPathname = '/'; // For the PostObject API, the Key is placed in the Body
for transmission, so the request path and signature path are /.
if (ForcePathStyle) {
// The path used by the suffix request during signing must contain the bucket nam
e. For more information, see "3. Suffix Request" described above.
signPathname = '/' + Bucket + '/';
}
getAuthorization({Method: 'POST', Pathname: signPathname}, function (AuthData) {
```

```
var requestTask = wx.uploadFile({
url: prefix,
name: 'file',
filePath: filePath,
formData: {
'key': Key,
'success_action_status': 200,
'Signature': AuthData.Authorization,
'x-cos-security-token': AuthData.XCosSecurityToken,
'Content-Type': '',
},
success: function (res) {
var url = prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/');
console.log(res.statusCode);
console.log(url);
if (/^2\d\d$/.test('' + res.statusCode)) {
wx.showModal({title: 'upload succeeded', content: url, showCancel: false});
} else {
wx.showModal({title: 'upload failed', content: JSON.stringify(res), showCancel: f
alse});
}
},
fail: function (res) {
wx.showModal({title: 'upload failed', content: JSON.stringify(res), showCancel: f
alse});
}
});
requestTask.onProgressUpdate(function (res) {
console.log('progress:', res);
});
});
};
// Upload the file through PUT.
var putFile = function (filePath) {
var Key = filePath.substr(filePath.lastIndexOf('/') + 1); // The name of the file
to be uploaded is specified here.
var signPathname = '/'; // For the PostObject API, the Key is placed in the Body
for transmission, so the request path and signature path are /.
if (ForcePathStyle) {
// The path used by the suffix request during signing must contain the bucket nam
e. For more information, see "3. Suffix Request" described above.
signPathname = '/' + Bucket + '/' + Key;
}
getAuthorization({Method: 'PUT', Pathname: signPathname}, function (AuthData) {
// The PUT request needs to read the file content from the temporary path of the
file.
var wxfs = wx.getFileSystemManager();
```

```
wxfs.readFile({
filePath: filePath,
success: function (fileRes) {
var requestTask = wx.request({
url: prefix + signPathname.substr(signPathname.lastIndexOf('/') + 1),
method: 'PUT',
header: {
'Authorization': AuthData.Authorization,
'x-cos-security-token': AuthData.XCosSecurityToken,
},
data: fileRes.data,
success: function success(res) {
var url = prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/');
if (res.statusCode === 200) {
wx.showModal({title: 'upload succeeded', content: url, showCancel: false});
} else {
wx.showModal({title: 'upload failed', content: JSON.stringify(res), showCancel: f
alse});
}
console.log(res.statusCode);
console.log(url);
},
fail: function fail(res) {
wx.showModal({title: 'upload failed', content: JSON.stringify(res), showCancel: f
alse});
},
});
requestTask.onProgressUpdate(function (res) {
console.log('in progress:', res);
});
},
});
});
};
// Select the file
wx.chooseImage({
count: 1, // Default value: 9
sizeType: ['original'], // You can specify whether the image is original or compr
essed. Original by default
sourceType: ['album', 'camera'], // You can specify whether the source is an albu
m or camera. Both are included by default.
success: function (res) {
putFile(res.tempFiles[0].path); // PUT for the upload request, which is recommend
ed.
// postFile(res.tempFiles[0].path); // POST for the upload request.
}
```

```
})
};
```

References

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：2021-11-16 11:45:09

Overview

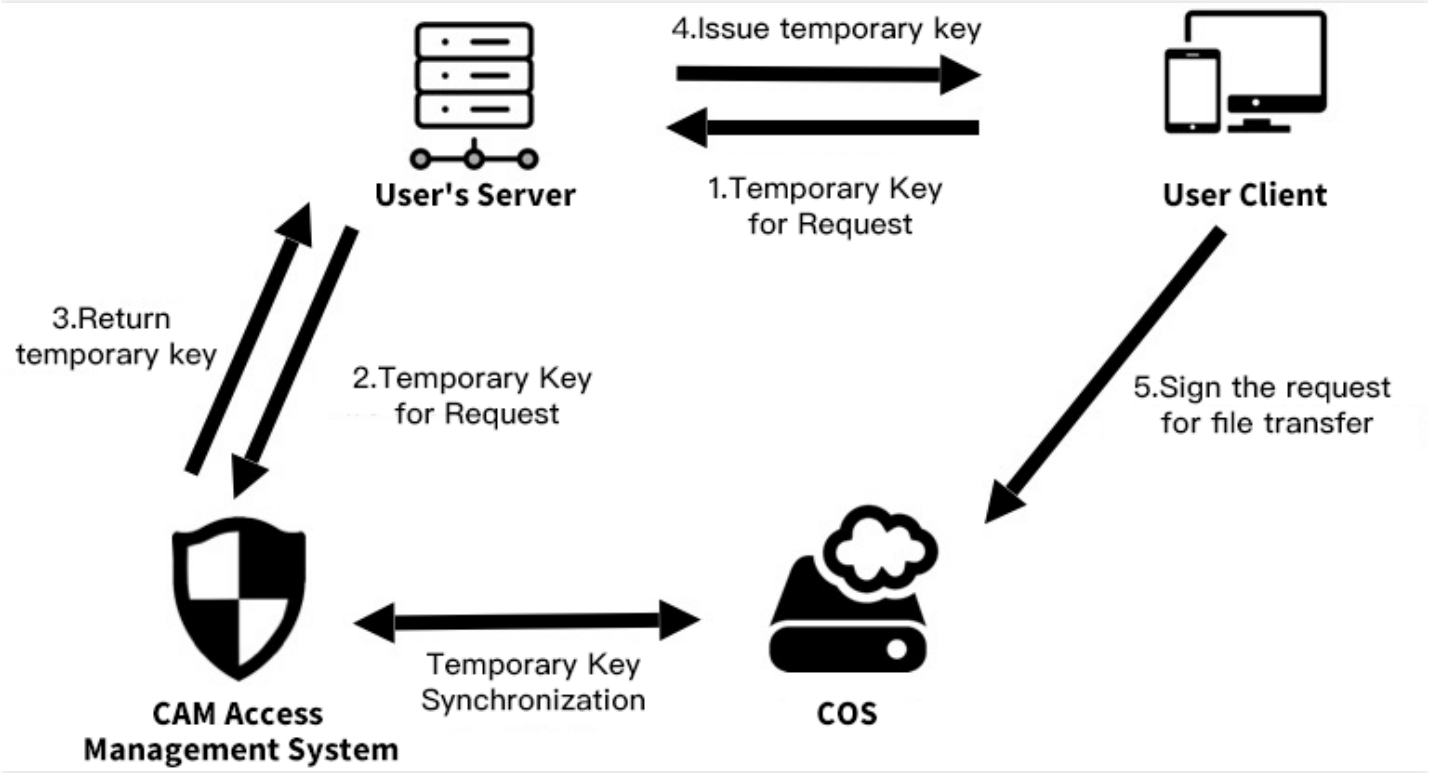
This document describes how to achieve direct file upload with Tencent Cloud COS for your mobile applications. COS makes it easy for you to store files and data as you only need to generate and manage the access key on your server.

Architecture

For client applications, storing permanent keys in the code increases the risk that your credentials could leak and makes it difficult to control access permissions. We recommend that your applications use temporary keys with a specified validity period to access your COS resources to prevent credential leakage.
The COS Mobile SDK (Android/IOS) allows you to apply temporary keys on COS access request authorization. You only need to set up the temporary key service on the backend to authorize the requests.

Workflow

The diagram below illustrates how to enable CAM for the COS:



Notes:

- User's client: user's mobile App.
- COS: Cloud Object Storage, which stores the data uploaded from the App.
- CAM: Cloud Access Management, which is used to generate temporary keys for COS.
- User's server: user's backend server, which is used to obtain the temporary keys and return them to the App.

Prerequisites

1. Create a bucket.
   Create a bucket using the COS Console. Depending on your needs, set your bucket permission to private read/write, or public read and private write. For detailed directions, see Creating Buckets and Setting Access Permission.
2. Obtain the permanent key.
   Temporary keys are generated with the permanent key. Get the SecretId and SecretKey on the API Key Management page. Get the APPID in the Account Center.

Directions

Setting up temporary key service

For security purposes, we recommend you use temporary keys to calculate a signature. To create and use temporary keys, you need to set up the temporary key service on your server and an API for your application. For more information, see Temporary Key Generation and Usage Guide.

> Note：
> In official deployment, add a layer of permission check of your website on the server side.

Defining permissions

We recommended that you define a scope for the permission granted to temporary keys through Policy based on the principle of "least privilege". A key that has all permissions to read and write the data increases the risk that your other data could leak. For more information, see Temporary Key Generation and Usage Guide.

Integrating SDK with authorization service

Android

After the temporary key service is set up, you integrate the SDK with the authorization service. Instead of you managing the temporary keys, the SDK controls the number of concurrent requests to process, cache the valid keys on your local devices, and requests new keys to replace the expired ones.

Authorization with standard response body

If you use the JSON data obtained from the STS SDK as a response body for the temporary key service (similar to cossign), you can create an authorization class in the COS SDK using the following code:

```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;

Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;
/**
* Get the authorization service URL
*/
URL url = null; // URL of the backend authorization service
try{
url = new URL("your_auth_server_url");
} catch (MalformedURLException e) {
e.printStackTrace();
}
/**
* Initialize the {@link QCloudCredentialProvider} object to provide a temporary k
ey to the SDK.
*/
QCloudCredentialProvider credentialProvider = new SessionCredentialProvider(new H
ttpRequest.Builder<String>()
.url(url)
.method("GET")
.build());

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig, cre
dentialProvider);
```

Note：
Because this method requires that the signature start time matches the local time on your mobile phone. A large difference (more than 10 minutes) between the time on your phone and the correct local time may cause a

signature error. In this case, we recommend you use the custom response body for authorization as described below:

Authorization with custom response body

For higher flexibility, you can inherit the `BasicLifecycleCredentialProvider` class and implement its `fetchNewCredentials()` for custom configurations. For example, you can customize the HTTP response body for the temporary key service to return the server time to the device as the signature start time so as to avoid signature error caused by big device time difference. You can also choose to use other protocols for the communication between the end device and the service side.

Define a `MyCredentialProvider` class first:

```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;

public class MyCredentialProvider extends BasicLifecycleCredentialProvider {
@Override
protected QCloudLifecycleCredentials fetchNewCredentials() throws QCloudClientExc
eption {

// First, get the response containing a signature from your temporary key server.
....

// Then, parse the response to get the key.
String tmpSecretId = ...;
String tmpSecretKey = ...;
String sessionToken = ...;
long expiredTime = ...;

// Return server time as the start time of the signature.
long beginTime = ...;

// todo something you want

// Finally return the temporary key info.
return new SessionQCloudCredentials(tmpSecretId, tmpSecretKey, sessionToken, begi
nTime, expiredTime);
}
}
```

Authorize the request with your custom `MyCredentialProvider` instance:

```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;
Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;

/**
* Initialize the {@link QCloudCredentialProvider} to provide a temporary key to t
he SDK.
*/
QCloudCredentialProvider credentialProvider = new MyCredentialProvider();
CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig, cre
dentialProvider);
```

For more information on how files are uploaded and downloaded between an Android device and COS, see Getting Started with Android SDK.

iOS

We provide QCloudCredentailFenceQueue to make it easier for you to obtain and manage temporary signatures. With QCloudCredentailFenceQueue, a request for signature will be processed only after the signature process is completed.

To use QCloudCredentailFenceQueue, you need to first create an instance.

```
//AppDelegate.m
//AppDelegate must follow QCloudCredentailFenceQueueDelegate protocol
//
- (BOOL)application:(UIApplication * )application didFinishLaunchingWithOptions:(
NSDictionary * )launchOptions {
// init step
self.credentialFenceQueue = [QCloudCredentailFenceQueue new];
self.credentialFenceQueue.delegate = self;
return YES;
}
```

Next, to call the QCloudCredentailFenceQueue class, you must follow QCloudCredentailFenceQueueDelegate and implement the method defined in the protocol:

```
- (void) fenceQueue:(QCloudCredentailFenceQueue * )queue requestCreatorWithContin
ue:(QCloudCredentailFenceQueueContinue)continueBlock
```

When you obtain the signature using QCloudCredentailFenceQueue, all the requests in the SDK that need signatures will not be performed until the parameters required for the signature are obtained via the method defined by the protocol and a valid signature is generated. See the example below:

```
- (void)fenceQueue:(QCloudCredentailFenceQueue *)queue requestCreatorWithContinu
e:(QCloudCredentailFenceQueueContinue)continueBlock {
QCloudHTTPRequest* request = [QCloudHTTPRequest new];
request.requestData.serverURL = @“your sign service url”;//Request URL

[request setConfigureBlock:^(QCloudRequestSerializer *requestSerializer, QCloudRe
sponseSerializer *responseSerializer) {
requestSerializer.serializerBlocks = @[QCloudURLFuseWithURLEncodeParamters];
responseSerializer.serializerBlocks = @[QCloudAcceptRespnseCodeBlock([NSSet setWi
thObjects:@(200), nil],nil),//An error is returned when the return code is other
than 200.
QCloudResponseJSONSerilizerBlock];//Return element parsed in JSON format
}];

[request setFinishBlock:^(id response, NSError *error) {
if (error) {
error = [NSError errorWithDomain:@"com.tac.test" code:-1111 userInfo:@{NSLocalize
dDescriptionKey:@"No temporary key is obtained."}];
continueBlock(nil, error);
} else {
QCloudCredential* crendential = [[QCloudCredential alloc] init];
crendential.secretID = response[@"data"][@"credentials"][@"tmpSecretId"];
crendential.secretKey = response[@"data"][@"credentials"][@"tmpSecretKey"];
credential.startDate =[NSDate dateWithTimeIntervalSince1970:@"Returned server tim
e"]
crendential.experationDate = [NSDate dateWithTimeIntervalSinceNow:[response[@"dat
a"][@"expiredTime"] intValue]];
crendential.token = response[@"data"][@"credentials"][@"sessionToken"];;
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc] initW
ithCredential:crendential];
continueBlock(creator, nil);

}
}];
[[QCloudHTTPSessionManager shareClient] performRequest:request];
}
```

For more information on how files are uploaded and downloaded between an iOS device and COS, see Getting Started With iOS SDK.

Demo code

Android

Download the Demo by clicking here, or scanning QR code with your Android mobile browser:

iOS

For complete sample project for iOS, see COS iOS Demo.

Modify the file `QCloudCOSXMLDemo/QCloudCOSXMLDemo/key.json` by adding your APPID, secretID, and secretKey, and run this command:

```
pod install
```

> Note：
> Get your APPID, secretID, and secretKey from the API Key Management page.

After executing the command, open QCloudCOSXMLDemo.xcworkspace to view the Demo.

uni-app Direct Upload Practice

Last updated：2022-08-23 15:06:25

Overview

This document describes how to use simple code to upload files to a COS bucket directly via uni-app without using an SDK.

> Note：
> This document is based on the XML API PostObject.

Solution

Directions

1. Select a file in the frontend, and the frontend sends the file extension to the server.
2. The server generates a random COS file path with time according to the file extension, calculates the corresponding PostObject policy signature, and returns the URL and signing information to the frontend.
3. The frontend calls the PostObject API to upload the file to COS.

Strengths

- Permission security: The PostObject policy signature effectively limits the security permission scope and can only be used to upload to a specified file path.
- Path security: The server determines the random COS file path, which effectively avoids the problem of overwriting existing files and security risks.
- Multi-terminal compatibility: The file selection and upload APIs provided by uni-app can be used to make the same code compatible with multiple terminals (web, mini programs, or apps).

Prerequisites

1. Log in to the COS console and create a bucket to obtain the `Bucket` (bucket name) and `Region` (region name). For more information, please see Creating Buckets.
2. Log in to the CAM console and obtain the `SecretId` and `SecretKey` of your project.

3. Go to the details page of the newly created bucket, choose **Security Management** > **CORS (Cross-Origin Resource Sharing)**, and click **Add a Rule**. A configuration example is shown in the following figure. For more information, see Setting Cross-Origin Access.

Directions

> Note：
> Before formal deployment, it is recommended that you add a permission verification step for your website itself on the server side.

Frontend upload

1. Implement a server API to generate random file paths, calculate signatures, and return them to the frontend. For implementation details, see the post-policy example.

2. Use HBuilderX's default template to create a uni-app app.
   The created app is a Vue project.

3. Copy the following code to replace the content of the `pages/index/index.vue` file and modify the `post-policy` API call link to make it point to your server address (the server API implemented in step 1).

```
<template>
<view class="content">
<button type="default" @click="selectUpload">Select file upload</button>
<image v-if="fileUrl" class="image" :src="fileUrl"></image>
</view>
</template>
<script>
export default {
data() {
return {
title: 'Hello',
fileUrl: ''
};
},
onLoad() {
},
methods: {
selectUpload() {
```

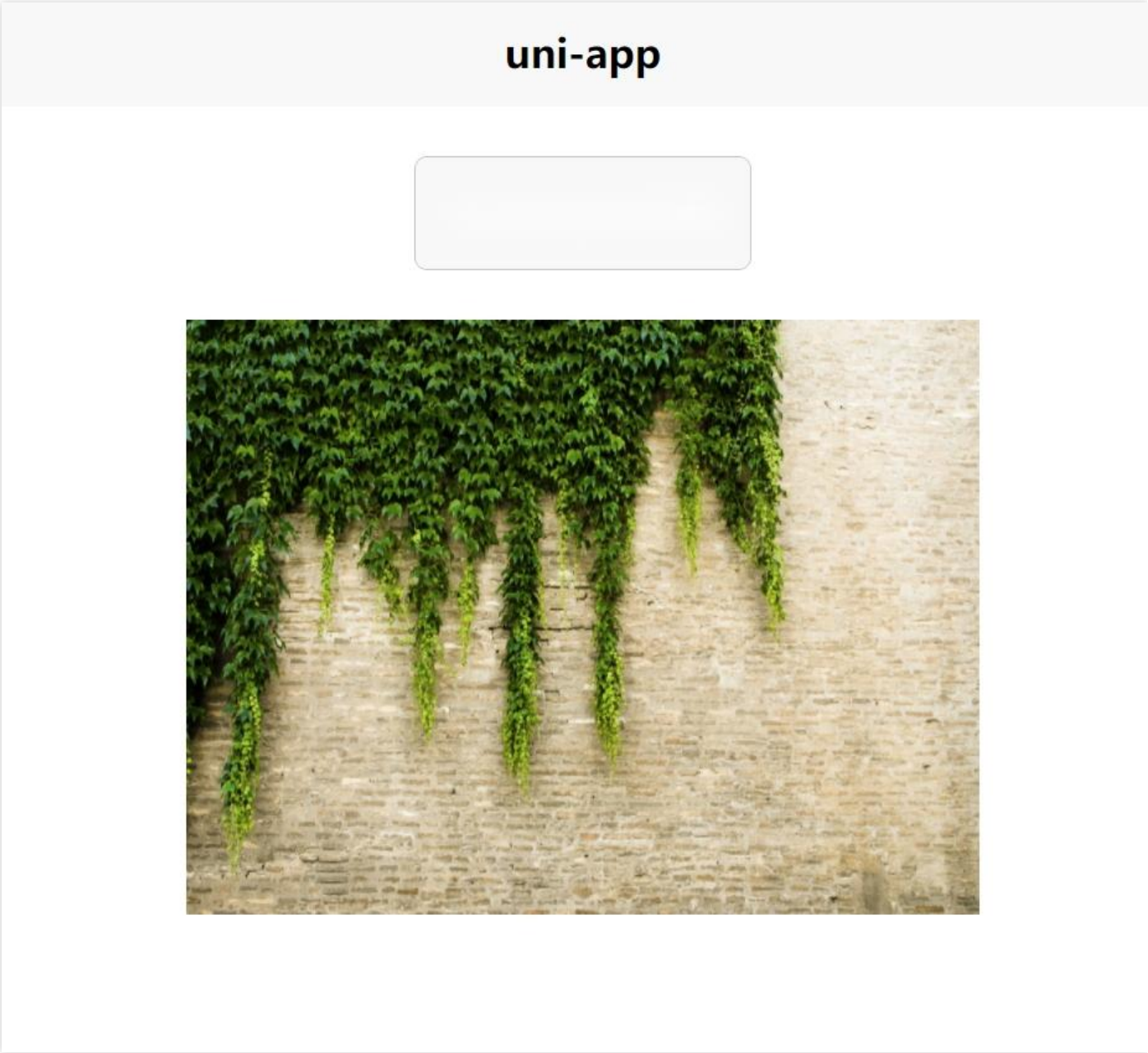
```
var vm = this;
// URL-encode more characters.
var camSafeUrlEncode = function (str) {
return encodeURIComponent(str)
.replace(/!/g, '%21')
.replace(/'/g, '%27')
.replace(/\(/g, '%28')
.replace(/\)/g, '%29')
.replace(/\*/g, '%2A');
};
// Get the upload path and credential
var getUploadInfo = function (extName, callback) {
// Pass in the file extension to enable the backend to generate a random COS ob
ject path and return the upload domain name and the policy signature required b
y the `PostObject` API.
// Refer to the server example at https://github.com/tencentyun/cos-demo/tree/m
ain/server/post-policy
uni.request({
url: 'http://127.0.0.1:3000/post-policy?ext=' + extName,
success: (res) => {
callback && callback(null, res.data.data);
},
error(err) {
callback && callback(err);
},
});
};
// Initiate an upload request, and the upload uses the `PostObject` API and pol
icy signature protection.
// API documentation: https://intl.cloud.tencent.com.cn/document/product/436/14
690#.E7.AD.BE.E5.90.8D.E4.BF.9D.E6.8A.A4
var uploadFile = function (opt, callback) {
var formData = {
key: opt.cosKey,
policy: opt.policy, // Base64 policy string
success_action_status: 200,
'q-sign-algorithm': opt.qSignAlgorithm,
'q-ak': opt.qAk,
'q-key-time': opt.qKeyTime,
'q-signature': opt.qSignature,
};
// If the server uses a temporary key for calculation, you need to pass in `x-c
os-security-token`.
if (opt.securityToken) formData['x-cos-security-token'] = opt.securityToken;
uni.uploadFile({
url: 'https://' + opt.cosHost, // This is only an example, not the real API add
ress.
```

```
filePath: opt.filePath,
name: 'file',
formData: formData,
success: (res) => {
if (![200, 204].includes(res.statusCode)) return callback && callback(res);
var fileUrl = 'https://' + opt.cosHost + '/' + camSafeUrlEncode(opt.cosKey).rep
lace(/%2F/g, '/');
callback && callback(null, fileUrl);
},
error(err) {
callback && callback(err);
},
});
};
// Select the file
uni.chooseImage({
success: (chooseImageRes) => {
var file = chooseImageRes.tempFiles[0];
if (!file) return;
// Get the path of the local file to upload
var filePath = chooseImageRes.tempFilePaths[0];
// Get the extension of the file to upload and then the backend can generate a
random COS path
var fileName = file.name;
var lastIndex = fileName.lastIndexOf('.');
var extName = lastIndex > -1 ? fileName.slice(lastIndex + 1) : '';
// Get the domain name, path, and credential for pre-upload
getUploadInfo(extName, function (err, info) {
// Upload the file
info.filePath = filePath;
uploadFile(info, function (err, fileUrl) {
vm.fileUrl = fileUrl;
});
});
}
});
},
}
}
</script>
<style>
.content {
padding: 20px 0;
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
```

```
}
.image {
margin-top: 20px;
margin-left: auto;
margin-right: auto;
}
</style>
```

4. In HBuilderX, choose **Run** > **Run to browser** > **Chrome**. Then you can select files to upload in the browser. The execution result is as shown below:

- Direct upload effect:



Demo Code Address

Demo download address: upload-demo

Domain Name Management Practice
Supporting HTTPS for Custom Endpoints

Last updated：2022-07-08 11:50:53

Overview

You can access the objects under a bucket using your own endpoint (the custom endpoint, for example, `test.cos.com`). Detailed directions are as follows:

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

Directions

Enabling CDN Acceleration

Step 1. Bind a custom domain name

Bind the bucket to your own endpoint and enable CDN acceleration. For detailed directions, please see Enabling Custom Accelerated Domain Name.

Step 2. Perform HTTPS configuration

You can configure HTTPS access in the CDN console. For detailed directions, please see HTTPS Configuration Guide.

Disabling CDN Acceleration

This section uses an example to describe how to support HTTPS access in COS by configuring custom endpoints through a reverse proxy (with CDN acceleration disabled). In this example, we use the custom endpoint `https://test.cos.com` to directly access the `testhttps-1250000000` bucket in the Guangzhou region with CDN acceleration disabled. The specific steps are as follows:

Step 1. Bind a custom domain name

HTTPS certificate hosting for custom origin server domain names of COS is supported in public cloud regions in the Chinese mainland and in Singapore. You can bind the certificate to the added custom origin server domain names via the console. For details, see Method 1. If no HTTPS certificate is available for your domain name, click Apply for Free Certificate.

This feature is currently not supported in other regions. To use an HTTPS certificate, see Method 2.

- Method 1: Bind a custom origin server domain name via the COS console
  Bind the `testhttps-1250000000` bucket to the `https://test.cos.com` domain and disable CDN acceleration. For detailed directions, please see Enabling Custom Accelerated Domain Name.

- Method 2: Configure a reverse proxy for the domain name
  Configure a reverse proxy for the `https://test.cos.com` endpoint on the server, as shown below (the Nginx configuration is for reference only):

```
server {
listen 443;
server_name test.cos.com ;
ssl on;
ssl_certificate /usr/local/nginx/conf/server.crt;
ssl_certificate_key /usr/local/nginx/conf/server.key;
error_log logs/test.cos.com.error_log;
access_log logs/test.cos.com.access_log;
location / {
root /data/www/;
proxy_pass http://testhttps-1250000000.cos.ap-guangzhou.myqcloud.com; // Config
ure the default download domain for the bucket.
}
}
```

`Server.crt;` and `server.key` are HTTPS certificates for your own (custom) domain. If no HTTPS certificate is available for your domain, you can apply for one at Tencent Cloud SSL Certificate Service.
If no certificate is available, the following configuration information can be deleted, but an alarm will occur during access. Click Continue to access the bucket:

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

Step 2. Resolve the domain name at a server

Resolve your endpoint at your endpoint's DNS provider.

Step 3. Perform advanced configurations

- **Opening the web page in a browser directly**
  After configuring the custom endpoint to support HTTPS, you can download objects in the bucket using your

domain. If your business requires directly accessing web pages and images in a browser, you can use the static website feature. For detailed directions, please see Setting Up a Static Website.
After the configuration is completed, add the following code to the Nginx configuration file, restart Nginx, and refresh the browser cache.

```
proxy_set_header Host $http_host;
```

- **Configuring referer hotlink protection**
  Public buckets might be hotlinked. You can use the hotlink protection feature to set a referer allowlist to prevent malicious hotlinking as follows:

1. 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 the configuration, if you open the file directly, the error `errorcode: -46616` (error message: `not hit white refer`) will be reported. In this case, you can access the custom endpoint with a proxy to open the page.

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

Setting CORS

Last updated：2022-09-01 16:20:54

One-Origin Policy

The one-origin policy is a key security mechanism for isolating potentially malicious files. It restricts the way files/scripts loaded from one origin interacts with resources from another origin. Resources with the same protocol, domain name (or IP), and port are considered to belong to the same origin. Scripts on one origin only have permissions to read/write resources on the origin but cannot access resources on other origins.

Definition of one-origin resources

Webpages from a single origin should have the same protocol, domain name, and port (if specified). The following table shows how to test whether a webpage belongs to the same origin as `http://www.example.com/dir/page.html`:

URL	Result	Reason
`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

CORS

Cross-Origin Resource Sharing (CORS) is also known as cross-origin access. It allows web application servers to perform cross-origin access control to ensure secure cross-origin data transfer. Both the browser and server need to support this feature before you can use it. The feature is compatible with all browsers (for IE, IE 10 or later is required).

The CORS communication process is automatically completed by the browser without any manual intervention required. For developers, CORS communication and one-origin AJAX communication work in the same way and use the same code. Once the browser identifies an AJAX request for cross-origin access, it automatically adds additional header information. In some cases, an additional request is made, but you will not perceive it.

Therefore, the key to CORS communication lies in the server. As long as the server implements CORS APIs, cross-origin communication can be implemented.

CORS Use Cases

CORS is used when you are using a browser. This is because access permissions are controlled by the browser but not the server. Therefore, if you use other clients, you don't need to care about cross-origin access.

With CORS, you can use AJAX in a browser to directly access, upload, and download COS data without using your app server for data transfer. If your website adopts both COS and AJAX technologies, we recommend you use CORS for direct communication with COS.

COS Support for CORS

COS supports configuring CORS rules to allow or deny cross-origin requests as needed. CORS rules are configured at the bucket level.

COS authentication and whether a CORS request is allowed are independent of each other. In other words, CORS rules of COS are only used to decide whether to add CORS-related headers. It is up to the browser whether to block the request.

All object and multipart APIs of COS support CORS authentication.

> Note：
> When two webpages (`www.a.com` and `www.b.com`) running in the same browser request the same cross-origin resource at the same time, if the request from `www.a.com` arrives at the server first, the server will return the resource to the user of `www.a.com` with the `Access-Control-Allow-Origin` header. If `www.b.com` sends a request later, the browser will return the cached response of the last request to the user. In this case, the header content does not match the CORS-required content, so the `www.b.com` request will fail.

CORS Configuration Example

The following example shows how to configure CORS to get data from COS by using AJAX. The bucket permission used in the example is set to public. For a bucket with private access permission, a signature needs to be added in the request, while other configurations are the same.

The bucket used in the following example is named `corstest`, with the access permission of public read/private write.

Preparations

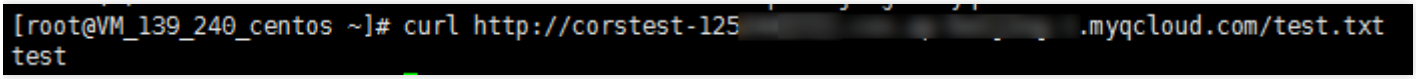
1. **Verify whether the file can be accessed.**

   Upload the `test.txt` file to `corstest`. The access address of this file is `http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt`.

   Access the text file by using curl, with the following address replaced with your file address:

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

If "test" (the file content) is returned, the file can be accessed normally.



2. **Access the file by using AJAX**

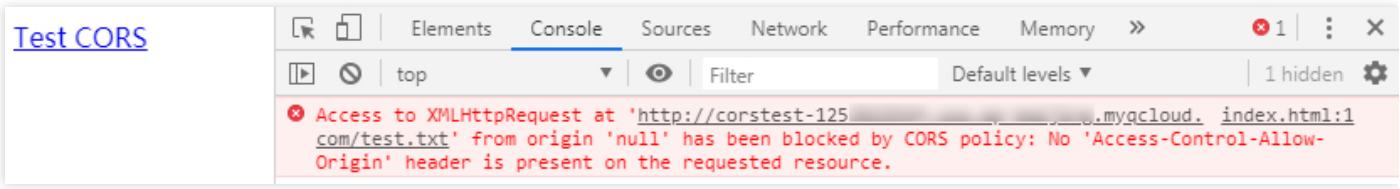
   You can access the `text.txt` file directly by using AJAX.

   (1) Copy the following code to a local HTML file and then open it with a browser. As no custom header is set, no preflight request is required.

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
</head>
<body>
<a href="javascript:test()">Test CORS</a>
<script>
function test() {
var url = 'http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt';
var xhr = new XMLHttpRequest();
xhr.open('HEAD', url);
xhr.onload = function () {
var headers = xhr.getAllResponseHeaders().replace(/\r\n/g, '\n');
alert('request success, CORS allow.\n' +
'url: ' + url + '\n' +
'status: ' + xhr.status + '\n' +
'headers:\n' + headers);
};
xhr.onerror = function () {
```

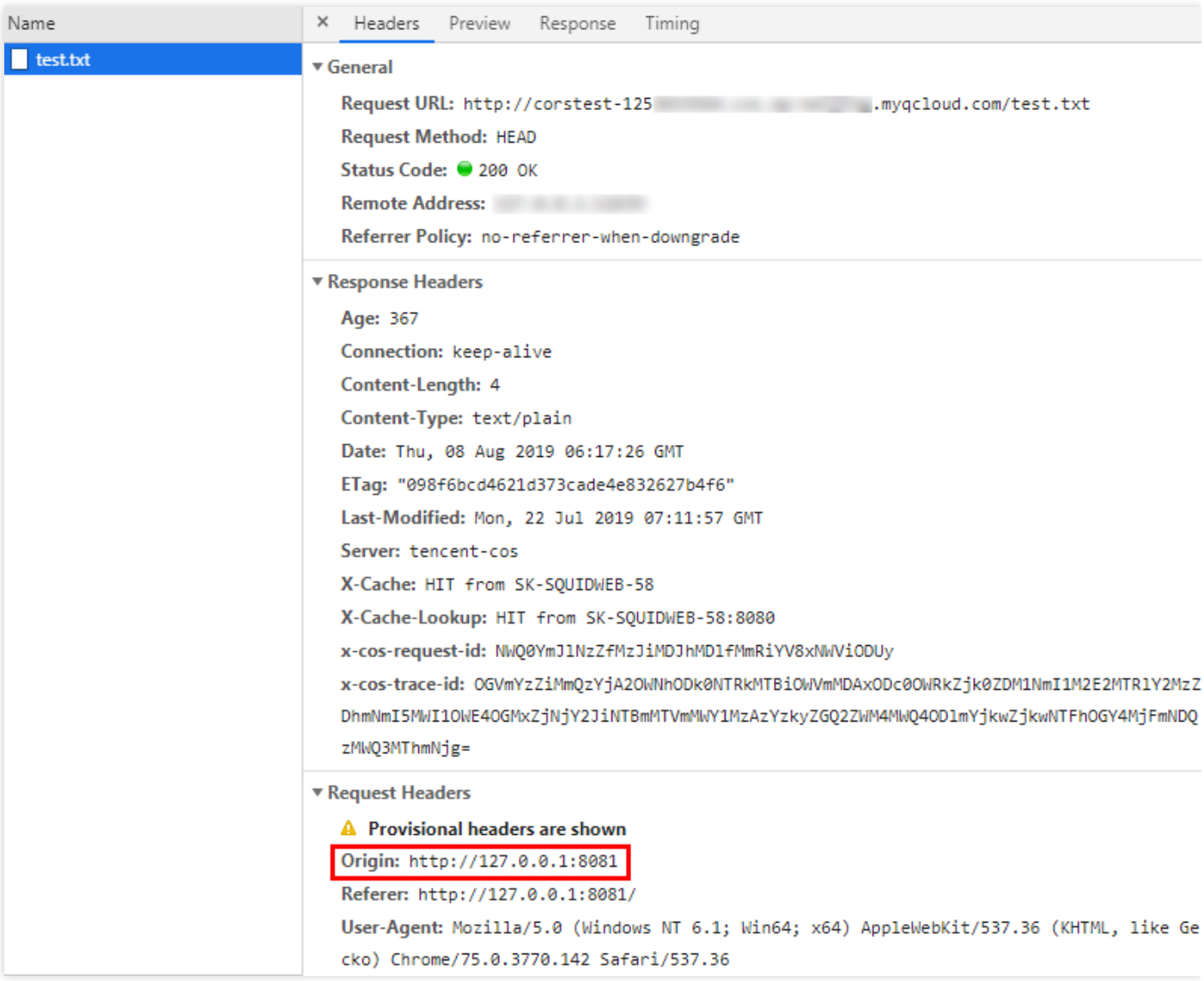
```
alert('request error, maybe CORS error.');
};
xhr.send();
}
</script>
</body>
</html>
```

(2) Open the HTML file in the browser and click **Test CORS** to send the request. The following error occurs with the message "Access denied. No Access-Control-Allow-Origin header is found". This is because CORS has not been configured on the server.



(3) When the access fails, go to the **Headers** page to find out the cause. You can see that the browser sent a

request with **Origin** specified, meaning it is a cross-origin request.



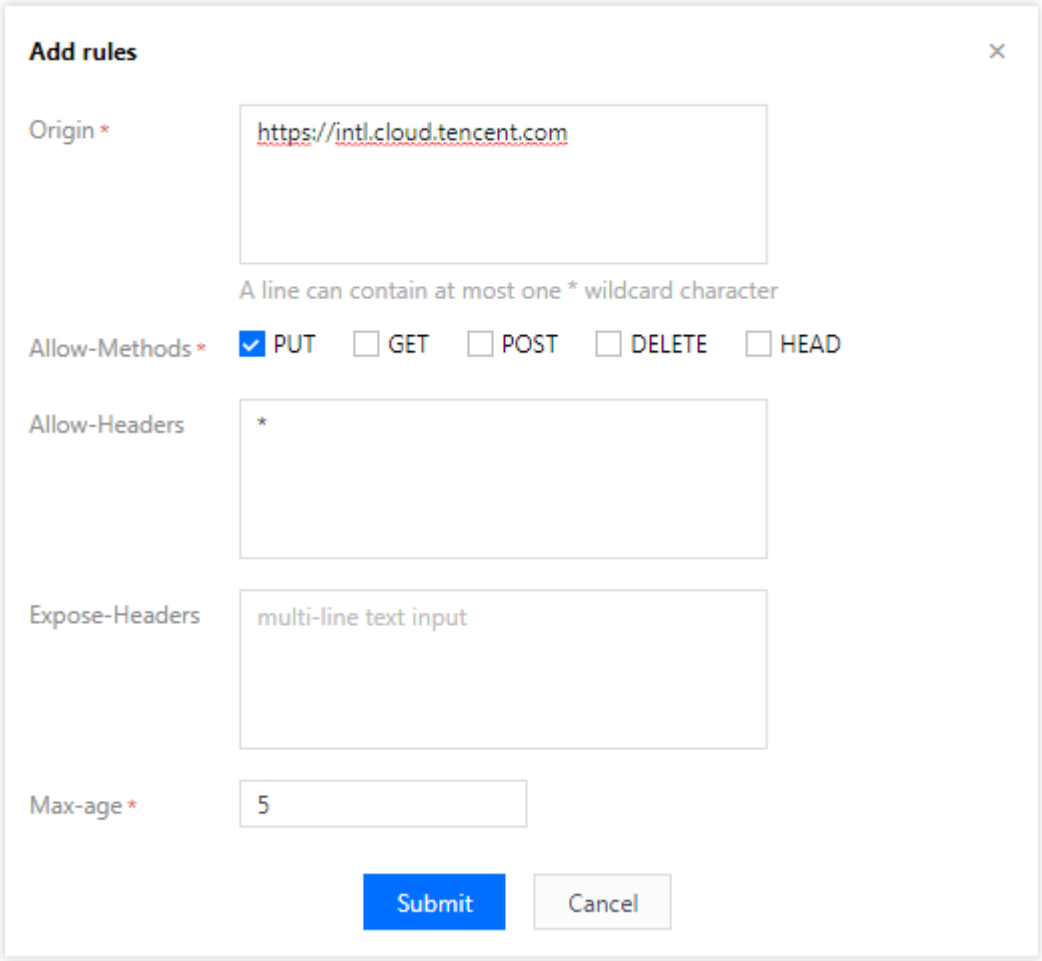
Note：

The webpage is set up on the server with the address `http://127.0.0.1:8081`. Therefore, the **Origin** is `http://127.0.0.1:8081`.

Configuring CORS

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

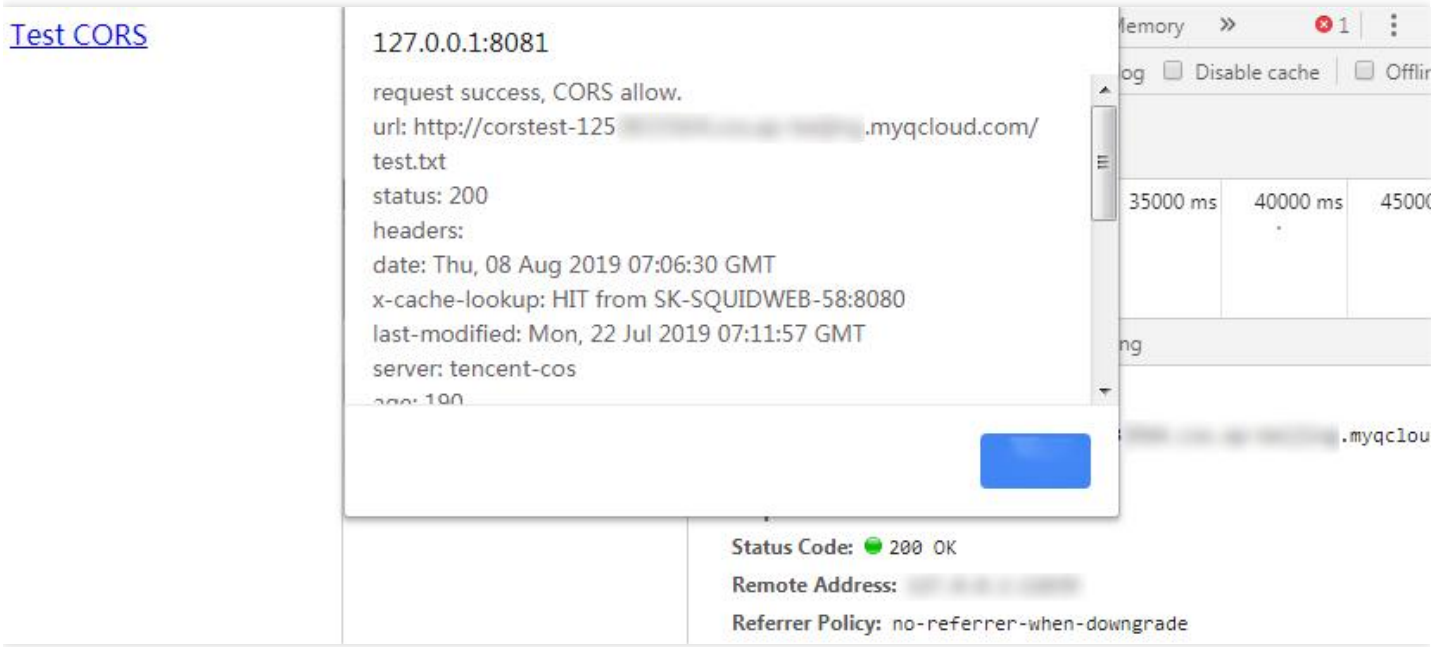
1. Log in to the COS console, click **Bucket List**, and click the name of the target bucket. Then, select the **Security Management** tab and find **CORS (Cross-Origin-Resource Sharing)** in the drop-down list.
2. Click **Add a Rule** to add the first rule with the following least restricted configuration:



Note：
The CORS configuration consists of multiple rules, which are matched individually from top to bottom. The first matched rule will be applied.

Verifying result

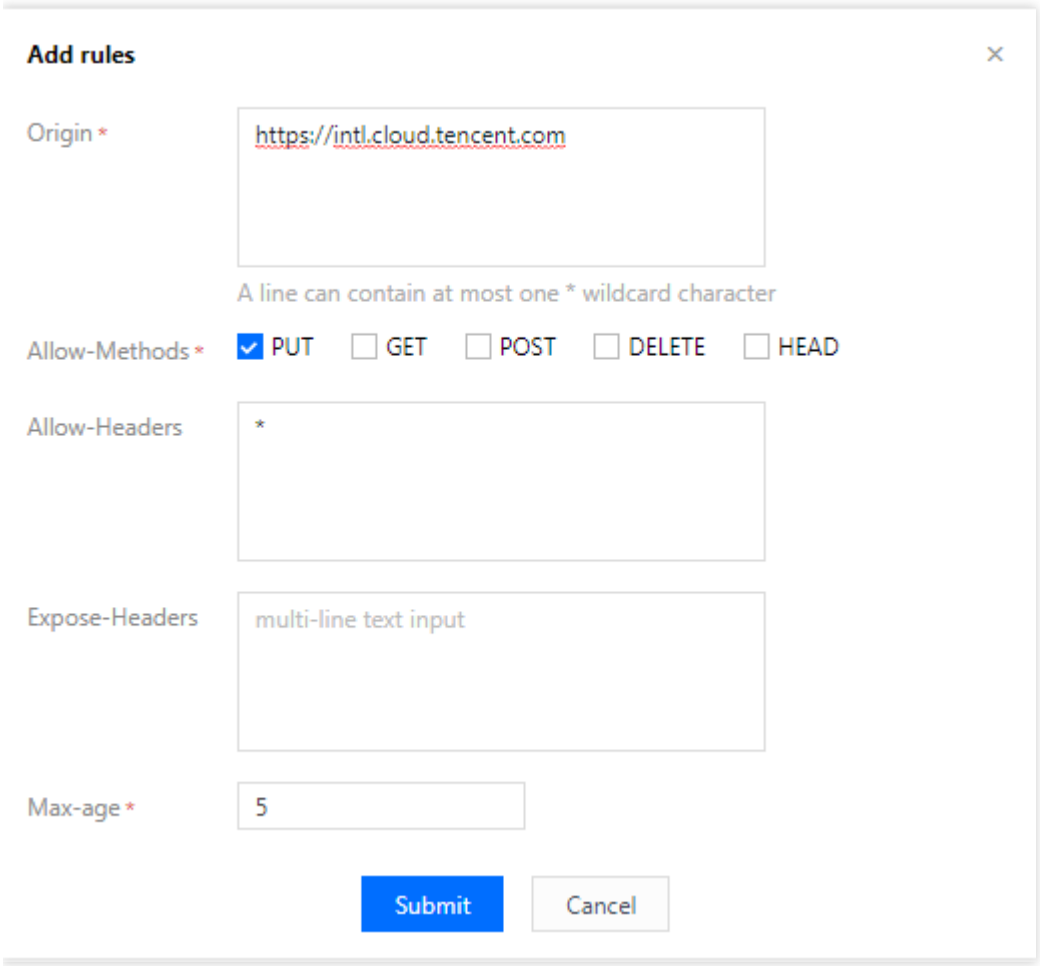
After the configuration is completed, try accessing the `test.txt` file again. If the result is as follows, the file can be accessed normally.



Troubleshooting and suggestions

To avoid problems related to cross-origin access, you can set the least restricted CORS rule as described above to allow all cross-origin requests. If an error still occurs under this configuration, the root cause may lie in other factors rather than CORS.

In addition to configuring the least restricted rule, you can also configure a more refined rule. For example, in this example, you can use the following most restricted configuration to ensure a successful match:



Therefore, for most scenarios, we recommend you use the most restricted configuration as needed to ensure security.

CORS Configuration Items

CORS configuration items are as follows:

Origin

This refers to the origin allowing cross-origin requests.

- Multiple domain names can be specified, with one domain name per line.
- Asterisk (*) is supported, meaning that all domain names are allowed. This is not recommended.
- A single specific domain name such as `http://www.abc.com` is supported.
- Second-level wildcard domain names such as `http://*.abc.com` are supported. Note that each line can contain only one `*` .
- Do not omit the protocol name HTTP or HTTPS. If the port is not the default port 80, the port should also be specified.

Allow-Methods

Enumerate one or multiple allowed cross-origin request methods.
Examples: GET, PUT, POST, DELETE, and HEAD.

Allow-Headers

Allowed cross-origin request header.

- Multiple domain names can be specified, with one domain name per line.
- Headers may be easily omitted. Therefore, if there is no special requirement, we recommend you set this field to `*`, meaning that all headers are allowed.
- The values are case-insensitive.
- Each header specified in `Access-Control-Request-Headers` must correspond to a value in `Allow-Headers`.

Expose-Headers

This is a list of headers exposed to the browser, i.e., the response headers that you access from the application (for example, JavaScript's `XMLHttpRequest` object).

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

Max-Age

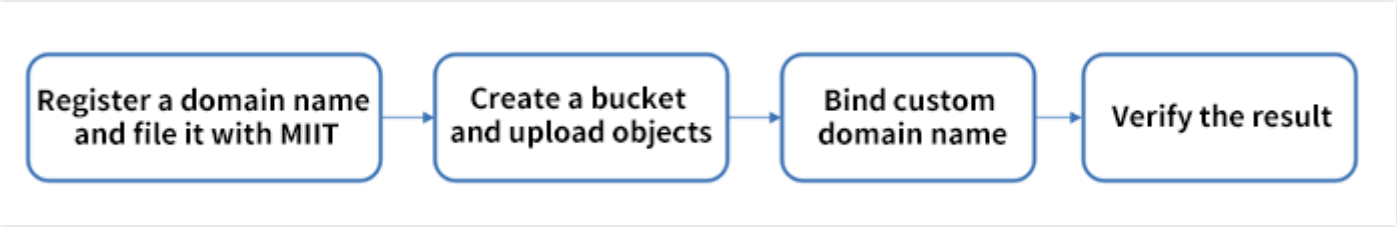
This is the time (in seconds) the browser can cache the results of a preflight request (OPTIONS request) for specific resources. In general cases, you can set it to a bigger value, for example, 60 seconds. This configuration item is optional.

Hosting Static Website

Last updated：2020-09-01 10:55:23

Overview

This document describes how to host an existing or new static website on COS and make it accessible to visitors through a custom domain name such as `www.example.com`. The specific steps are outlined below:



- For the static website configuration to take effect, you need to use the domain name of the static website instead of the COS default domain name to access the COS origin server.
- This practice only applies to scenarios where you need a domain name in Mainland China for your business.

Preparations

The following services will be used in the steps outlined below:

- Tencent Cloud Domain Service: Before hosting a static website, you need to register a domain name such as `www.example.com`. You can do so via the Tencent Cloud Domain Service.
- COS: You need to create a bucket in COS for storing the uploaded webpage contents.
- CDN: Together with Tencent Cloud DNS, CDN can accelerate the delivery of static website contents, reduce access delay, and improve availability, in addition to binding the domain name to the website content.
- Tencent Cloud DNS: Allows you to access the static website through a custom domain name.

The sample domain name `www.example.com` is used in the steps outlined in this document. For your purposes, replace it with your own domain name.

Step 1. Register the domain name and obtain ICP filing for service in China

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 service in China for it, skip this step and proceed to [step 2](#Create a bucket).
- Obtain ICP filing for the registered domain name.
- Register a domain name and obtain ICP filing for it.

Step 2. Create a bucket and upload content

After having registered the domain name and obtained ICP filing, you need to perform the following steps on the COS console:

- Create a bucket
- Configure the bucket and upload content

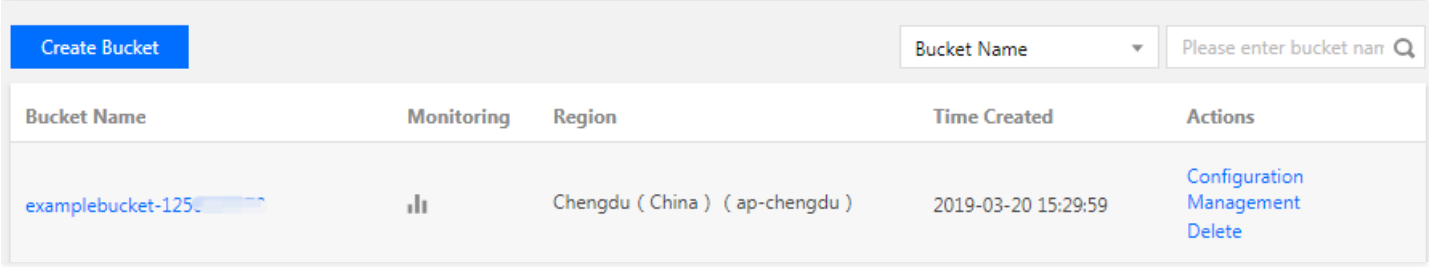
1. Create a bucket

Please log in to the COS Console with your Tencent Cloud account and create a bucket for your website. This bucket can be used to store website content and data.

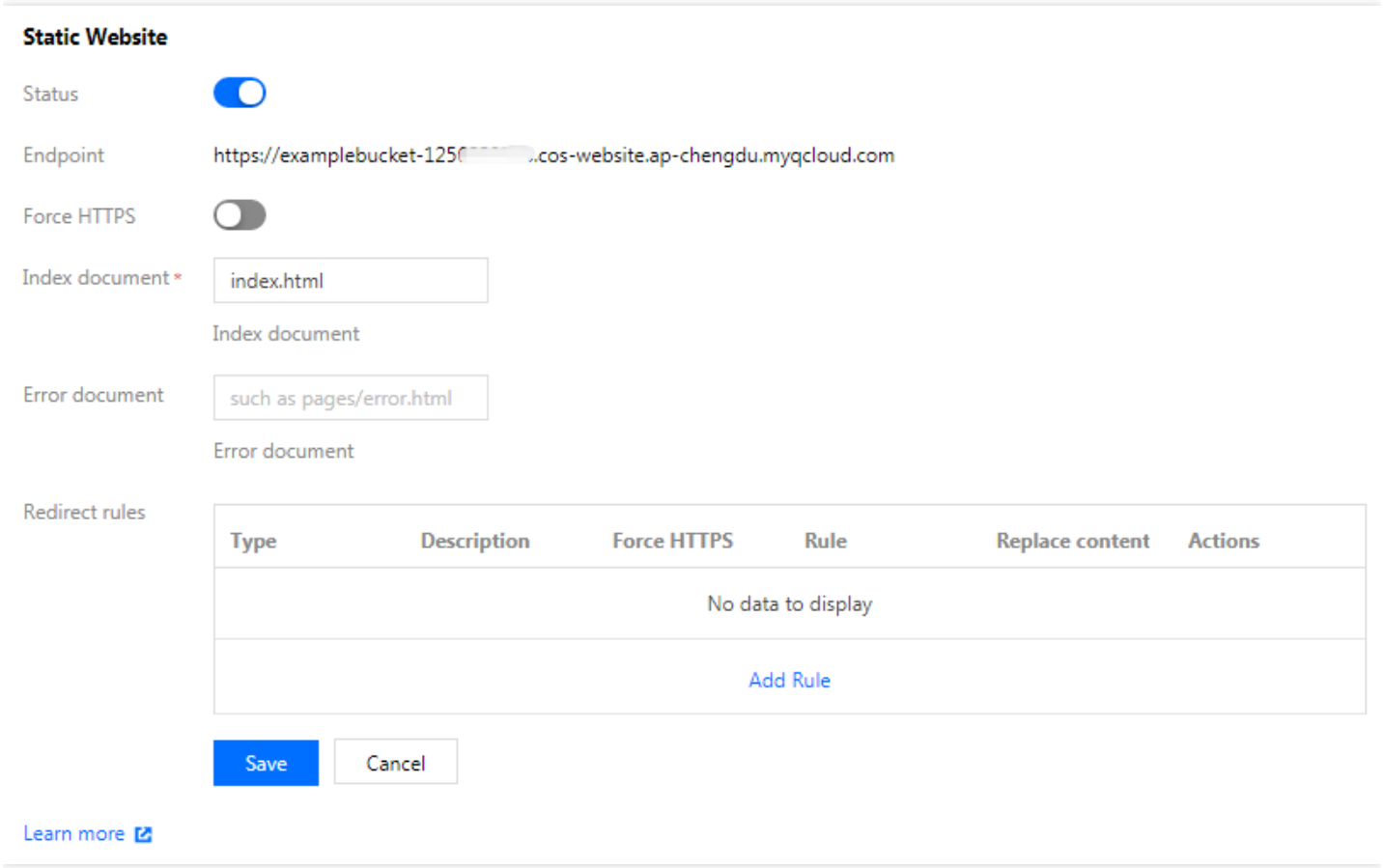
If you are using COS for the first time, you can create a bucket by accessing the **Bucket List** page on the left sidebar of the console and then clicking **Create Bucket** in the upper left-hand corner. For detailed directions, see Creating Buckets.

2. Configure the bucket and upload content

(1) Enable a **Static Website** for the bucket in the following way:
a. Log in to the COS Console, and click **Bucket List** on the left sidebar. Locate the bucket, and click **Configuration Management** under the **Actions** column.



b. Scroll down to the **Static Website** section under **Basic Configuration**, click **Edit**, and switch the **Status** to on. Configure the **Index document** field to `index.html`, leaving the other fields blank for now, 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>Thank you for using the static website feature of &nbspCOS&.</p>
<p>This is the homepage!</p>
```

```
</body>
</html>
```

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.

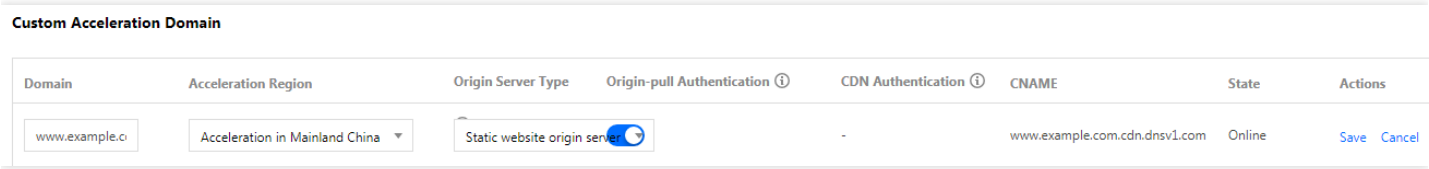
Step 3. Bind a custom domain name

We recommend that you use a custom CDN domain name so that Tencent Cloud CDN can accelerate the delivery of your static website contents for a better user experience.

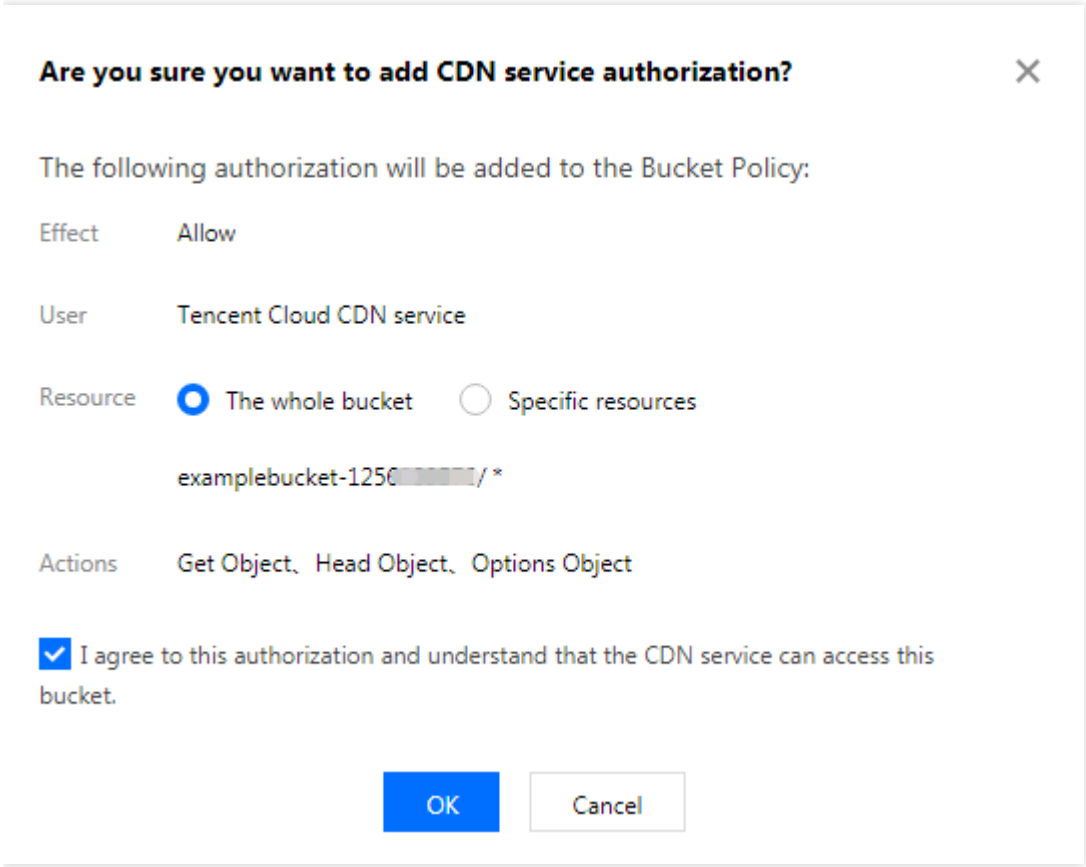
1. Add a domain name

(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) Click **Domain Management** to open the details page and then click **Add Domain** in the **Custom CDN Acceleration Domain** section to add a domain. Configure the domain as follows and then click **Save**.

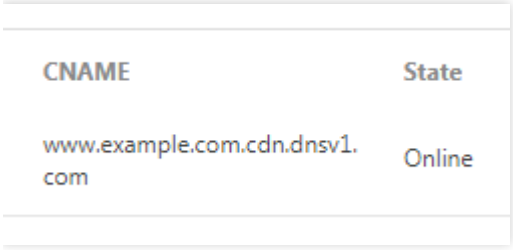
- **Domain**: Enter the custom domain name you have purchased (e.g. `www.example.com`).
- **Origin Server Type**: Select **Static website origin server**.
- **Origin-pull Authentication**: Turn it on.



(3) Follow the prompt above the domain list to add **CDN Service Authorization**.
Click **OK** in the pop-up window.



(4) Wait a few minutes until the domain name's status is displayed as **Online**. Then, copy the CNAME and proceed to [Step 3.2](#Step 3.2).



2. DNS

If you registered your domain name through a third-party ISP, go to your ISP, add a CNAME record for your custom CDN domain name, and point it to the CNAME record configured above. To use Tencent Cloud DNS, follow these steps:

(1) Log in to the DNS Console, locate your domain name, and click **DNS**.

(2) Click **Add Record**.

- **Host record**: www
- **Type**: CNAME

- **Value**: the CNAME from the preceding step

(3) Click **Save**.

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 bucket named `example`.
- `http://www.example.com/folder/`: Returns the index page (folder/index.html) under the `folder` directory in the bucket named `example`.
- `http://www.example.com/test.html` (a non-existing file): returns 404 error. If you want to customize the error document, you can do so in step 2 when configuring the static website, so that when an access attempt is made to a non-existing file, this error document will be displayed.

- 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：2022-01-17 14:15:20

What Is a Single-Page Application?

A single-page application (SPA), is a model of a web application or website that interacts with the user by dynamically rewriting the current page, instead of the traditional method of reloading entire new pages from the server. This approach avoids interruptions to the user experience by switching between pages and makes the application more like a desktop application. In an SPA, all necessary code (HTML, JavaScript, and CSS) is either retrieved with a single page load or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions.

At present, in the field of frontend development, common SPA development frameworks include React, Vue, and Angular.

This document uses two popular frameworks to illustrate how to use the **static website** feature provided by **Tencent Cloud's Cloud Object Storage (COS)** to quickly build an online available SPA, and provides solutions to some common problems.

Preparations

1. Install the Node.js environment.
2. Sign up for a Tencent Cloud account and verify your identity to ensure that you can log in to the Tencent Cloud COS console.
3. Create a bucket (to facilitate testing, set the bucket permission to **Public read & Private write**).

Writing Frontend Code

> Note：
> If you have already implemented the code, skip to Configuring the Bucket Static Website.

Quickly building an SPA with Vue

1. Run the following command to install the Vue CLI:

```
npm install -g @vue/cli
```

2. Run the following command in the Vue CLI to quickly create a Vue project. For more information, please see the official document.

```
vue create vue-spa
```

3. Run the following command to install `vue-router` in the root directory of the project:

```
npm install vue-router -S (Vue 2.x)
```

Or

```
npm install vue-router@4 -S (Vue 3.x)
```

4. Modify the `main.js` and `App.vue` files in the project.

Modify `main.js` as follows:

```
import { createApp } from 'vue'
import { createRouter, createWebHistory } from 'vue-router'
import App from './App.vue'
import Home from './components/Home.vue'
import Foo from './components/Foo.vue'
import Bar from './components/Bar.vue'
import Default from './components/404.vue'

const routes = [
    { path: '/', component: Home },
    { path: '/foo', component: Foo },
    { path: '/bar', component: Bar },
    { path: '/: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：
For simplicity, only some key code is shown here. For the full code, click here to download.

5. After modifying the code, run the following command for local preview:

```
npm run serve
```

6. After debugging and preview check, run the following command to package the production environment code:

```
npm run build
```

The `dist` directory is generated in the root directory of the project, and the Vue program code is ready.

Quickly building an SPA with React

1. Run the following command to install `create-react-app`:

```
npm install -g create-react-app
```

2. Use `create-react-app` to quickly create a React project. For more information, please see the official document.
3. Run the following command to install `react-router-dom` in the root directory of the project:

```
npm install react-router-dom -S
```

4. Modify the `App.js` file in the project.

```
import React from 'react';
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom'
import './App.css';

function Home() {
  return <h2>Home</h2>;
}

function About() {
  return <h2>About</h2>;
}

function NoMatch() {
  return <h2>404 Page</h2>
}

function App() {
  return (
    <Router>
      <div className="App">
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
          </ul>
        </nav>
        <Switch>
          <Route exact path="/">
            <Home />
          </Route>
          <Route path="/about">
            <About />
          </Route>
          <Route path="*">
            <NoMatch />
          </Route>
        </Switch>
      </div>
    </Router>
  );
}

export default App;
```

Note：

For simplicity, only some key code is shown here. For the full code, click here to download.

5. After modifying the code, run the following command for local preview:

```
npm run start
```

6. After debugging and preview check, run the following command to package the production environment code:

```
npm run build
```

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

Configuring the Bucket Static Website

1. Go to the details page of the created bucket and choose **Basic Configurations** > **Static Website**.
2. On the static website management page, configure information as shown in the figure below. For operation details, please see Setting up a Static Website.

Deploying the SPA to COS

1. Locate the bucket for which the static website is configured, and go to the corresponding **File List** page.
2. Upload all files in the compilation directory (default compilation directory: `dist` for Vue and `build` for React) to the root directory of the bucket. For operation details, please see Uploading Objects.
3. Access the bucket's static website domain (the access node shown in the figure below).

Then you can view the homepage of the deployed SPA (the Vue application in this example).

4. Try to switch between routes (**Home**, **Foo**, and **Bar**) and refresh the page to check whether the application works properly (no 404 error is reported upon refreshing under new routes).

FAQs

What if I don't want to use the default domain name of the static website? Can I use my own domain name?

In addition to the default static website domain name used above, COS also provides the default CDN acceleration domain name, custom CDN acceleration domain name, custom origin domain name, and more.

For configuration details, please see Domain Name Management Overview. After successful configuration, you can use your desired domain name to access the application.

After the application is deployed, rendering is successful after route switching, but the 404 error is reported whenever the page is refreshed. Why is that?

This may be caused by the missing or incorrect configuration of **Error document**. As the figure in Configuring the Bucket Static Website shows, **Error document** and **Index document** are both set to `index.html`.

Due to the nature of single-page applications, we need to ensure that the application entry (typically `index.html`) can be successfully accessed in any case in order to trigger a set of internal logic for subsequent routing.

After the route is switched, the page is displayed normally, but the HTTP status code is still 404. How do I make the HTTP status code 200?

The reason is that **Error Document Response Code** is not set during static website configuration. To solve this problem, set **Error Document Response Code** to 200. See the figure in Configuring the Bucket Static Website.

What should I do to make the application still return the status code 404 for accessing an incorrect path after Error document is set?

It is recommended to implement 404 logic in the frontend code: configure an underlying matching rule at the bottom layer of the routing configuration to configure the system to render a 404 component if the matching of all the preceding rules fails. The content of the 404 component can be designed and implemented according to your own requirements. For more information, please see the last configuration of the routing configuration in the code demo provided in this document.

Why is the error "403 Access Denied" reported during page access?

The possible cause is that the bucket permission is set to **Private (read-write)**. To solve the problem, change the permission to **Public read & Private write**.

In addition, if you use the CDN acceleration domain name to access a bucket with the **Private (read-write)** permission, be sure to enable **origin-pull authentication** so that you can authorize the CDN service to access COS resources.

Audio/Video Practices
Playing back COS Video File with TCPlayer

Last updated：2022-11-07 15:17:41

Overview

This document describes how to use the TCPlayer integrated in the TCToolkit SDK together with the rich audio/video capabilities of Cloud Infinite (CI) to play back video files stored in COS in a web browser.

Integration Guide

Step 1. Import player style and script files into the page

```
<!--Player style file-->
<link href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/tcplayer.min.css" rel="stylesheet">
<!--Player script file-->
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.0/tcplayer.v4.5.0.min.js"></script>
```

Note：

- We recommend you deploy the above static resources on your own when using the player SDK. Click here to download the player resources.
- Deploy the folder generated after decompression. Do not adjust the directories in the folder; otherwise, resource import exceptions may occur.

Step 2. Set the player container node

Place the player container in the desired place on the page. For example, add the following code to `index.html` (the container ID, width, and height can be customized).

```
<video id="player-container-id" width="414" height="270" preload="auto" playsinline webkit-playsinline>
</video>
```

Note：

- The player container must be a `<video>` tag.
- The `player-container-id` in the sample is the ID of the player container, which can be customized.
- We recommend you set the size of the player container zone through CSS, which is more flexible than the attribute and can achieve effects such as fit to full screen and container adaption.
- The `preload` attribute in the sample specifies whether to load the video after the page is loaded, which is usually set to `auto` for faster playback start. Other options include `meta` (to only load the metadata after the page is loaded) and `none` (to not load the video after the page is loaded). Due to system restrictions, videos will not be automatically loaded on mobile devices.
- The `playsinline` and `webkit-playsinline` attributes are used to implement inline playback if the standard mobile browser does not hijack the video playback. They are just for reference here and can be used as needed.
- If the `x5-playsinline` attribute is set, the X5 UI player will be used in the TBS kernel.

Step 3. Get the video file object address

1. Create a bucket.
2. Upload a video file object.
3. Get the video file object address in the format of `https://<bucketname-appid>.cos.<region>.myqcloud.com/xxx.<video format="">` .

Note：

- If cross-origin access is involved, you need to set CORS for the bucket as instructed in Setting CORS.
- If the bucket permission is private read/write, the object address needs to carry a signature. For more information, see Request Signature.

Step 4. Initialize the player and pass in the COS video file object URL

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is the player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); // COS video object address
```

Feature Guide

Playing back video files in different formats

1. Get the object address of the video file in the COS bucket.

> Note：
> We recommend you transcode videos for playback because untranscoded videos may experience compatibility issues during playback. You can get video files in different formats with CI's audio/video transcoding feature.

2. For different video formats, in order to ensure the compatibility with different browsers, corresponding dependencies need to be imported.

- MP4: There is no need to import other dependencies.
- HLS: If you want to play back HLS videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `hls.min.js` before importing `tcplayer.min.js` .

```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.min.0.13.2m.js"></script>
```

- FLV: If you want to play back FLV videos through HTML5 in a modern browser such as Chrome and Firefox, you need to import `flv.min.js` before importing `tcplayer.min.js` .

```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.min.1.6.2.js"></script>
```

- DASH: You need to import the `dash.all.min.js` file.

```
<script src="https://cos-video-1258344699.cos.ap-guangzhou.myqcloud.com/lib/dash.all.min.js"></script>
```

3. Initialize the player and pass in the object address.

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is the player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); // COS video address
```

Get code samples:

- Sample code for MP4 playback
- Sample code for FLV playback
- Sample code for HLS playback
- Sample code for DASH playback

Playing back PM3U8 video

PM3U8 refers to private M3U8 video file. COS provides a download authorization API for getting private M3U8 TS resources. For more information, see Private M3U8 API.

```
var player = TCPlayer("player-container-id", {
poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-process=pm3u8&expires=3600" // Relative validity period of the download credential for the private TS resource URL, which is 3,600 seconds.
});
```

Get code samples:

- Sample code for PM3U8 playback

Setting thumbnail

1. Get the object address of the thumbnail in the COS bucket.

> Note：
> CI's intelligent thumbnail feature can extract optimal frames to generate thumbnails to make the video content more engaging.

2. Set the thumbnail.

```
var player = TCPlayer("player-container-id", {
poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png"
});
```

Get code samples:

- Sample code for thumbnail configuration

Playing back HLS encrypted video

To ensure the security of video content and prevent unauthorized download and distribution of videos, CI provides the feature of encrypting HLS video content, which is more secure than privately readable files. Encrypted videos cannot be distributed to users without access for playback. Even if they are downloaded, they are still encrypted and cannot be redistributed maliciously. This protects your video copyrights from being infringed upon.

The steps are as follows:

1. Generate an encrypted video as instructed in Playing back HLS Encrypted Video.
2. Initialize the player and pass in the video object address.

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is t
he player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); //
HLS encrypted video address
```

Get code samples:

- Sample code for HLS encrypted playback

Switching definition

CI's adaptive bitrate streaming feature can transcode a video and remux it into adaptive bitstreams for output, helping you quickly distribute video content in different network conditions. The player can dynamically select the most appropriate bitrate to play back the video based on the current bandwidth.
The steps are as follows:

1. Generate the multi-bitrate adaptive HLS or DASH target file with CI's adaptive bitrate streaming feature.
2. Initialize the player and pass in the video object address.

```
var player = TCPlayer("player-container-id", {}); // `player-container-id` is t
he player container ID, which must be the same as that in HTML.
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); //
Multi-bitrate video address
```

Get code samples:

- Sample code for definition switching

Setting dynamic watermark

The player supports adding a dynamic watermark that changes its position and speed to a video. When using the dynamic watermark feature, the reference of the player object should not be exposed to the global environment;

otherwise, the dynamic watermark can be easily removed. CI also allows you to add a dynamic watermark to a video in the cloud. For more information, see Watermark Template APIs.

```
var player = TCPlayer("player-container-id", {
plugins:{
DynamicWatermark: {
speed: 0.2, // Speed
content: "Tencent Cloud CI", // Text
opacity: 0.7 // Opacity
}
}
});
```

Get code samples:

- Sample code for dynamic watermark configuration

Setting roll image ad

The steps are as follows:

1. Prepare the ad thumbnail and link.
2. Initialize the player, set the ad thumbnail and link, and set the ad node.

```
var PosterImage = TCPlayer.getComponent('PosterImage');
PosterImage.prototype.handleClick = function () {
window.open('https://intl.cloud.tencent.com.cn/products/ci'); // Set the ad lin
k
};
var player = TCPlayer('player-container-id', {
poster: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx..png', // Ad
thumbnail
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
var adTextNode = document.createElement('span');
adTextNode.className = 'ad-text-node';
adTextNode.innerHTML = 'Ad';
var adCloseIconNode = document.createElement('i');
adCloseIconNode.className = 'ad-close-icon-node';
adCloseIconNode.onclick = function (e) {
e.stopPropagation();
player.posterImage.hide();
};
player.posterImage.el_.appendChild(adTextNode);
player.posterImage.el_.appendChild(adCloseIconNode);
```

Get code samples:

- Sample code for roll image ad configuration

Setting video progress thumbnail (image sprite)

The steps are as follows:

1. Generate an image sprite with CI's video frame capturing feature.
2. Get the object addresses of the image sprite and VTT (image sprite location description file) generated in step 1.
3. Initialize the player and set the video and VTT file addresses.

```
var player = TCPlayer('player-container-id', {
plugins: {
VttThumbnail: {
vttUrl: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.vtt' // VTT f
ile
},
},
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
```

Get code samples:

- Sample code for image sprite configuration

Setting video subtitles

The steps are as follows:

1. Generate a subtitle file with CI's speech recognition feature.
2. Get the object address of the SRT file generated in step 1.
3. Initialize the player and set the video and SRT file addresses.

```
var player = TCPlayer('player-container-id', {});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
// Add the subtitles file
var subTrack = player.addRemoteTextTrack({
src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.srt', // Subtitl
es file
kind: 'subtitles',
srclang: 'zh-cn',
label: 'Chinese',
default: 'true',
```

```
}, true);
});
```

Get code samples:

- Sample code for video subtitles configuration

Setting multilingual video subtitles

The steps are as follows:

1. Generate a subtitles file with CI's speech recognition feature and translate it into multiple languages.
2. Get the object address of the multilingual SRT file generated in step 1.
3. Initialize the player and set the video and multilingual SRT file addresses.

```
var player = TCPlayer('player-container-id', {});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
// Set Chinese subtitles
var subTrack = player.addRemoteTextTrack({
src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/zh.srt', // Subtitle
s file
kind: 'subtitles',
srclang: 'zh-cn',
label: 'Chinese',
default: 'true',
}, true);
// Set English subtitles
var subTrack = player.addRemoteTextTrack({
src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/en.srt', // Subtitle
s file
kind: 'subtitles',
srclang: 'en',
label: 'English',
default: 'false',
}, true);
});
```

Get code samples:

- Sample code for multilingual subtitles configuration

Playing back HLS Encrypted Video

Last updated：2022-07-06 16:10:00

Background

To ensure the security of video content and prevent unauthorized download and distribution of videos, COS data processing provides the feature of encrypting HLS video content, which is more secure than privately readable files. Encrypted videos cannot be distributed to users without access for playback. Even if they are downloaded, they are still encrypted and cannot be redistributed maliciously. This protects your video copyrights from being infringed upon.

Based on the HLS encryption process, this practice builds a basic key management service and combines with Tencent Cloud VOD superplayer to play back video files transcoded and encrypted by COS HLS.

Directions

Step 1. Encrypt a video

Encrypt a video file in the COS bucket as instructed in Preventing Video Leakage with HLS Encryption.

Step 2. Set up the key service

1. Call the key API to generate KMS API sample code based on your programming language.

2. The following takes Node.js as an example to describe how to build a key service based on the sample code with Koa and call the KMS API to get a decryption key.

```
const Koa = require('koa')
const cors = require('koa2-cors')
const app = new Koa()
const tencentcloud = require("tencentcloud-sdk-nodejs")
app.use(cors()) // CORS configuration
app.use(async (ctx) => {
// The URI request in the generated m3u8 file will carry the parameters by defa
ult
const { Ciphertext, KMSRegion } = ctx.query
const KmsClient = tencentcloud.kms.v20190118.Client
const clientConfig = {
credential: {
// Account API key, which can be obtained at https://console.intl.cloud.tencen
t.com.cn/cam/capi
```

```
secretId: "SecretId",
secretKey: "SecretKey",
},

region: KMSRegion, // Region, such as "ap-guangzhou"
profile: {
httpProfile: {
endpoint: "kms.tencentcloudapi.com",
},
},
};
// Create a KMS object instance
const client = new KmsClient(clientConfig);
const params = {
"CiphertextBlob": Ciphertext,
};

try {
// Initiate a request to get the decryption key
const res = await client.Decrypt(params)

// Take out the key and return its binary data after Base64-decryption
const plaintext = res.Plaintext
const plainBuff = Buffer.from(plaintext, 'base64');
ctx.body = plainBuff
} catch (error) {
console.log(error);
}

})
// Listen on port 8080.
app.listen('8080', () => {
console.log('127.0.0.1:8080');
})
```

Step 3. Play back the encrypted video on the web

1. Import the player style and script files into the page:

```
<!--Player style file-->
<link href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.2/tcplayer.
min.css" rel="stylesheet"/>
<!--If you want to play back HLS videos through HTML5 in a browser such as Chro
me and Firefox, you need to import `hls.min.0.13.2m.js` before importing `tcpla
yer.v4.2.2.min.js`.-->
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.2/libs/hl
```

```
s.min.0.13.2m.js"></script>
<!--Player script file-->
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.2/tcplaye
r.v4.2.2.min.js"></script>
```

We recommend you deploy the above static resources on your own when using the player SDK. Click here to download the player resources.

Deploy the folder generated after decompression. Do not adjust the directories in the folder; otherwise, resource import exceptions may occur.

2. Set the player container node.

Add a player container where you want to play back videos. For example, you can add the following code to `index.html` (the container ID, width, and height are customizable).

```
<video id="player-container-id" width="414" height="270" preload="auto" playsinli
ne webkit-playsinline>
</video>
```

Note：

- The player container must be a `<video>` tag.
- The `player-container-id` in the sample is the ID of the player container, which can be customized.
- We recommend you set the size of the player container zone through CSS, which is more flexible than the attribute and can achieve effects such as fit to full screen and container adaption.
- The `preload` attribute in the sample specifies whether to load the video after the page is loaded, which is usually set to `auto` for faster start of the playback. Other options include `meta` (only loads the metadata after the page is loaded) and `none` (does not load the video after the page is loaded). Due to system restrictions, videos will not be automatically loaded on mobile devices.
- The `playsinline` and `webkit-playsinline` attributes are used to implement inline playback when the standard mobile browser does not hijack the video playback. They are just for reference here and can be used as needed.
- If the `x5-playsinline` attribute is set, the X5 UI player will be used in the TBS kernel.

3. On the bucket list page, get the **m3u8** file object address of the video encrypted in step 1.

4. Initialize the player and pass in the m3u8 object address.

```
var player = TCPlayer('player-container-id', {}); // `player-container-id` is t
he player container ID, which must be the same as that in HTML
player.src(https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/path/
example.m3u8); // m3u8 object address
```

Step 4. View the effect

1. The m3u8 file and decryption key are successfully obtained.

2. The video is decrypted and played back successfully.

Content Moderation
Blocking CDN Cache Based on Moderation Result

Last updated：2022-09-28 15:48:27

Overview

The content moderation feature can automatically block non-compliant files. It only applies to data stored on COS origins rather than data cached on CDN nodes.

This document describes how to promptly block non-compliant data cached in CDN through SCF and API Gateway.

Directions

1. Log in to the SCF console, select **Functions**, and click **Create** to create a function.
2. Select **Create from scratch** and set the following basic configuration items:

- Function type: Select **Event-triggered function**.
- Function name: Enter a custom function name.
- Region: Select the region of the bucket with the moderation feature enabled.
- Runtime environment: Select **Python 2.7**.

3. Configure the function code as follows:

- Submitting method: Select **Local ZIP file** or **Upload a ZIP pack via COS**. You can download the ZIP package here.
- Execution: Enter **index.main_handler**.

4. Click **Advanced configuration** to configure **Environment configuration**. You can modify all configuration items except the environment variables as needed.

- Resource type: CPU
- MEM: 512 MB
- Initialization timeout period: 65

- Execution timeout period: 30
- Environment variable:
  - CI_AUDITING_CALLBACK: Enter the callback address configured in the callback settings of content moderation here. The callback will be performed only if the callback address is set.
  - CDN_URL: Set the required CDN address to purge the CDN cache.
  - REGION: Set the required region where the bucket resides.
  - BUCKET_ID: Set the required bucket ID (i.e., bucket name), which is used to query the image style. Entering an incorrect value will result in an error during style query.
  - IMAGE_STYLE_SEPARATORS: Set the image style separator if you want to purge image style. You can enter multiple separators consecutively with no need to separate them.
  - CDN_REFRESH_TYPE: Set the image object cache purge method, which is purge by URL by default (that is, only the style will be purged). If you set this variable to **path**, the cache accessed by image processing parameters will be purged. Note that purge by path will remove files with longer filenames containing this filename.

    Be sure to configure the above environment variables correctly so that cache purge can work as expected.

5. Select **Execution Role** for **Permission configuration** and click **Create execution role** to enter the **Create custom role** page.

6. Select **Serverless Cloud Function (SCF)** as the role entity and click **Next**.

7. Select the `QcloudCDNFullAccess` and `QcloudCIReadOnlyAccess` role policies and click **Next**.

8. Name the role and click **Complete**.

9. After creating the custom role, go back to the function creation page, refresh the role drop-down list, and select the newly created role.

0. Click **Complete**.

1. Go to the API Gateway console to activate the API Gateway service.

2. Create an API gateway as instructed in Creating APIs Connecting to the SCF Backend.

3. Select **Publish** for **Release Environment** and click **Publish service**.

4. On the content moderation page in the CI console, set callback parameters for image or other target types, and set the address of the API gateway created in the previous step as the callback URL.

5. After the callback is configured, resource cache in CDN will be automatically purged based on the moderation callback result.

Data Security
Introduction to COS Data Security Solution

Last updated：2020-12-18 11:13:06

Pre-Event Protection

1. Isolating permissions

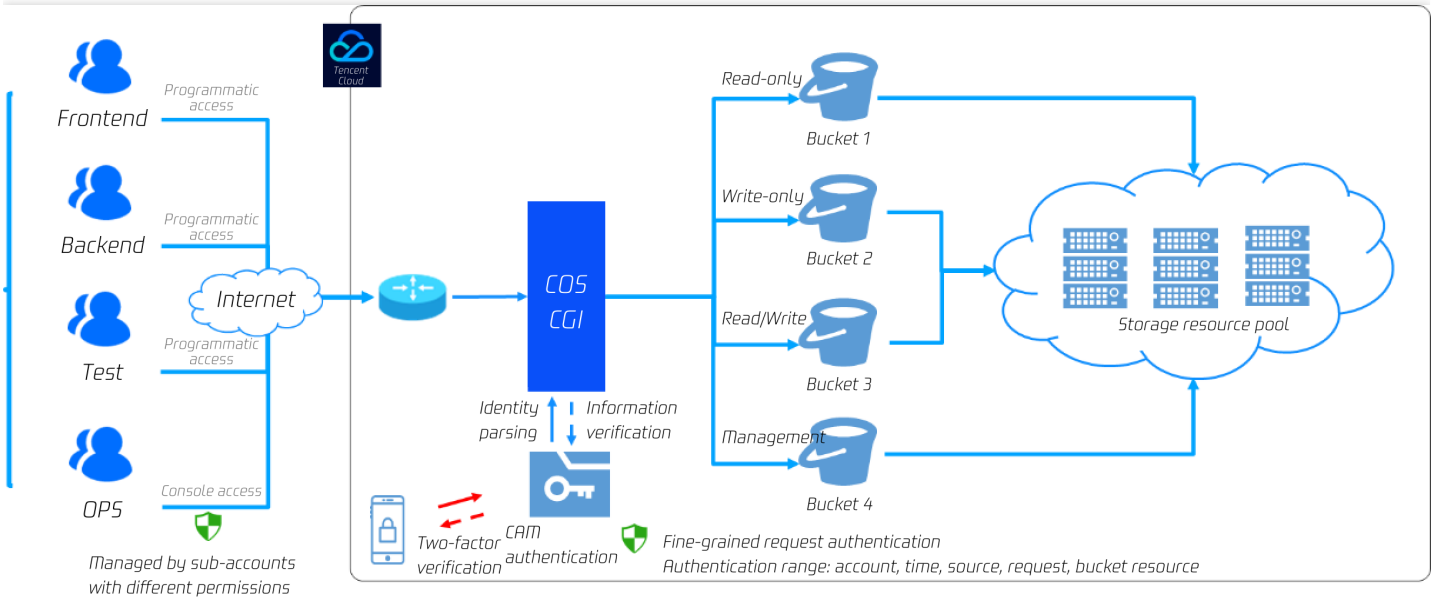
An in-cloud enterprise should pay attention to account security and resource authorization to protect the security system. To manage in-cloud resources properly, the following risks should be avoided:

- Using the Tencent Cloud root account for routine operations
- Over-authorizing employees' sub-accounts
- No access control over risky operations and high-permission sub-accounts
- Failure to regularly audit user permissions and login information
- No regulation for permission management

With Tencent Cloud Access Management (CAM), users can set levels for accounts and permissions for clearer and securer permission management. As for account levels, the root account can grant permissions, such as programmatic access and console access, to all valid CAM users (e.g., sub-accounts and collaborators). As for permission levels, you can grant service-level, API-level, resource-level, and other levels of permissions to CAM users. In this way, you can specify the operations a user can perform and the methods and resources the user can use under a specific condition.

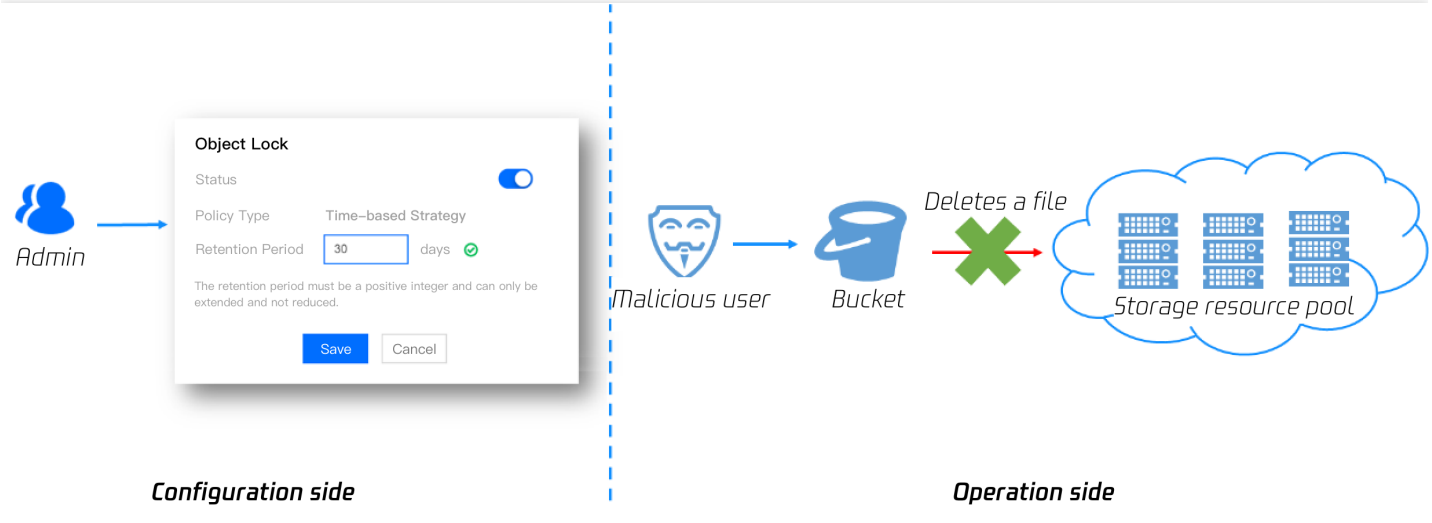
For example, you can create a sub-account and grant it permissions to manage the root account's resources without sharing the identity credential of the root account. Also, different access permissions to different resources can be granted to different users. For example, some sub-accounts can be granted the read access permission to a COS bucket, while some other sub-accounts or root accounts can own the write access to a COS object. The aforementioned resources, access permissions, and users can all be managed in batches to facilitate refined permission management.

You can also limit risky operations (for example, data deletion) to the console and enable MFA for two-factor verification at the same time. After MFA is enabled, risky operations will trigger an SMS verification code.



2. Locking objects

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

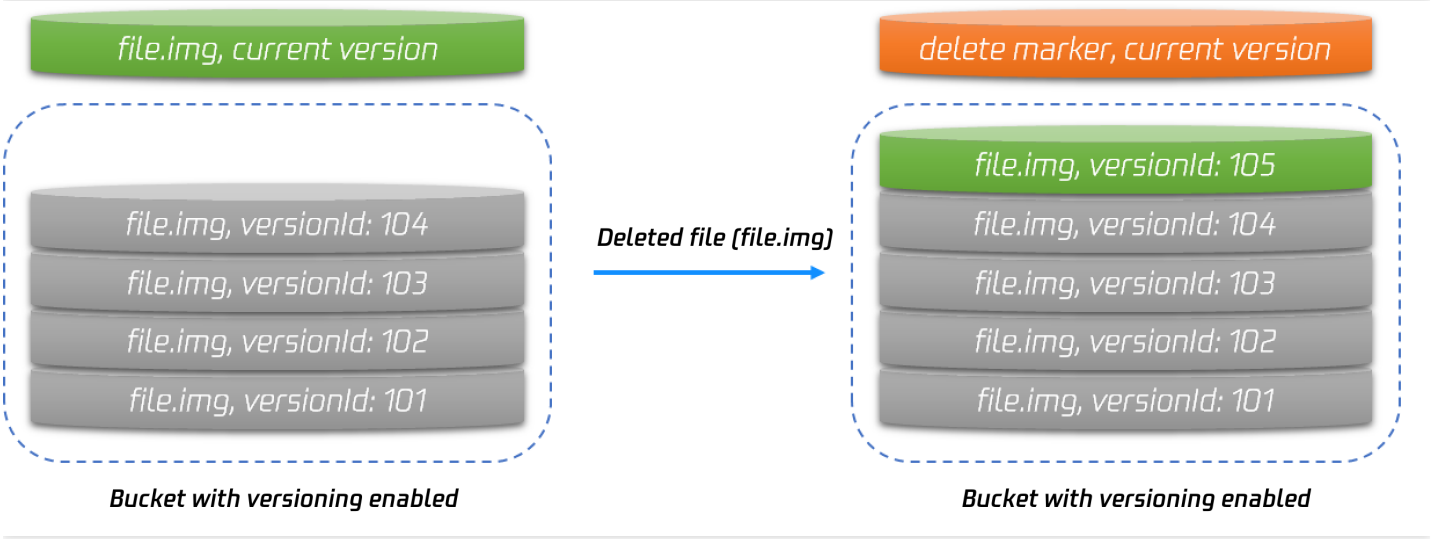


3. Disaster recovery

Tencent Cloud COS provides data management capabilities, including data encryption (secures read/write operations of sensitive files), versioning and bucket replication (facilitate remote disaster recovery for data persistence and data

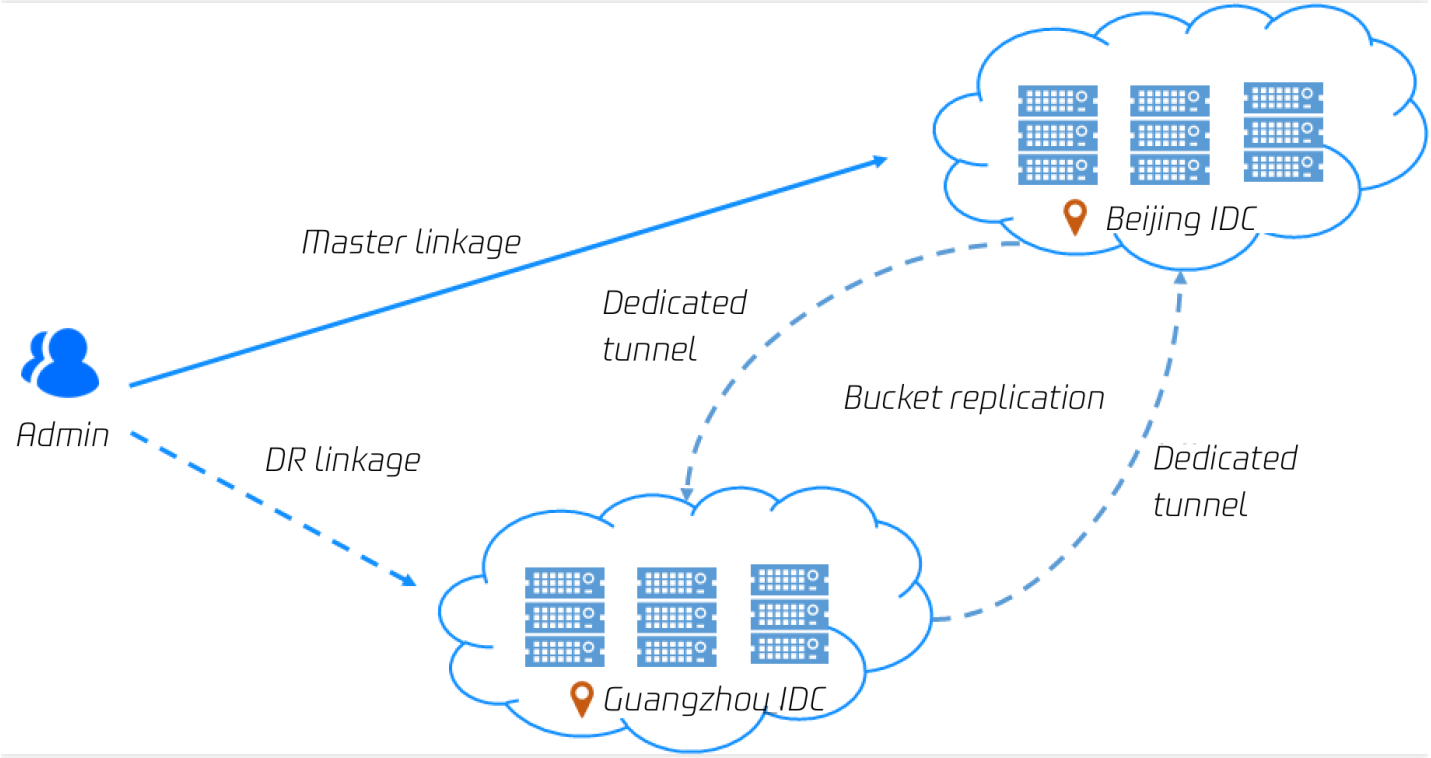
recovery for data deleted accidentally or maliciously), and lifecycle (transitions and deletes data for storage cost reduction).

The versioning feature ensures that files will not be overwritten or deleted. After versioning is enabled, if you upload a file whose name already exists, a new version of the file will be generated. If you delete a file, a delete marker will be inserted. You can access the data of any version with a version ID to implement data rollback, freeing yourself from the hassles associated with mis-deleting or overwriting data.

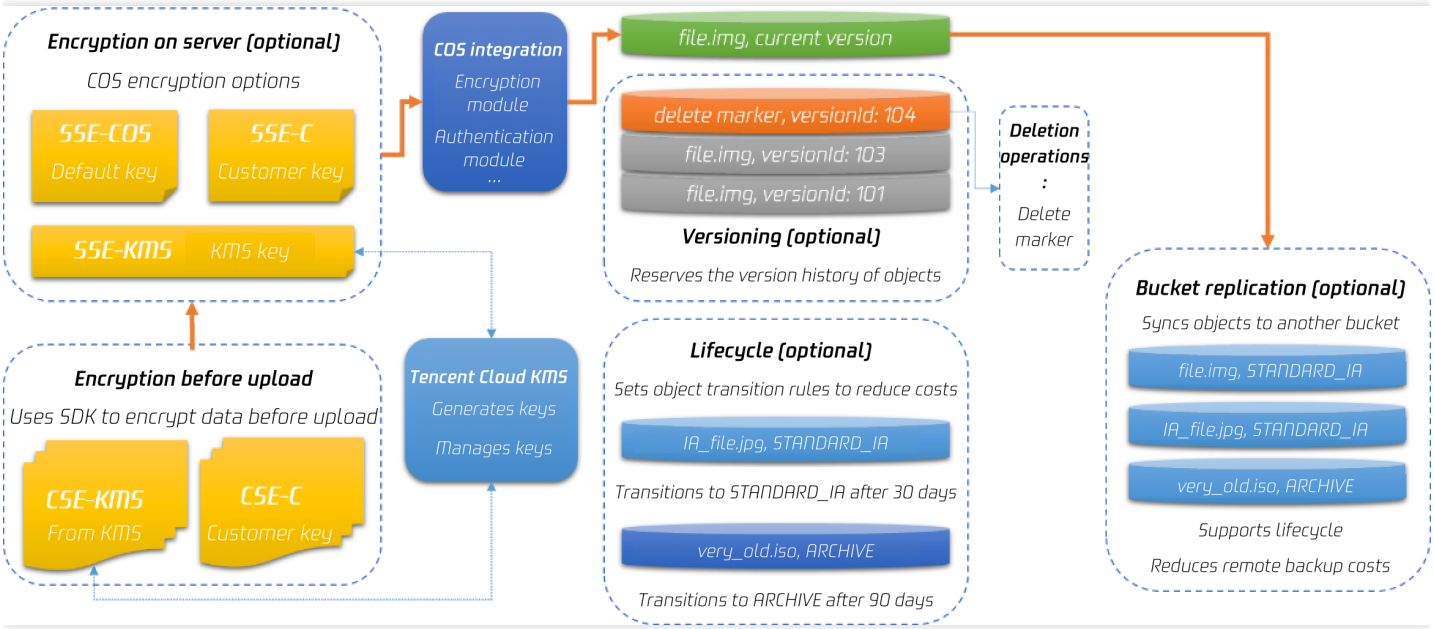


COS bucket replication enables users to copy all incremental files to IDCs in other cities over a dedicated tunnel to implement remote disaster discovery. Data deleted in the master bucket can be restored by batch-replicating data

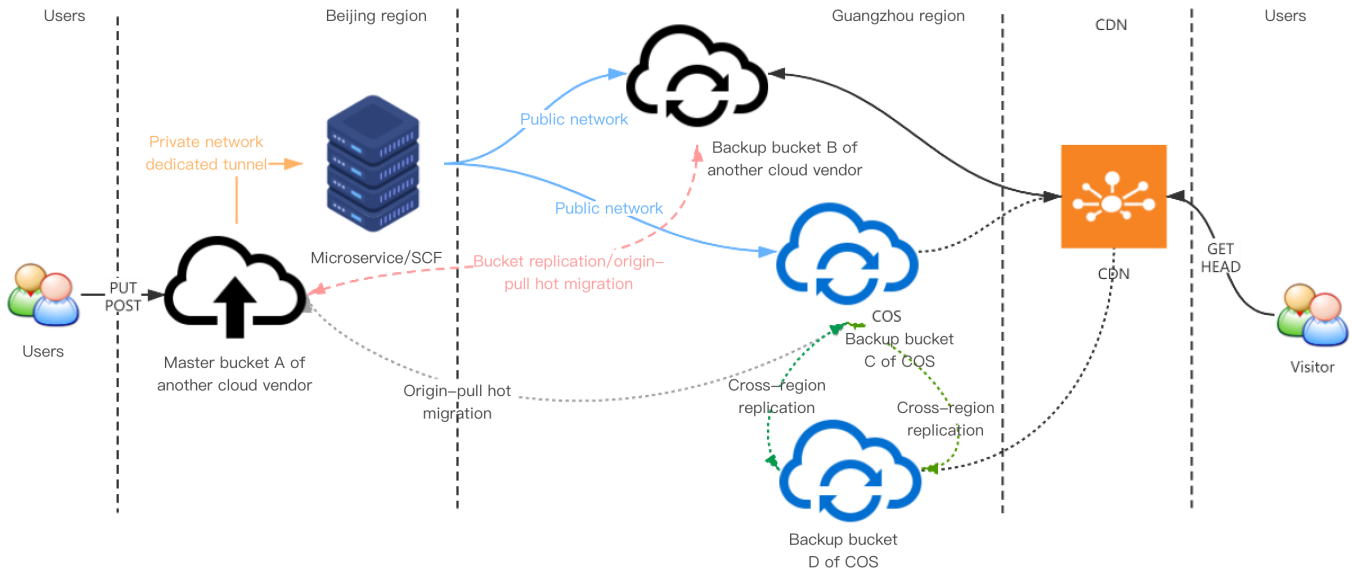
from the backup bucket.



Since versioning and bucket replication might lead to an increase in files, users can leverage the lifecycle feature to transition some backup data to a more affordable storage class (such as STANDARD_IA and ARCHIVE). Leveraging its data encryption, versioning, bucket replication, and lifecycle features, Tencent Cloud COS offers a complete code backup solution:



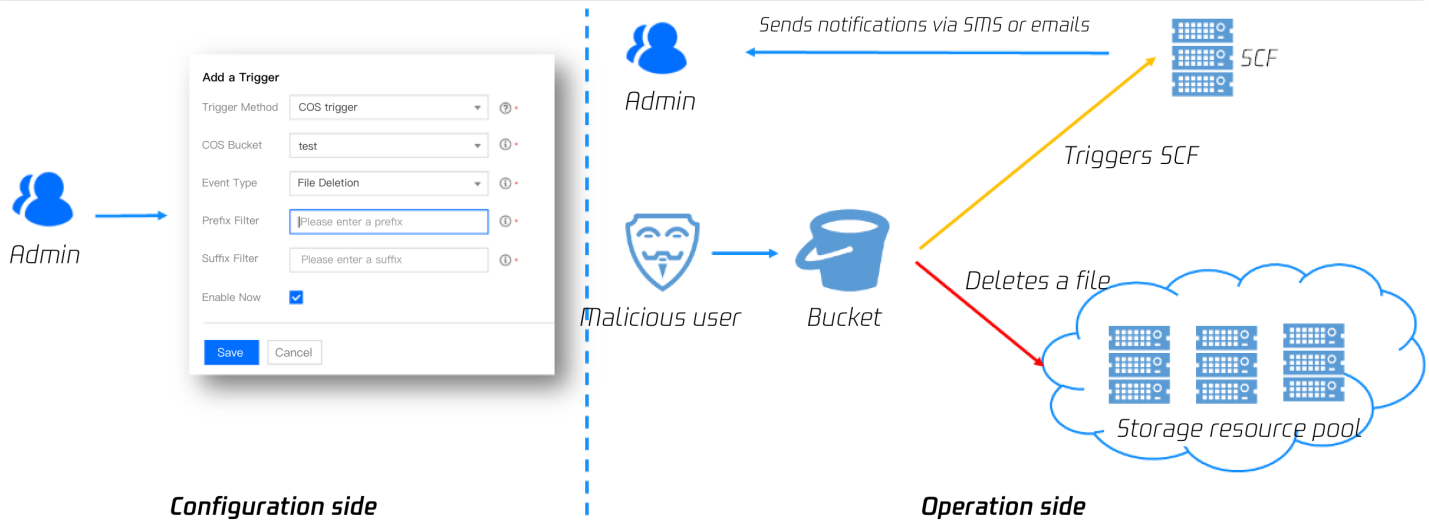
If you are using storage services of other cloud vendors and have strict requirements on data persistence, you can choose the SCF-based COS disaster recovery solution. For example, if your data is stored in other cloud vendors (e.g., AWS or OSS), you can implement remote disaster recovery by using SCF to trigger data sync or by using the bucket replication feature. You can also use SCF to trigger data migration to back up essential data to COS, and use the bucket replication feature for remote disaster recovery. Moreover, Tencent CAM allows you to manage the access permissions of COS data, ensuring that you can restore data from COS even in extreme cases.



Mid-Event Monitoring

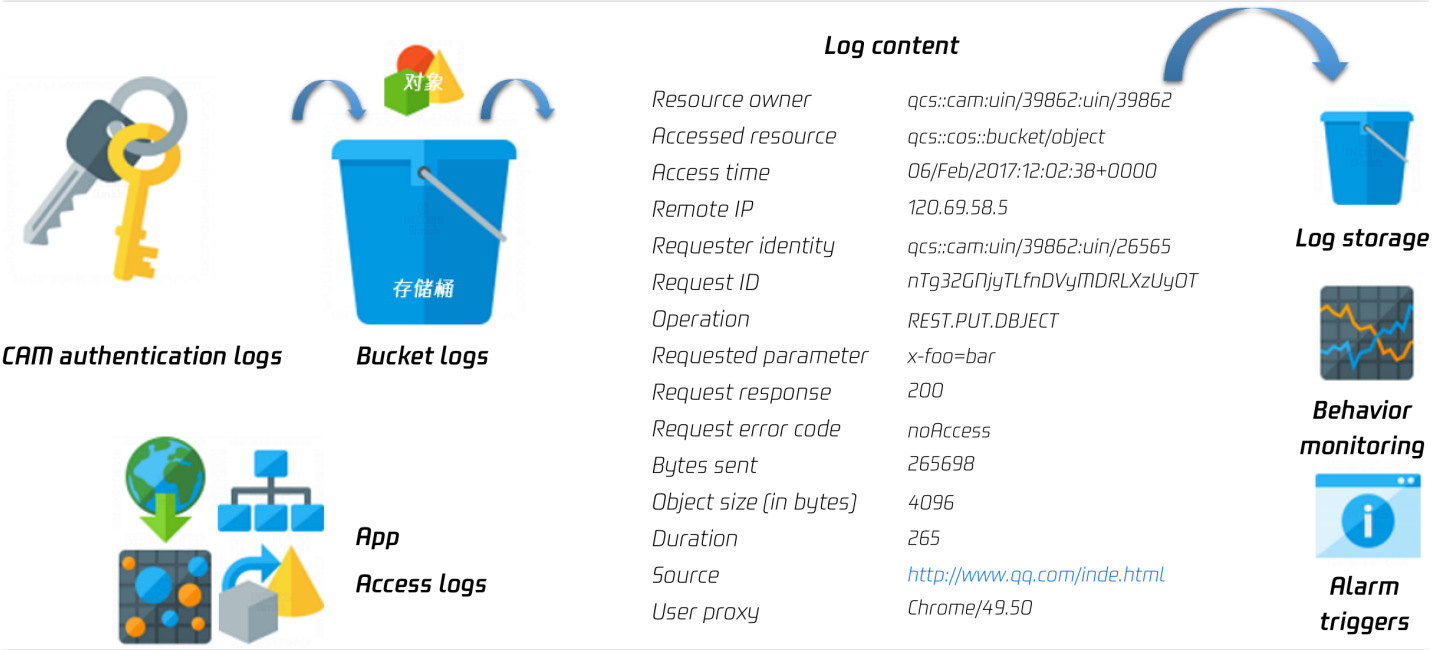
COS supports event notifications by working with SCF. You can use SCF to configure risky operations (such as `DeleteObject`). In this way, notifications will be sent to your email or phone when a risky operation is taking

place, allowing you to promptly detect and respond to risky operations.



Post-Event Tracing

COS provides multiple methods with a low threshold for log monitoring and audit. The logging feature allows users to trace the access logs of buckets. In this way, risky operations such as `DeleteObject`, `PutObjectCopy`, and `PutObjectACL` can be traced. In addition, users can leverage the CloudAudit logs to trace the bucket configurations, such as `DeleteBucket`, `PutBucketACL`, and `PutBucketPolicy`. Moreover, a modification on the permission configuration can also be traced.



Hotlink Protection Practice

Last updated：2021-07-29 10:36:15

Overview

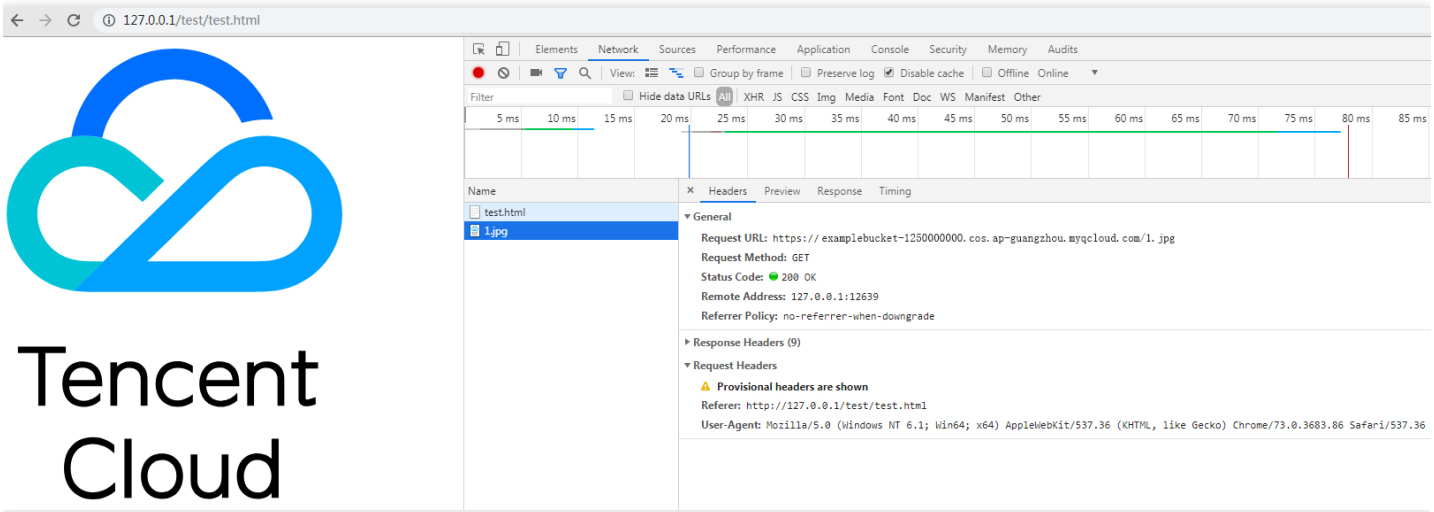
COS allows you to configure hotlink protection for your bucket. You can set a blocklist and allowlist for access sources to prevent resource hotlinking. This document describes how to configure hotlink protection for a bucket.

How Does Hotlink Protection Work

Hotlink protection works by checking the Referer address in the request header:

- Referer is a part of the header. When a browser sends a request to a web server, it usually carries a Referer to tell the server which page the request comes from, so that the server can decide to deny or allow the access to resources.
- If you open the file link `https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg` directly in a browser, the request header will not have a Referer.

For example, in the figure below, the image `1.jpg` is embedded in `https://127.0.0.1/test/test.html`, and a Referer pointing to the access origin will be carried when you access `https://127.0.0.1/test/test.html`:



Hotlink Protection Case Study

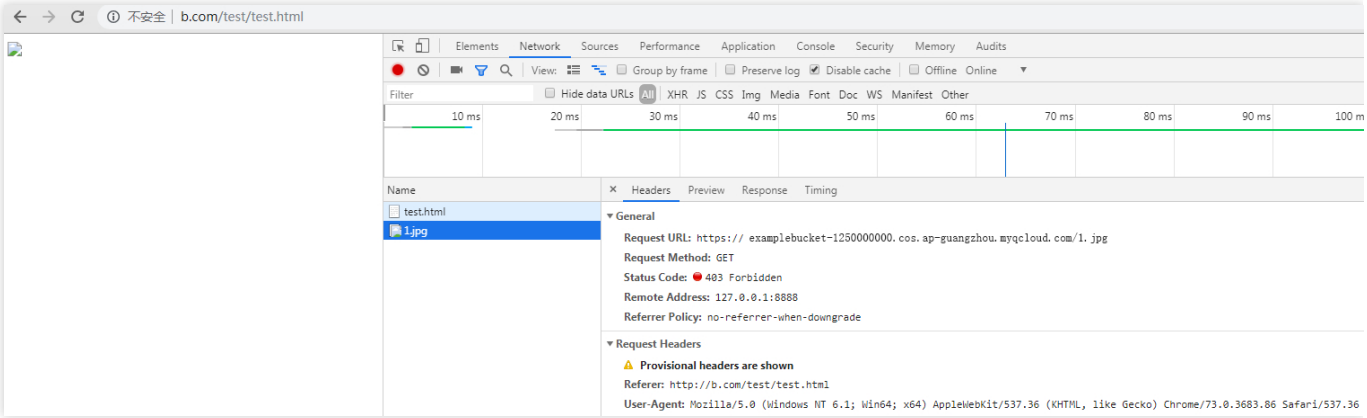
User A uploaded the image resource `1.jpg` to COS, and the accessible link to the image is `https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg`.

- User A embedded the image in their webpage `https://example.com/index.html` and the image is accessible.
- User B saw the image on user A's webpage and decided to embed it in their own webpage `https://b.com/test/test.html`, and user B's webpage can also display the image properly.
  In the above case, user A's image resource `1.jpg` was hotlinked by user B. User A doesn't know that their resource in COS is being used by user B's webpage and suffers from losses caused by extra traffic fees.

Solution

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

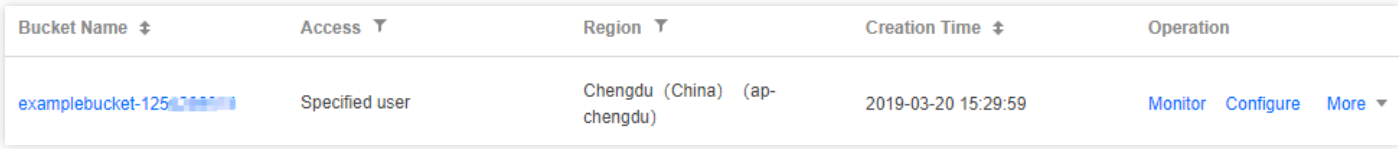
1. Set a hotlink protection rule for the bucket "examplebucket-1250000000". There are two options for preventing user B from hotlinking:
   - Option 1: configure a blacklist by entering the domain name `*.b.com`, and save it.
   - Option 2: configure a whitelist, enter `*.example.com` for the domain name, and save.
2. After hotlink protection is enabled:
   - The image can be displayed properly when `https://example.com/index.html` is accessed.
   - The image cannot be displayed when `https://b.com/test/test.html` is accessed, as shown below:



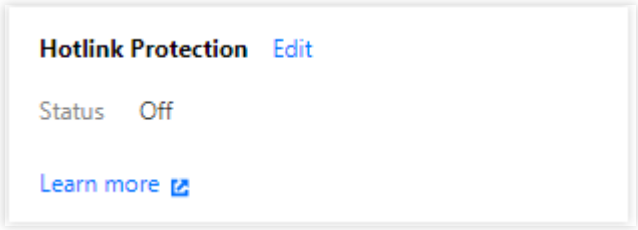
Directions

1. Log in to the COS Console and click **Bucket List** on the left sidebar to enter the bucket list page.

2. Select the bucket for which to configure hotlink protection and enter it.



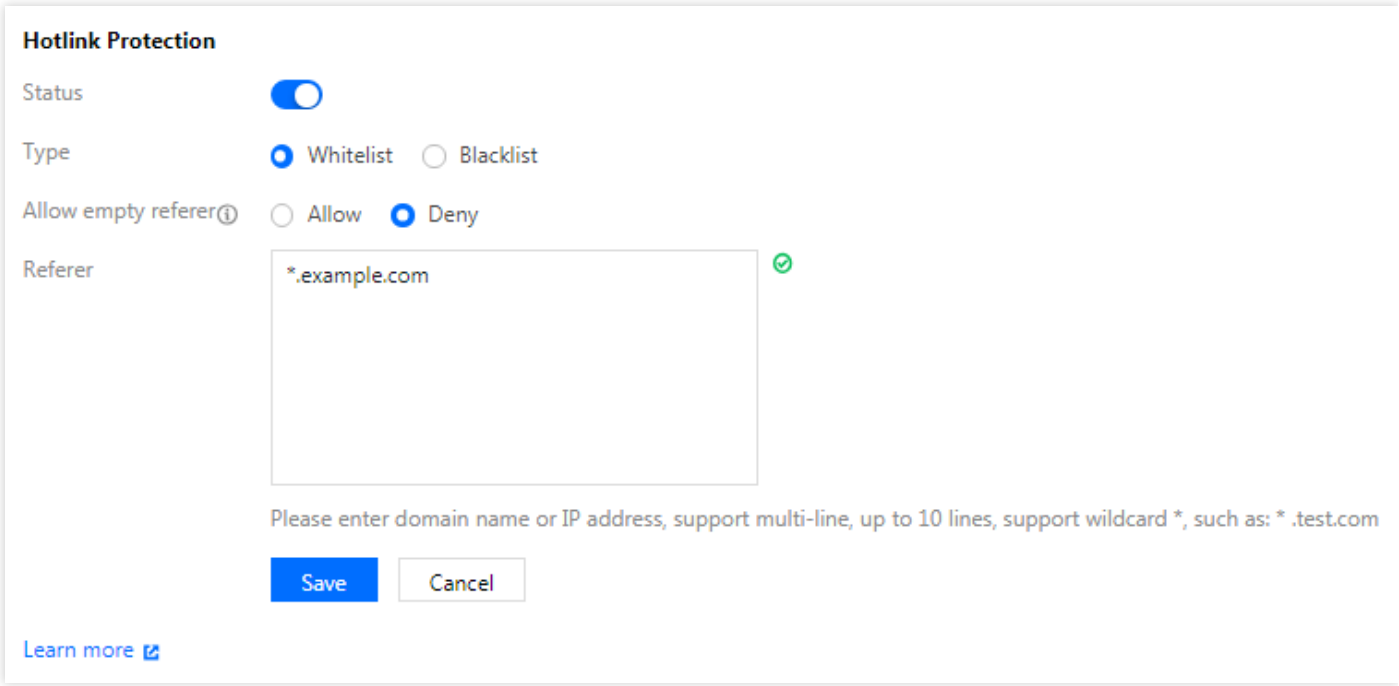
3. Click **Security Management** > **Hotlink Protection** on the left.
4. In the **Hotlink Protection** area, click **Edit**.



5. Enable hotlink protection and configure the list type and domain name. Here, select Option 2 as detailed below:
   - **Type**: blocklist or allowlist
     - **Blocklist**: It prohibits domain names in the list to access the default access address of the bucket. If a domain name **in the list** accesses the default access address of the bucket, a 403 error will be returned.
     - **Allowlist**: It prohibits domain names not in the list to access the default access address of the bucket. If a domain name **not in the list** accesses the default access address of the bucket, a 403 error will be returned.
   - **Referer** : Up to 10 domain names can be set and they will be matched by a prefix. Domain names, IPs, and asterisk `*` are supported formats (one address per line). Below are configuration rule description and examples:
     - Domain names and IPs with a specific port are supported, such as `example.com:8080` and `10.10.10.10:8080` .
     - If `example.com` is configured, addresses prefixed with `example.com` can be hit, such as `example.com/123` .
     - If `example.com` is configured, addresses prefixed with `https://example.com` and `http://example.com` can be hit.
     - If `example.com` is configured, the domain name with a specific port can also be hit, such as `example.com:8080` .
     - If `example.com:8080` is configured, the domain name `example.com` cannot be hit.
     - If `*.example.com` is configured, its second-level and third-level domain names can be restricted, such as `example.com` , `b.example.com` , and `a.b.example.com` .

> Note：
> After hotlink protection is **enabled**, the corresponding domain names must be entered.

6. After completing the configuration, click **Save**.



FAQs

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

How to Prevent Video Disclosure via HLS Encryption

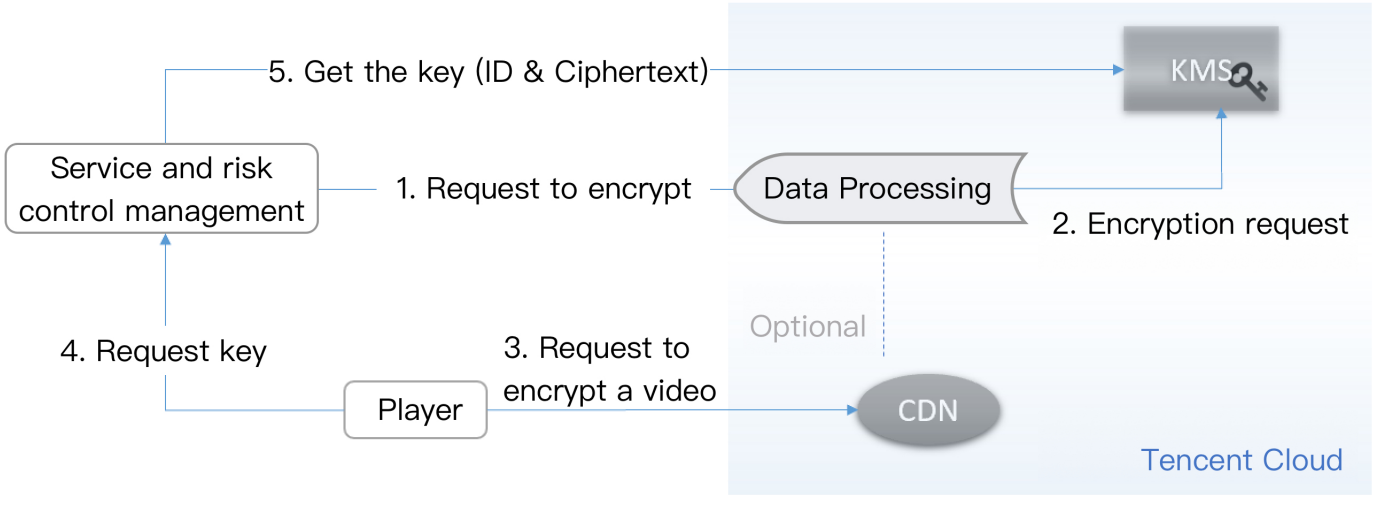
Last updated：2022-07-13 14:07:27

Overview

COS data processing offers the encryption feature for HTTP Live Streaming (HLS) videos. The encrypted video is not available for users without access permissions. HLS encryption must be used together with Key Management Service (KMS) and the Token service. This document describes a solution suitable for services that can build a complete set of authentication and KMS.

Note：
It supports only HLS encryption.

How It Works



.

Note：

In this solution, COS is connected to KMS.

Encryption procedure

1. Upload a video to COS and request HLS encryption.
2. COS requests an encryption key from KMS.
3. COS encrypts the HLS video through transcoding.
4. After encryption, COS will deliver the encrypted video file through Content Delivery Network (CDN).

Decryption procedure

1. The end user logs in to the player terminal and is identified by the service side. After the identity verification is successful, the service side will assign a token to the player terminal and return the tokenized playback URL to the player.
2. After obtaining the URL, the player parses the URI of the M3U8 file and request the key from URI.
3. Risk Control Management Service verifies the validity of the request, and then queries the key by calling KMS APIs.
4. KMS returns the decryption key to the player terminal. The player decrypts the M3U8 file by using the key for playback.

Encryption Directions

1. Log in to the COS console.
2. Click **Bucket List** on the left sidebar.
3. Click the name of the bucket where you want to store the video.
4. On the left sidebar, select **Data Processing Workflow** > **Common Configuration**. Then, select the **Template** tab to go to the template configuration page.
5. Select **Audio/Video Transcoding** and click **Create Transcoding Template**.
6. In the **Create Transcoding Template** window, configure the following items:

- Template Name: It can contain up to 64 letters, digits, underscores (_), hyphens (-), and asterisks (*).
- Encapsulation Format: Select HLS.
- Transcoding Duration: You can select the input file duration or custom configuration.
- Advanced configuration:
  - Video Encryption: Enable.
  - UriKey: KMS URL built by users.

7. Click **OK**, and then you can apply this template to encrypt videos when configuring workflow or configuring job.

Data Verification
MD5 Verification

Last updated：2021-05-25 11:40:25

Overview

Errors may occur when data is being transmitted between the client and the server. COS can guarantee the integrity of the uploaded data through MD5 verification. Only when the MD5 checksum received by the COS server is the same as that you set can the data be successfully uploaded.

Each object in COS has a corresponding ETag, which is the information identifier of the object content when the object is created. However, the ETag is not necessarily equivalent to the MD5 checksum of the object content. Therefore, the ETag cannot be used to verify whether the downloaded object is the same as the original object. In this case, you can use custom object metadata (x-cos-meta-*) to verify the object consistency.

Data Verification Methods

- Verify an uploaded object
  If you need to verify whether the object uploaded to COS is the same as the local object, you can set the Content-MD5 field in the HTTP upload request to the Base64-encoded MD5 checksum of the object content. After that, the COS server will verify the uploaded object. Only when the MD5 checksum received by the COS server is the same as the Content-MD5 value you set can the object be successfully uploaded.
- Verify a downloaded object
  If you need to verify whether the downloaded object is the same as the original object, you can use a verification algorithm to calculate the checksum of the object when it is uploaded, set the checksum of the object through custom metadata, recalculate the checksum of the object after downloading the object, and then verify it against the custom metadata. In this mode, you can choose the verification algorithm as you wish, but for the same object, the algorithm used during upload should be the same as that used during download.

API Samples

Simple Upload Request

Below is a sample request for object upload. When uploading the object, set the Content-MD5 to the Base64-encoded MD5 checksum of the object content and set the custom metadata "x-cos-meta-md5" to the checksum of the object.

Only when the MD5 checksum received by the COS server is the same as the Content-MD5 value you set can the object be successfully uploaded.

> Note：
> In the sample, the checksum of the object is obtained through the MD5 checksum algorithm, and you can choose other algorithms as you wish.

```
PUT /exampleobject HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 21 Jun 2019 09:24:28 GMT
Content-Type: image/jpeg
Content-Length: 13
Content-MD5: 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 using multipart upload, COS will verify the MD5 checksum of each part instead of the MD5 checksum of the merged file.

Object download response

Below is a sample response obtained after you send an object download request. You can get the custom metadata "x-cos-meta-md5" of the object from the response and then check it against the recalculated checksum of the object to verify whether the downloaded object is the same as the original object.

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 13
Connection: close
Accept-Ranges: bytes
Cache-Control: max-age=86400
Content-Disposition: attachment; filename=example.jpg
Date: Thu, 04 Jul 2019 11:33:00 GMT
ETag: "b62e10bcab55a88240bd9c436cffdcf9"
Last-Modified: Thu, 04 Jul 2019 11:32:55 GMT
Server: tencent-cos
x-cos-request-id: 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] for Python SDK.(https://intl.cloud.tencent.com.cn/document/product/436/31546).

1. Initialization configuration

Configure user attributes, including SecretId, SecretKey, and region, and create a client object.

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import logging
```

```
import hashlib
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# Set user attributes, including SecretId, SecretKey, and region
# APPID has been removed from the configuration. Please specify it using the `Buc
ket` parameter in the format 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 sa
mple)
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. Verify an object uploaded using simple upload

(1) Calculate the checksum of the object

Get the checksum of the object through the MD5 checksum algorithm (you can choose other algorithms as you wish).

```
object_body = 'hello cos'
# Get the MD5 checksum of the object
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

(2) upload objects using simple upload

EnableMD5=True in the code indicates the enablement of MD5 verification for object upload. The SDK for Python will calculate the Content-MD5. Enabling this will increase the time it takes to upload the object. Only when the MD5 checksum of the object received by the COS server is the same as the Content-MD5 can the object be successfully uploaded.

x-cos-meta-md5 is a custom parameter (in the name format of x-cos-meta-*), which represents the MD5 checksum of the object.

```
# Upload the object using simple upload and enable MD5 verification
response = client.put_object(
Bucket='examplebucket-1250000000', # Replace with your own bucket name. Here, exa
mplebucket is a sample bucket, and 1250000000 is a sample APPID
Body='hello cos', # Content of the uploaded object
Key='example-object-1', # Replace with the key value of your uploaded object
EnableMD5=True, # Enable MD5 verification for upload
Metadata={ # Set the custom parameter and save the MD5 checksum of the object to
the COS server as the parameter value
```

```
'x-cos-meta-md5' : md5_str
}
)
print 'ETag: ' + response['ETag'] # Etag value of the object
```

(3) Download the object

Download the object and get the custom parameter.

```
# Download the object
response = client.get_object(
Bucket='examplebucket-1250000000', # Replace with your own bucket name. Here, exa
mplebucket is a sample bucket, and 1250000000 is a sample APPID
Key='example-object-1' # Key value of the download object
)
fp = response['Body'].get_raw_stream()
download_object = fp.read() # Get the object content
print "get object body: " + download_object
print 'ETag: ' + response['ETag']
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5'] # Get the custom parameter
"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 an object uploaded in parts

(1) Calculate the checksum of the object

Simulate object parts and calculate the checksum of the entire object. The MD5 checksum algorithm is used to obtain the checksum of the object in the sample below, and you can choose other algorithms as you wish.

```
OBJECT_PART_SIZE = 1024 * 1024 # Size of each simulated part
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123 # Total size of the object
object_body = '1' * OBJECT_TOTAL_SIZE # Object content
# Calculate the MD5 checksum of the entire object content
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

(2) Initialize the multipart upload

When initializing the multipart upload, set the custom parameter "x-cos-meta-md5" and use the MD5 checksum of the entire object as the parameter value.

```
# Initialize the multipart upload
response = client.create_multipart_upload(
Bucket='examplebucket-1250000000', # Replace with your own bucket name. Here, exa
mplebucket is a sample bucket, and 1250000000 is a sample APPID
Key='exampleobject-2', # Replace with the key value of the uploaded object
StorageClass='STANDARD', # Storage class of the object
Metadata={
'x-cos-meta-md5' : md5_str # Set the custom parameter to the MD5 checksum
}
)
# 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 parts
response = client.upload_part(
Bucket='examplebucket-1250000000', #Replace with your own bucket name and APPID
Key='exampleobject-2', # Key value of the object
Body=body,
PartNumber=part_number,
UploadId=upload_id,
EnableMD5=True # Enable part verification and the COS server will perform MD5 ver
ification on each part
)
etag = response['ETag'] # ETag represents the MD5 checksum of each part
part_list.append({'ETag' : etag, 'PartNumber' : part_number})
print etag + ', ' + str(part_number)
```

(4) Complete the multipart upload

After all parts are uploaded, you need to complete the multipart upload operation. The ETag and PartNumber of each part should be in one-to-one correspondence which will be used by the COS server to verify the part accuracy. After the multipart upload completes, the returned ETag represents the unique tag value of the merged object but not the MD5 checksum of the entire object content. As a result, you can use the custom parameter to verify the object when downloading it.

```
#Complete the multipart upload
response = client.complete_multipart_upload(
Bucket='examplebucket-1250000000', # Replace with your own bucket name. Here, exa
mplebucket is a sample bucket, and 1250000000 is a sample APPID
Key='exampleobject-2', # Key value of the object
UploadId=upload_id,
MultipartUpload={ # Requires one-to-one correspondence between ETag and PartNumbe
r for each part
'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 unique
ness
print "ETag: " + response['ETag']
```

```
print "Location: " + response['Location'] #URL
print "Key: " + response['Key']
```

(5) Download the object

Download the object and get the custom parameter.

```
# Download the object
response = client.get_object(
Bucket='examplebucket-1250000000', #Replace with your own bucket name and APPID
Key='exampleobject-2' # Key value of the object
)
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：2021-11-11 14:35:36

Overview

Errors may occur when data is transferred between the client and the server. COS can not only verify data integrity through MD5 and custom attributes, but also the CRC64 check code.

COS will calculate the CRC64 of the newly uploaded object and store the result as object attributes. It will carry x-cos-hash-crc64ecma in the returned response header, which indicates the CRC64 value of the uploaded object calculated according to ECMA-182 standard. If an object already has a CRC64 value stored before this feature is activated, COS will not calculate its CRC64 value, nor will it be returned when the object is obtained.

Description

APIs that currently support CRC64 include:

- APIs for simple upload
  - PUT Object and POST Object: you can get the CRC64 check value for your file from the response headers.
- Multipart upload APIs
  - Upload Part: you can compare and verify the CRC64 value returned by COS against the value calculated locally.
  - Complete Multipart Upload: returns a CRC64 value for the entire object only if each part has a CRC64 attribute. Otherwise, no value is returned.
- The Upload Part - Copy operation returns a corresponding CRC64 value.
- When you call the PUT Object - Copy, the CRC64 value is returned only if the source object has one.
- The HEAD Object and GET Object operations return the CRC64 value provided the object has one. You can compare and verify the CRC64 value returned by COS against that calculated locally.

API Samples

Upload Part response

The following example shows the response to an Upload Part request. The `x-cos-hash-crc64ecma` header represents the CRC64 value of a part, which you can compare against the locally calculated CRC64 value to verify the part integrity.

```
HTTP/1.1 200 OK
content-length: 0
connection: close
date: Thu, 05 Dec 2019 01:58:03 GMT
etag: "358e8c8b1bfa35ee3bd44cb3d2cc416b"
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0MmJfMjBiNDU4NjRfNjkyZl80ZjZi****
```

Complete Multipart Upload response

The following example shows the response to a Complete Multipart Upload request. The `x-cos-hash-crc64ecma` header represents the CRC64 value of an entire object, which you can compare against the locally calculated CRC64 value to verify the object integrity.

```
HTTP/1.1 200 OK
content-type: application/xml
transfer-encoding: chunked
connection: close
date: Thu, 05 Dec 2019 02:01:17 GMT
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0ZWRfMjNiMjU4NjRfOGQ4Ml81MDEw****
[Object Content]
```

SDK Samples

Python SDK

The following example uses the Python SDK to verify object integrity. The complete sample code is as follows.

> Note：
> The code is based on Python 2.7. For more information on how to use the Python SDK, see Object Operations.

1. Initialization configuration

Configure user attributes, including SecretId, SecretKey, and region, and create a client object.

```
# -*- coding=utf-8
from qcloud_cos import CosConfig
```

```
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import logging
import hashlib
import crcmod
logging.basicConfig(level=logging.INFO, stream=sys.stdout)
# Configure user attributes, including SecretId, SecretKey, and region
# APPID has been removed from the configuration. Please specify it using the `Buc
ket` parameter in the format 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 sa
mple)
token = None # If a temporary key is used, the token needs to be specified. This
is optional and 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 checksum

Simulate object parts and calculate the checksum of the entire object.

```
OBJECT_PART_SIZE = 1024 * 1024 # Size of each simulated part
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123 # Total size of the object
object_body = '1' * OBJECT_TOTAL_SIZE # Object content
#Calculate the checksum of the entire object.
c64 = crcmod.mkCrcFun(0x142F0E1EBA9EA3693L, initCrc=0L, xorOut=0xffffffffffffffff
L, rev=True)
local_crc64 =str(c64(object_body))
```

3. Initialize the multipart upload

```
# Initialize the multipart upload
response = client.create_multipart_upload(
Bucket='examplebucket-1250000000', #Replace with your own bucket name and APPID
Key='exampleobject', # Replace with the key value of your uploaded object
StorageClass='STANDARD', # Storage class of the object
)
#Get the UploadId of the multipart upload
upload_id = response['UploadId']
```

4. Upload the object using multipart upload

During a multipart upload, an object is divided into multiple (up to 10,000) parts for upload. The size of each part ranges from 1 MB to 5 GB, except the last part can be less than 1 MB. When uploading parts, configure the PartNumber of each part and calculate its corresponding CRC64 value. After the parts are successfully uploaded, check the returned CRC64 value with the locally calculated value to verify the object integrity.

```
#Upload an object in parts where the size of each part is OBJECT_PART_SIZE except
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_crc_64 = c64(body)#Calculate CRC64 locally
response = client.upload_part(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Body=body,
PartNumber=part_number,
UploadId=upload_id,
)
part_crc_64 = response['x-cos-hash-crc64ecma']# CRC64 returned by the server
if local_part_crc64 != part_crc_64:# Data Check
print 'crc64 check FAIL'
exit(-1)
etag = response['ETag']
part_list.append({'ETag' : etag, 'PartNumber' : part_number})
```

5. Complete the multipart upload

After all parts are successfully uploaded, you need to complete the multipart upload. Then, you can verify the CRC64 value returned by COS against that of the local object.

```
#Complete the multipart upload
response = client.complete_multipart_upload(
Bucket='examplebucket-1250000000', #Replace with your own bucket name and APPID
Key='exampleobject', #Key value of the object
UploadId=upload_id,
MultipartUpload={ #Require one-to-one correspondence between ETag and PartNumber
```

```
for each part
'Part' : part_list
},
)
crc64ecma = response['x-cos-hash-crc64ecma']
if crc64ecma != local_crc64:# Data Check
print 'check crc64 Failed'
exit(-1)
```

Java SDK

You are advised to use advanced APIs of the Java SDK to upload objects. For more information, please see Object Operations.

Calculating CRC64 locally

```
String calculateCrc64(File localFile) throws IOException {
CRC64 crc64 = new CRC64();
try (FileInputStream stream = new FileInputStream(localFile)) {
byte[] b = new byte[1024 * 1024];
while (true) {
final int read = stream.read(b);
if (read <= 0) {
break;
}
crc64.update(b, read);
}
}
return Long.toUnsignedString(crc64.getValue());
}
```

Getting CRC64 of a COS file and verifying it with the local one

```
// For more information about how to create COSClient, see [Getting Started](http
s://intl.cloud.tencent.com.cn/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：2020-05-29 10:57:55

Environment Dependencies

- HADOOP-COS and Hadoop-COS-Java-SDK (included in the `dep` directory of HADOOP-COS)
- Druid version: Druid-0.12.1

Download and Installation

Downloading HADOOP-COS

Download HADOOP-COS on Github.

Installing HADOOP-COS

Druid-hdfs-extension is required if Druid uses COS for Deep Storage.
After downloading HADOOP-COS, copy the version you want displayed as `hadoop-cos-2.x.x.jar` under the `dep` directory to the Druid installation path `extensions/druid-hdfs-storage` and the `hadoop-dependencies/hadoop-client/2.x.x`. Since Druid accesses COS using HDFS plugin, the version you selected needs to be the same as that of the HDFS plugin.

Directions

Modifying configuration

1. Modify the file `conf/druid/_common/common.runtime.properties· under Druid installation path, add the extension of hdfs to ·druid.extensions.loadList·, specify hdfs as Druid's deep storage, and enter the path of cosn:

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

2. Create a hdfs configuration file `hdfs-site.xml` under the directory `conf/druid/_common/`, and enter your COS keys and other information:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>fs.cosn.userinfo.secretId</name>
<value>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 supported items for the above configuration are exactly the same as those described in the HADOOP-COS official documentation. For more information, see HADOOP Tool.

Getting started

After the Druid processes are started in turn, the Druid data can be loaded into the COS.

Importing/Exporting COS Using DataX

Last updated：2022-04-19 14:23:43

Environmental Dependencies

- HADOOP-COS and the corresponding cos_api-bundle.
- DataX version: DataX 3.0

Download and Installation

Downloading HADOOP-COS

Download HADOOP-COS and the corresponding cos_api-bundle on Github.

Downloading DataX package

Download DataX on Github.

Installing HADOOP-COS

After HADOOP-COS is downloaded, copy `hadoop-cos-2.x.x-${version}.jar` and `cos_api-bundle-${version}.jar` to the Datax decompression paths `plugin/reader/hdfsreader/libs/` and `plugin/writer/hdfswriter/libs/` .

How to Use

DataX configuration

Modifying datax.py script

Open the `bin/datax.py` script in the DataX decompression directory, and modify the CLASS_PATH variable in the script as follows:

```
CLASS_PATH = ("%s/lib/*:%s/plugin/reader/hdfsreader/libs/*:%s/plugin/writer/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"
}
}
}]
}
}
```

Notes:

- Configure `hadoopConfig` as required for cosn.
- Use `defaultFS` to specify the cosn path, e.g. `cosn://examplebucket-1250000000/` .
- In `fs.cosn.userinfo.region` , enter the region where your bucket resides, such as `ap-beijing` . For more information, see Regions and Access Endpoints.
- For `COS_SECRETID` and `COS_SECRETKEY` , use your own COS key information.

The other fields can be the same as those for hdfs.

Migrating data

Save the configuration file as `hdfs_job.json` in the `job` directory by running

```
bin/datax.py job/hdfs_job.json
```

The resulting output is as shown below:

```
2020-03-09 16:49:59.543 [job-0] INFO JobContainer -
[total cpu info] =>
averageCpu | maxDeltaCpu | minDeltaCpu
-1.00% | -1.00% | -1.00%

[total gc info] =>
NAME | totalGCCount | maxDeltaGCCount | minDeltaGCCount | totalGCTime | maxDeltaG
```

```
CTime | 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
Job end time : 2020-03-09 16:49:48
Job duration : 11s
Average job traffic : 3B/s
Recorded write speed : 0rec/s
Recorded read count : 2
Read/Write failure count : 0
```

Configuring COSN for CDH

Last updated：2022-06-23 14:07:20

Overview

CDH (Cloudera's distribution, including Apache Hadoop) is one of the most popular Hadoop distributions in the industry. This document describes how to use the COSN storage service in a CDH environment to separate big data computing from storage.

> Note：
> COSN refers to the Hadoop-COS file system.

Currently, the support for big data modules by COSN is as follows:

Module	Supported	Service Module to Restart
YARN	Yes	NodeManager
YARN	Yes	NodeManager
Hive	Yes	HiveServer and HiveMetastore
Spark	Yes	NodeManager
Sqoop	Yes	NodeManager
Presto	Yes	HiveServer, HiveMetastore, and Presto
Flink	Yes	None
Impala	Yes	None
EMR	Yes	None
Self-built component	To be supported in the future	No
HBase	Not recommended	None

Versions

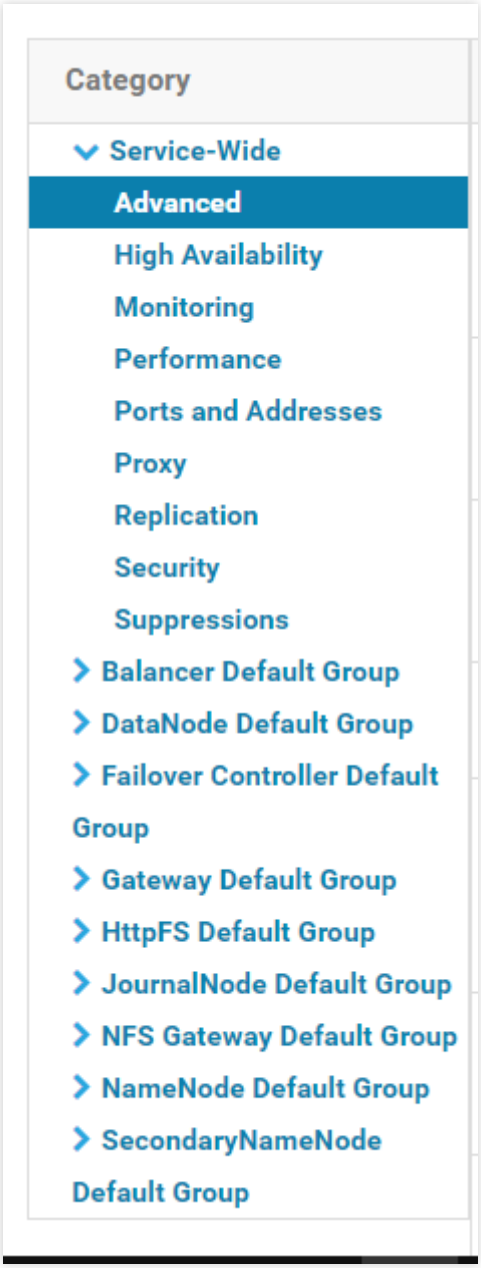
This example uses software versions as follows:

- CDH 5.16.1
- Hadoop 2.6.0

How to Use

Configuring storage environment

1. Log in to the CDH management page.
2. On the homepage, select **Configuration** > **Service-Wide** > **Advanced** as shown below:



3. Specify your COSN settings in the configuration snippet `Cluster-wide Advanced Configuration Snippet(Safety Valve) for core-site.xml` .

```
<property>
<name>fs.cosn.userinfo.secretId</name>
<value>AK***</value>
</property>
<property>
<name>fs.cosn.userinfo.secretKey</name>
<value></value>
</property>
<property>
<name>fs.cosn.impl</name>
<value>org.apache.hadoop.fs.CosFileSystem</value>
</property>
<property>
<name>fs.AbstractFileSystem.cosn.impl</name>
<value>org.apache.hadoop.fs.CosN</value>
</property>
<property>
<name>fs.cosn.bucket.region</name>
<value>ap-shanghai</value>
</property>
```

The following lists the required COSN settings (to be added to `core-site.xml`). For other settings, see Hadoop.

COSN Configuration Item	Value	Description
fs.cosn.userinfo.secretId	AKxxxx	API key information of the account
fs.cosn.userinfo.secretKey	Wpxxxx	API key information of the account
fs.cosn.bucket.region	ap-shanghai	Bucket region
fs.cosn.impl	org.apache.hadoop.fs.CosFileSystem	The implementation class of COSN f FileSystem, which is fixed at `org.apache.hadoop.fs.CosFileSyster
fs.AbstractFileSystem.cosn.impl	org.apache.hadoop.fs.CosN	The implementation class of COSN f 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 JAR file needs to be put in the same location on each server in the cluster.

Data migration

Use Hadoop Distcp to migrate your data from CDH HDFS to COSN. For details, see Migrating Data Between HDFS and COS.

Using COSN for big data suites

1. MapReduce

Directions

(1) Configure HDFS settings as instructed in Data migration and put the JAR file of the COSN SDK in the correct HDFS directory.
(2) On the CDH homepage, find YARN and restart the NodeManager service (recommended). You can choose not to restart it for the TeraGen command, but must restart it for the TeraSort command because of the internal business logic.

Sample

The example below shows TeraGen and TeraSort in Hadoop standard test:

```
hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar teragen -Dmapred.job.maps=500 -D
fs.cosn.upload.buffer=mapped_disk -Dfs.cosn.upload.buffer.size=-1 1099 cosn://exa
mplebucket-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=4194
304 -Dfs.cosn.read.ahead.queue.size=32 cosn://examplebucket-1250000000/terasortv1
/1k-input cosn://examplebucket-1250000000/terasortv1/1k-output
```

Note：
`cosn:// Replace the content behind` schema` with your own bucket path

2. Hive

2.1 MR engine

Directions

(1) Configure HDFS settings as instructed in Data migration and put the JAR file of the COSN SDK in the correct HDFS directory.
(2) On the CDH homepage, find Hive and restart the HiveServer2 and HiveMetastore roles.

Sample

To query your actual business data, use the Hive command line to create a location as a partitioned table on CHDFS:

```
CREATE TABLE `report.report_o2o_pid_credit_detail_grant_daily`(
`cal_dt` string,
`change_time` string,
`merchant_id` bigint,
`store_id` bigint,
`store_name` string,
`wid` string,
`member_id` bigint,
`meber_card` string,
`nickname` string,
`name` string,
`gender` string,
`birthday` string,
`city` string,
`mobile` string,
`credit_grant` bigint,
`change_reason` string,
`available_point` bigint,
`date_time` string,
`channel_type` bigint,
`point_flow_id` bigint)
PARTITIONED BY (
`topicdate` string)
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
'cosn://examplebucket-1250000000/user/hive/warehouse/report.db/report_o2o_pid_cre
dit_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 output is as shown below:

```
hive> select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
Query ID = root_20200713121818_3a911a0c-2496-4e7e-b59d-b2a26266a6ab
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1594351711155_0014, Tracking URL = http://hadoop01:8088/prox
y/application_1594351711155_0014/
Kill Command = /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/hadoop/bin
/hadoop job  -kill job_1594351711155_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-07-13 12:18:19,189 Stage-1 map = 0%,  reduce = 0%
2020-07-13 12:18:25,391 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.89 s
ec
2020-07-13 12:18:30,544 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 10.8
 sec
MapReduce Total cumulative CPU time: 10 seconds 800 msec
Ended Job = job_1594351711155_0014
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.8 sec   HDFS Read: 17383
HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 800 msec
OK
27677
Time taken: 19.128 seconds, Fetched: 1 row(s)
```

2.2 Tez engine

You need to import the COSN JAR file as part of a Tez tar.gz file. The following example uses apache-tez.0.8.5:

Directions

(1) Locate and decompress the Tez tar.gz file installed in the CDH cluster, e.g., /usr/local/service/tez/tez-0.8.5.tar.gz.

(2) Put the COSN JAR file in the resulting directory, and then compress it into a new tar.gz file.

(3) Upload this new file to the path as specified by tez.lib.uris, or simply replace the existing file with the same name.

(4) On the CDH homepage, find Hive and restart HiveServer and HiveMetaStore.

3. Spark

Directions

(1) Configure HDFS settings as instructed in Data migration and put the JAR file of the COSN SDK in the correct HDFS directory.
(2) Restart NodeManager.

Sample

The following takes the `Spark example word count` test conducted with COSN as an example.

```
spark-submit --class org.apache.spark.examples.JavaWordCount --executor-memory 4g
--executor-cores 4 ./spark-examples-1.6.0-cdh5.16.1-hadoop2.6.0-cdh5.16.1.jar cos
n://examplebucket-1250000000/wordcount
```

The output is as shown below:

4. Sqoop

Directions

(1) Configure HDFS settings as instructed in Data migration and put the JAR file of the COSN SDK in the correct HDFS directory.

(2) Put the JAR file of the COSN SDK in the Sqoop directory, for example, `/opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/sqoop/` .

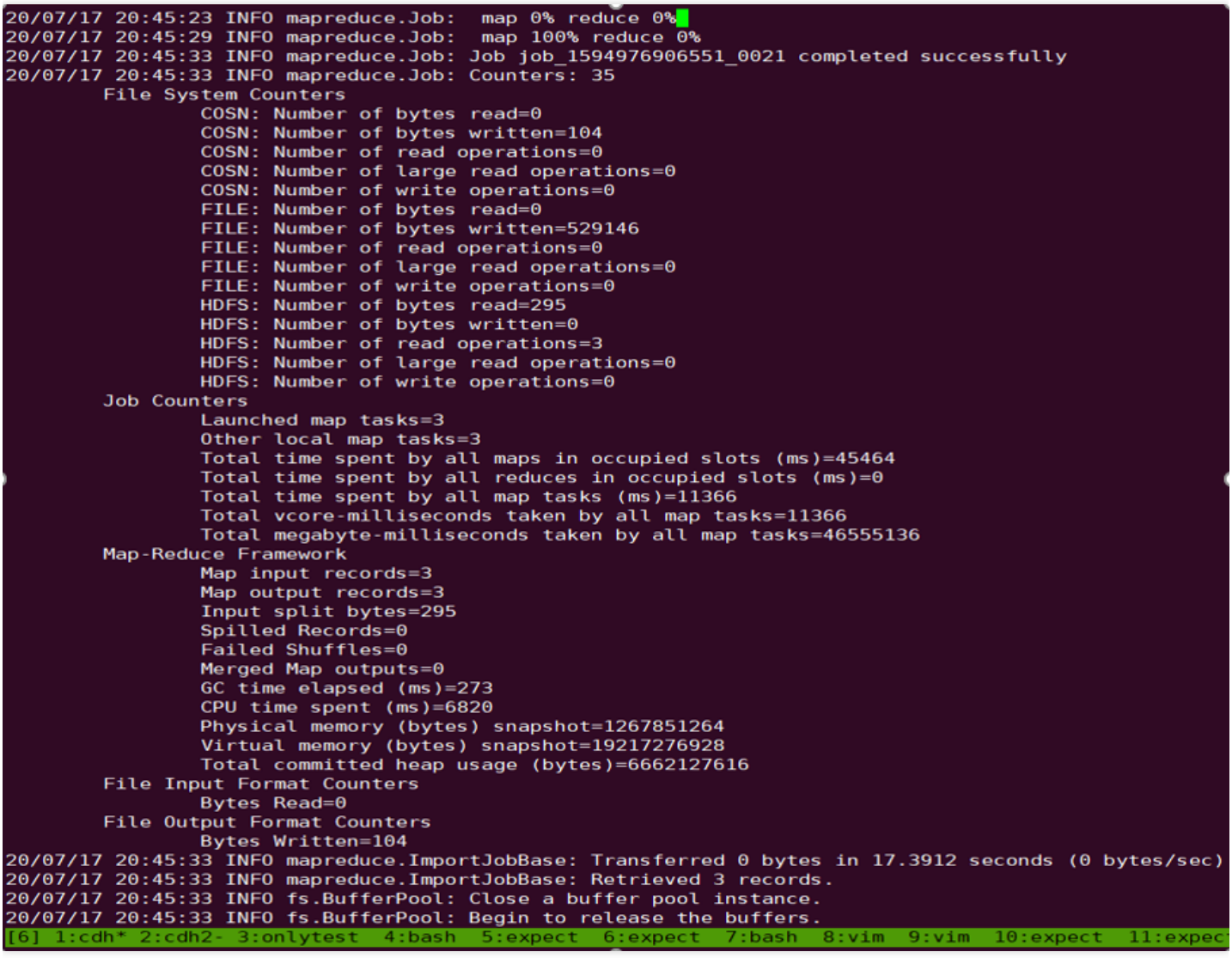
(3) Restart NodeManager.

Sample

For example, to export MySQL tables to COSN, refer to Import/Export of Relational Database and HDFS.

```
sqoop import --connect "jdbc:mysql://IP:PORT/mysql" --table sqoop_test --username
root --password 123** --target-dir cosn://examplebucket-1250000000/sqoop_test
```

The output is as shown below:



5. Presto

Directions

(1) Configure HDFS settings as instructed in Data migration and put the JAR file of the COSN SDK in the correct HDFS directory.

(2) Put the JAR file of the COSN SDK in the Presto directory, for example,

`/usr/local/services/cos_presto/plugin/hive-hadoop2` .

(3) Presto does not load the gson-2...jar JAR file (only used for CHDFS) from Hadoop Common, so you need to manually put it into the presto directory, for example, `/usr/local/services/cos_presto/ plugin/hive-hadoop2` .

(4) Restart HiveServer, HiveMetaStore, and Presto.

Sample

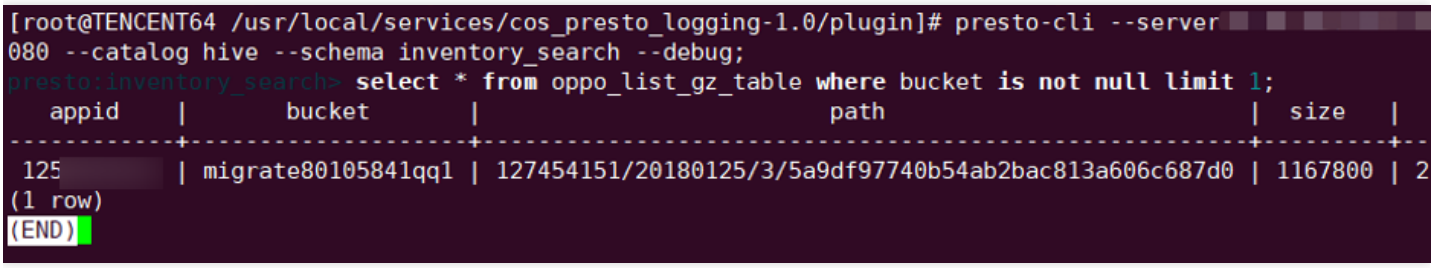
The example below queries the COSN scheme table as a Hive-created location:

```
select * from cosn_test_table where bucket is not null limit 1;
```

Note：

`cosn_test_table` is a table with location as `cosn scheme` .

The output is as shown below:



COS Ranger Permission System Solution

Last updated：2022-11-08 14:36:47

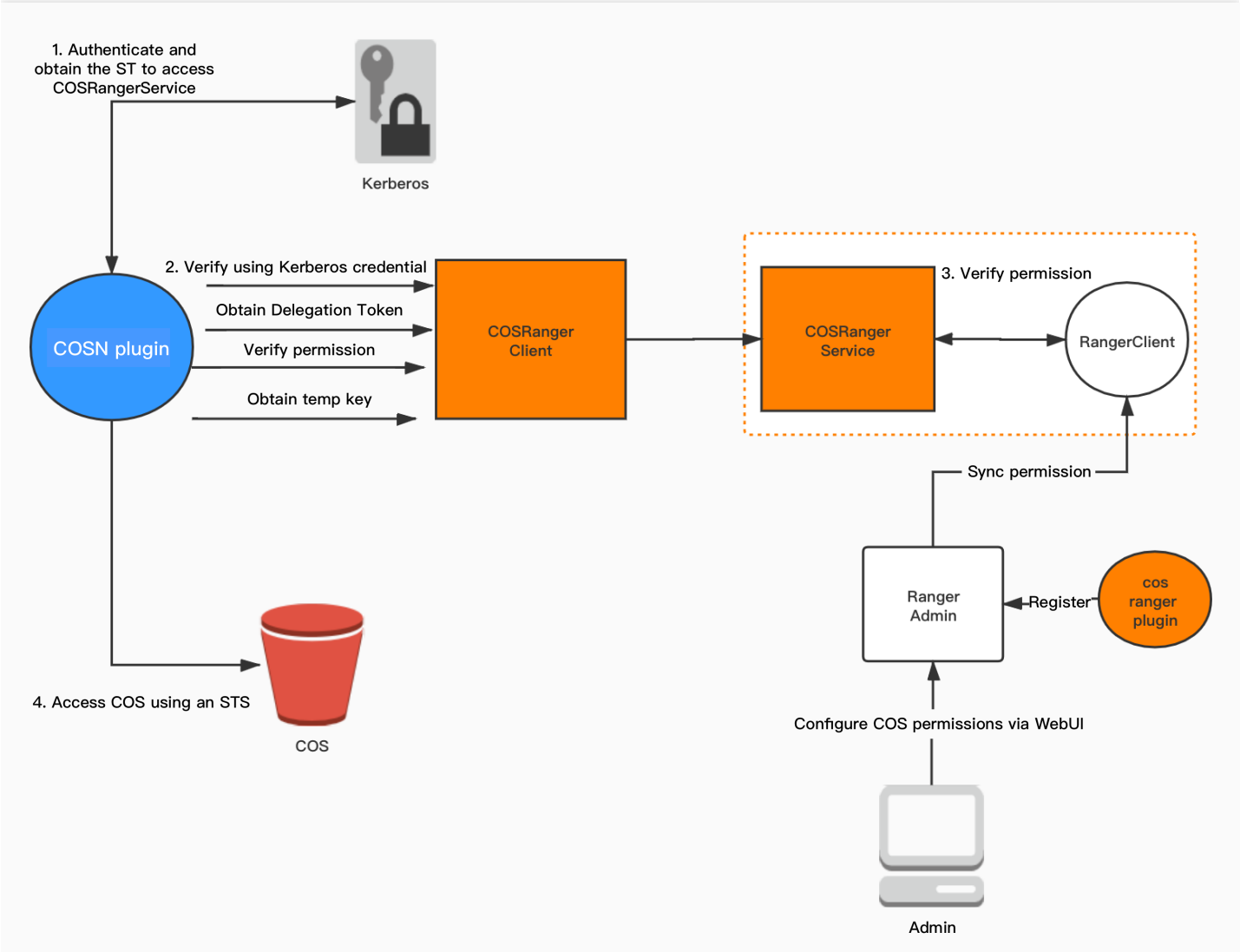
Background

Hadoop Ranger is a permission solution for big data scenarios. By adopting the computing/storage separation mode, you can host data in COS. However, COS uses the CAM permission system, meaning that user roles and permission policies may be different from those of Hadoop Ranger. Therefore, a solution is introduced here to integrate COS with Ranger.

Strengths

- Fine-grained and Hadoop-compatible permission control allow you to centrally manage permissions for big data components and data hosted in the cloud.
- There is no need to set keys in core-site on the plugin side; instead, keys are centrally set in COS Ranger Service to avoid key plaintext exposure.

Solution Architecture



In the Hadoop permission system, authentication is offered by Kerberos and authorized by Ranger. On the basis of this, the following components are provided to support the COS Ranger permission solution:

1. COS Ranger Plugin: As a service defining plugin used on the Ranger server, it provides the COS service description on the Ranger side, including permission types and definitions of required parameters (such as bucket and region). Once it is deployed, you can set permission policies on the Ranger control panel.
2. COS Ranger Service: It integrates the Ranger client, periodically syncs permission policies from the Ranger server, and verifies the permission locally after receiving an authentication request. In addition, it also provides `DelegationToken` generation and renewal APIs in Hadoop, all of which are defined through Hadoop IPC.
3. Cos Ranger Client: It is dynamically loaded by the COSN plugin to forward permission verification requests to COS Ranger Service.

Environment Deployment

- Hadoop environment
- ZooKeeper, Ranger, and Kerberos (if there are authentication requirements)

> Note：
> The components above are open-source and stable. You can install them on your own.

Component Deployment

Deploy components in the following sequence: COS Ranger Plugin, COS Ranger Service, COS Ranger Client, COSN.

- Deploying COS Ranger Plugin
- Deploying COS Ranger Service
- Deploying COS Ranger Client
- Deploying COSN

COS Ranger Plugin extends the service types in the Ranger Admin console. You can set the operation permissions related to COS in the Ranger console.

Code address

You can get the code from the `ranger-plugin` directory at GitHub.

Version

v1.0 or later.

Deployment steps

1. Create a `COS` directory in the service definition directory of Ranger (note: make sure that the directory permissions include at least `x` and `r` permissions).
   a. For an EMR environment, the path is `ranger/ews/webapp/WEB-INF/classes/ranger-plugins`.
   b. For a self-built Hadoop environment, you can find the components connected to the Ranger service through

`find hdfs` in the `ranger` directory in order to find the location of the directory.

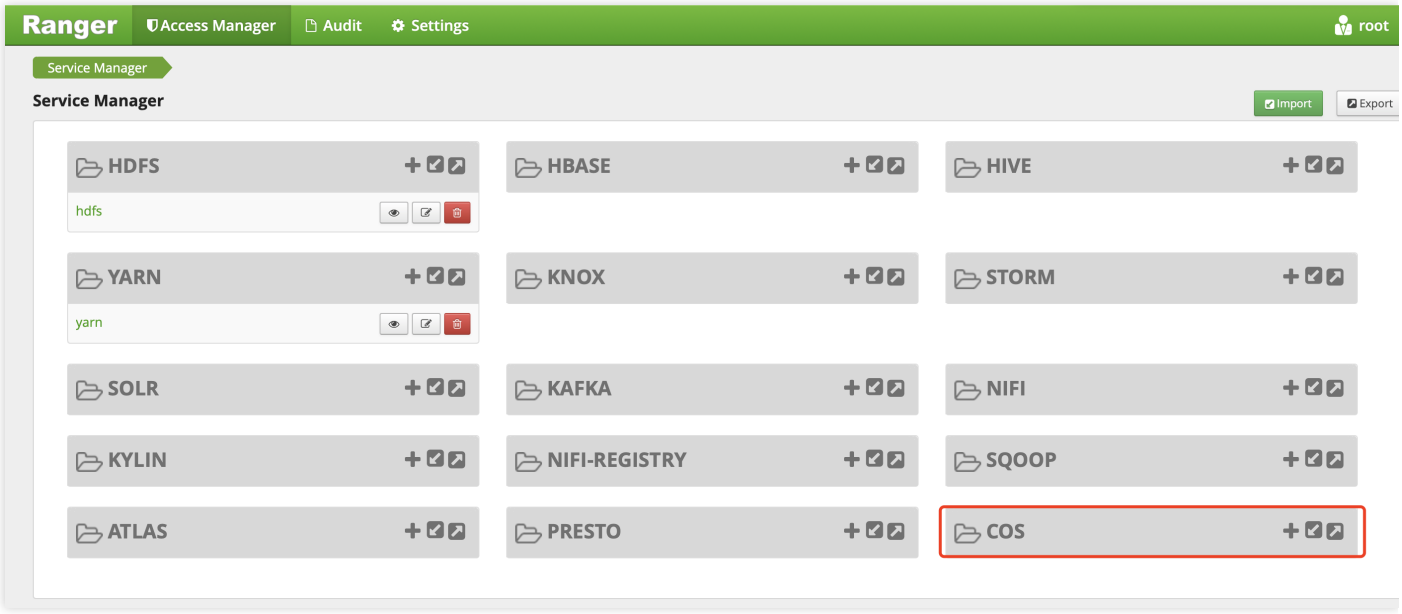
```
[hadoop@10 ranger-plugins]$ ls -l
total 68
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 atlas
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 chdfs
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 cos
drwxr-xr-x 2 hadoop hadoop 4096 Feb 25  2020 hbase
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 hdfs
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 hive
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kafka
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kms
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 knox
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kylin
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 nifi
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 nifi-registry
drwxr-xr-x 2 hadoop hadoop 4096 Aug  5 20:48 presto
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 solr
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 sqoop
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 storm
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 yarn
```

2. Place `cos-chdfs-ranger-plugin-xxx.jar` (with at least the `r` permission) and `cos-ranger.json` files in the COS directory. You can get them from GitHub.
3. Restart Ranger.
4. Register the COS service on Ranger. You can refer to the following command:

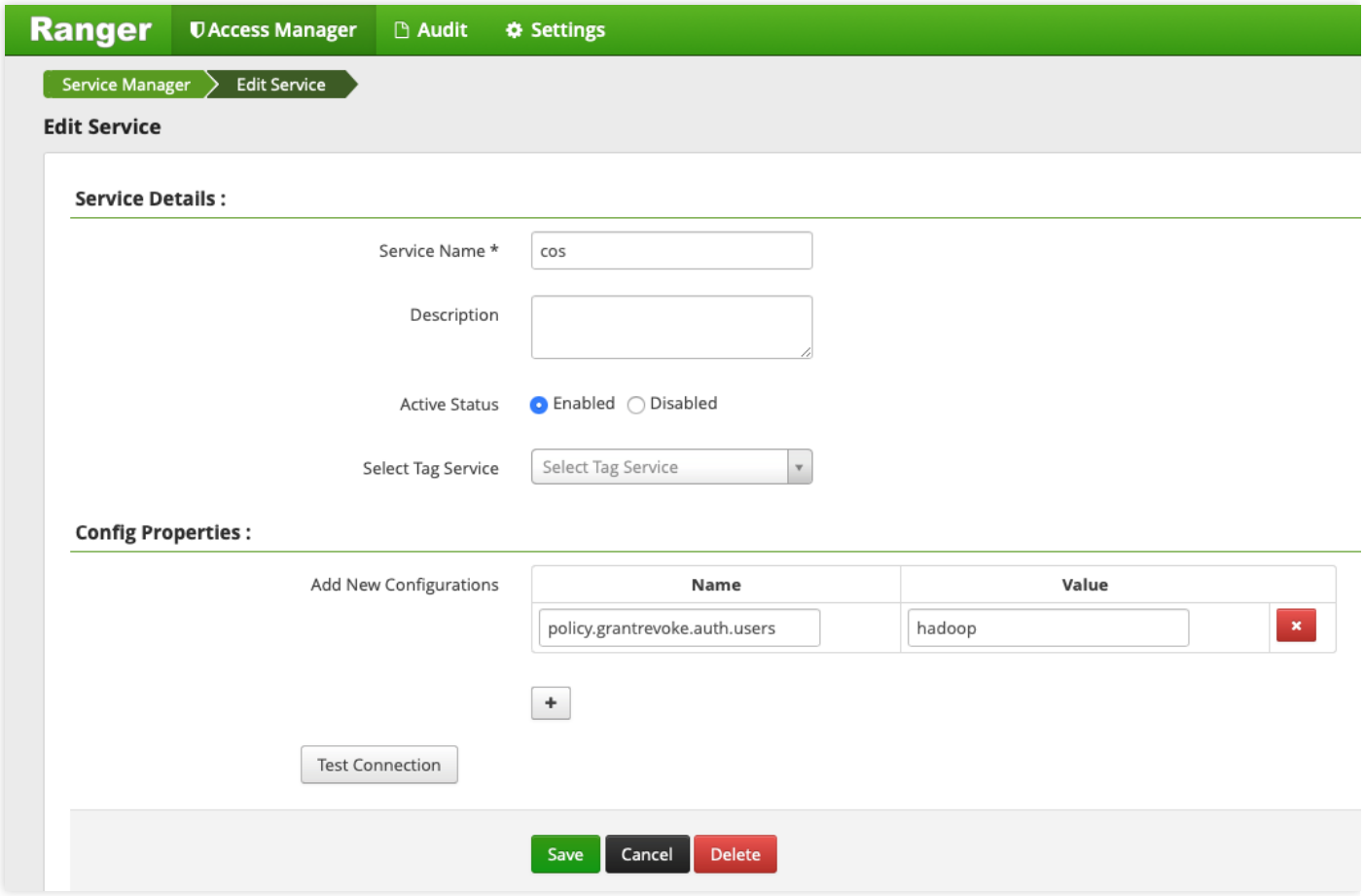
```
## Create the service. The Ranger admin account and password as well as the Ra
nger service address should be specified.
## For an EMR cluster, the admin user is root, and the password is the root pa
ssword set when the EMR cluster is created. Replace the IP of the Ranger servi
ce with the primary node IP of EMR.
adminUser=root
## The password set during EMR cluster creation, which is also the login passw
ord of the Ranger web service.
adminPasswd=xxxxxx
## If the Ranger service has multiple primary nodes, select any of them.
rangerServerAddr=10.0.0.1:6080
## Specify the .json file in step 2 as -d in the command.
curl -v -u${adminUser}:${adminPasswd} -X POST -H "Accept:application/json" -H
"Content-Type:application/json" -d @./cos-ranger.json http://${rangerServerAdd
r}/service/plugins/definitions
## To delete the service just defined, you need to pass in the service ID retu
rned during creation.
serviceId=102
curl -v -u${adminUser}:${adminPasswd} -X DELETE -H "Accept:application/json" -
```

```
H "Content-Type:application/json" http://${rangerServerAddr}/service/plugins/definitions/${serviceId}
```

5. After the service is successfully created, you can see the COS service in the Ranger console.



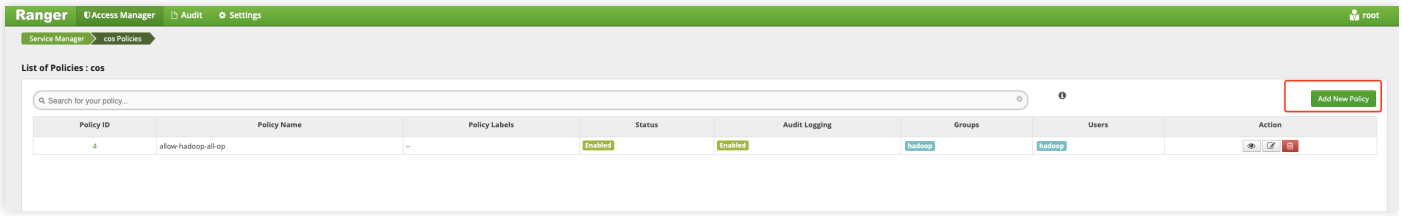
6. Click + next to the COS service to define a new service instance. The service instance name is customizable; for example, you can enter `cos` or `cos_test`. The service configuration is as follows:



You need to set the username subsequently used to start COS Ranger Service (i.e., the user allowed to pull permission policies) as `policy.grantrevoke.auth.users`. We generally recommend you set it to `hadoop`.

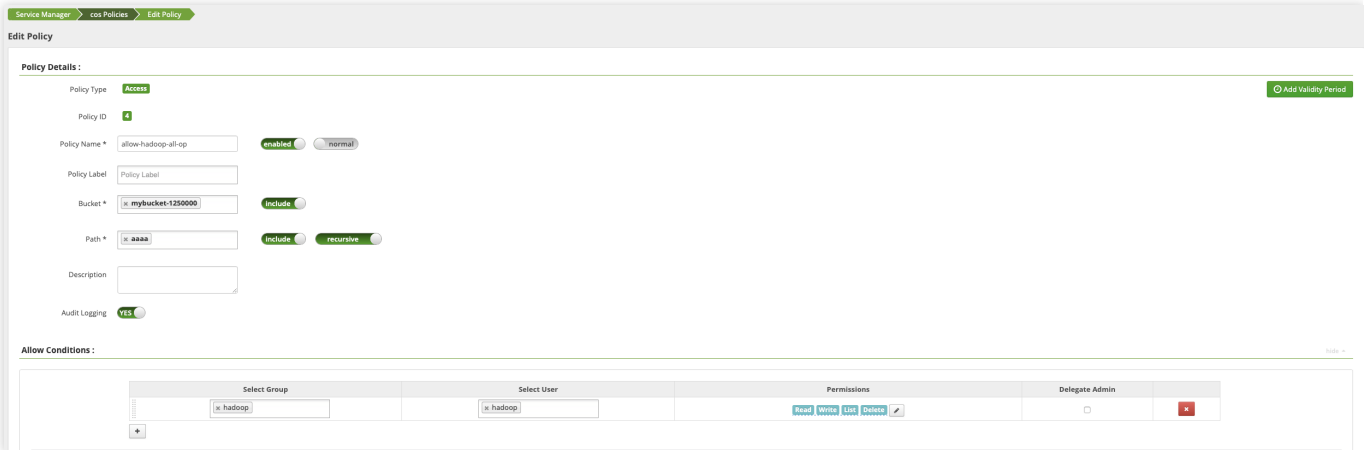
7. Click the newly created COS service instance.

Add a policy.



8. On the page that is displayed, configure the following parameters:

- **Bucket**: Bucket name, such as `examplebucket-1250000000` , which can be viewed in the COS console.
- **Path**: Path of the COS object. Note that it does not start with a slash (/).
  - include: Indicates whether the set permission applies to the specified path itself or other paths except it.
  - recursive: Indicates that the permission applies to not only the specified path but also the subpaths under it (i.e., recursive subpaths). It is usually used when the path is set as a directory.
- **User/Group**: Username and user group in logical OR relationship; that is, the operation is authorized as long as the username or user group condition is met.
- **Permissions**:
  - Read: Read operation, which corresponds to the GET and HEAD operations in COS, such as downloading objects and querying object metadata.
  - Write: Write operation, which corresponds to the PUT operation in COS, such as uploading objects.
  - Delete: Deletion operation, which corresponds to the object deletion operation in COS. To rename a path in Hadoop, you need to have the deletion permission for the original path and write permission for the new path.
  - List: Traversal permission, which corresponds to the `List Object` operation in COS.



Verification

1. Use Hadoop commands to perform operations involving COSN access to check whether the current operations comply with the permissions set by the root account. Below is an example:

```
# Replace the bucket, path, and other information with that of the root accoun
t.
hadoop fs -ls cosn://examplebucket-1250000000/doc
hadoop fs -put ./xxx.txt cosn://examplebucket-1250000000/doc/
hadoop fs -get cosn://examplebucket-1250000000/doc/exampleobject.txt
hadoop fs -rm cosn://examplebucket-1250000000/doc/exampleobject.txt
```

2. Use MR Job for verification. Before the verification, be sure to restart related services such as YARN and Hive.

FAQs

Do I have to install Kerberos?

Kerberos meets the authentication needs. If users in a cluster are trusted, and the purpose of the authentication is only to avoid maloperations performed by unauthorized users, you can skip installing Kerberos and only use Ranger for authentication. As a matter of fact, Kerberos also compromises the performance. Therefore, you can balance your needs for security and performance as needed. If authentication is needed, you can enable Kerberos and then configure COS Ranger Service and COS Ranger Client.

What would happen if I enable Ranger but don't set any policy or no policy is matched?

If no policy is matched, the operation will be denied by default.

Can a sub-account configure the key in COS Ranger Service?

Yes. A sub-account with relevant permissions of the manipulated bucket can generate a temporary key for the COSN plugin to perform corresponding operations. Normally, you can grant all permissions of the bucket to the configured key.

How do I update a temporary key? Do I need to get it from COS Ranger Service every time before I access COS?

The temporary key is cached in the COSN plugin. It will be periodically updated asynchronously.

What should I do if the policy modified on the Ranger page doesn't take effect?

Decrease the `ranger.plugin.cos.policy.pollIntervalMs` value (in milliseconds) in the `ranger-cos-security.xml` file and restart COS Ranger Service. After the policy is tested, we recommend you change it back to the original value (if the time interval is too short, the polling frequency will be high, causing a high CPU utilization).

Connecting Oceanus to COS

Last updated：2022-05-31 15:41:01

Oceanus Overview

Oceanus is a powerful real-time analysis tool in the big data ecosystem. With it, you can easily build various applications in just a few minutes, such as website clickstream analysis, targeted ecommerce recommendation, and IoT. Oceanus is developed based on Apache Flink and provides fully managed cloud services, so you don't need to care about the Ops of infrastructure. It can also be connected to data sources in the cloud for a complete set of supporting services.

Oceanus comes with a convenient console for you to write SQL analysis statements, upload and run custom JAR packages, and manage jobs. Based on the Flink technology, it can achieve a sub-second processing latency in datasets at the petabyte level.

This document describes how to connect Oceanus to COS. Currently, Oceanus is available in the dedicated cluster mode, where you can run various jobs and manage related resources in your own cluster.

Prerequisites

Creating Oceanus cluster

Log in to the Oceanus console and create an Oceanus cluster.

Creating COS bucket

1. Log in to the COS console.
2. Click **Bucket List** on the left sidebar.
3. Click **Create Bucket** to create a bucket as instructed in Creating a Bucket.

> Note：
> When you write data to COS, the Oceanus job must run in the same region as COS.

Directions

Go to the Oceanus console, create an SQL job, and select a cluster in the same region as COS.

1. Create a source

```
CREATE TABLE `random_source` (
f_sequence INT,
f_random INT,
f_random_str VARCHAR
) WITH (
'connector' = 'datagen',
'rows-per-second'='10', -- Number of date rows generated per second
'fields.f_sequence.kind'='random', -- Random number
'fields.f_sequence.min'='1', -- Minimum sequential number
'fields.f_sequence.max'='10', -- Maximum sequential number
'fields.f_random.kind'='random', -- Random number
'fields.f_random.min'='1', -- Minimum random number
'fields.f_random.max'='100', -- Maximum random number
'fields.f_random_str.length'='10' -- Random string length
);
```

Note：

Here, the built-in connector `datagen` is selected. Select a data source based on your actual business needs.

2. Create a sink

```
-- Replace `<bucket name>` and `<folder name>` with your actual bucket and folder names.
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 to which data is to be written
'format' = 'json', --- Format of written data
'sink.rolling-policy.file-size' = '128MB', --- Maximum file size
'sink.rolling-policy.rollover-interval' = '30 min', --- Maximum file write time
'sink.partition-commit.delay' = '1 s', --- Partition commit delay
'sink.partition-commit.policy.kind' = 'success-file' --- Partition commit method
);
```

Note：
For more `WITH` parameters of a sink, see "Filesystem (HDFS/COS)".

3. Configure the business logic

```
INSERT INTO `cos_sink`
SELECT * FROM `random_source`;
```

Note：
This is for demonstration only and has no actual business purposes.

4. Set job parameters

Select `flink-connector-cos` as the **Built-in Connector** and configure the COS URL in **Advanced Parameters** as follows:

```
fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvid
er
fs.cosn.bucket.region: <COS region>
fs.cosn.userinfo.appid: <COS user appid>
```

The job is configured as follows:

- Replace `<cos region="">` with your actual COS region, such as `ap-guangzhou` .
- Replace `<cos user="" appid="">` with your actual `APPID` , which can be viewed in Account Center.

Note：
For more information on job parameter settings, see "File System (HDFS/COS)".

5. Start the job

Click **Save** > **Check Syntax** > **Release Draft**, wait for the SQL job to start, and go to the corresponding COS directory to view the written data.

Custom Processing
Custom Hash Calculation

Last updated：2022-05-12 14:23:49

Overview

Errors may occur when data is being transmitted between the client and the server. Cloud Object Storage (COS) combined with Serverless Cloud Function (SCF) can ensure the integrity of uploaded data through data verification, for example, MD5 verification. When users upload files to COS, SCF will help verify the objects uploaded by the users to ensure the integrity and correctness of the uploaded data.

Background

None of the existing public cloud object storage services in the industry provides MD5 verification, and after a user uploads a file, the following situations may occur:

- Duplicated files and rising costs
- File errors and lower business efficiency
- File missing

Solution Strengths

- Visual operation: one-click configuration simplifies the development process without coding, greatly improving R&D efficiency
- Multiple options: MD5, SHA1, SHA256, and CRC64 are available, meeting user requirements in various scenarios
- Automatic execution: once a file is uploaded to COS, the workflow for calculating the verification code is triggered

Directions

1. Log in to the COS console.

2. create a workflow, customize a format filter rule, and create a custom function node,For detailed steps, see Configuring a Workflow.

3. In the function node pop-up window, click **Add Function**.

4. On the SCF creation page, select the **Calculate COS object hash** template.

5. Based on the user's file size, configure the execution timeout period in the basic settings and configure sufficient memory in the advanced settings.

6. Configure the function code. This function template supports the following two environment variables:

- hashTypeList: indicates the list of calculation algorithms. This variable is optional. The default value is `["crc64", "md5", "sha1", "sha256"]`.
- caseType: indicates the hash case. This variable is optional. The default value is `lowercase`. You can also pass in `uppercase`.

7. Enable permission configuration and bind a role that has the read/write permission of the current bucket. If you need to create an execution role, see Role and Authorization.

8. Click **Finish**.

9. Go back to the previous workflow page, select the custom transcoding function created just now, and save the workflow.

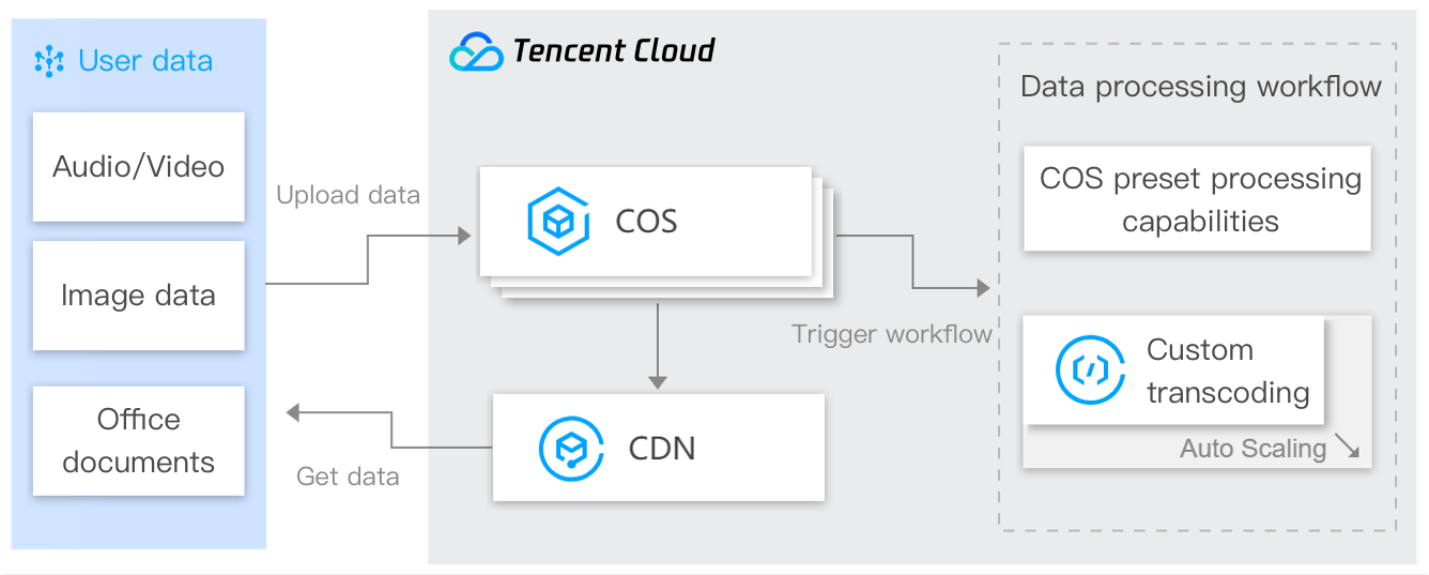
0. Upload the file. After the workflow processing is successful, you can see that multiple hash headers are successfully added to the uploaded file.

Custom Transcoding

Last updated：2021-12-30 16:23:08

Overview

As the largest part of traffic in information dissemination, audio and video are very important in all industries, and the processing logic of audio and video in different business scenarios may be specific to the industry. Although public cloud provides a large number of video processing services for users to choose from, it still cannot fully cover users' requirements for special processes and customization. Using the workflow processing of Cloud Object Storage (COS) combined with the customization logic of Serverless Cloud Function (SCF) is an excellent choice to help users quickly create a variety of audio and video processing businesses to meet their needs.



Use Cases

- Fast access to users' self-built transcoding clusters, compatible with their existing businesses
- Supports industry-specific formats and processing logic, suitable for film,Media and other special industries
- Supports custom processing logic, meeting process customization requirements of various scenarios
- Workflow for template-based batch processing, meeting common audio and video processing requirements of video websites, education, and social networking industries

Solution Strengths

- Accelerated development: eliminates the need to focus on resource OPS and component overhead, greatly reducing the complexity for constructing service structures.
- Reduced overhead: when the platform is idle, no resource is executed. When a function is executed, the service fee is determined by the number of requests and the running time of the computing resource. The price advantage is obvious.
- High availability and scalability: the platform automatically adjusts service resources in parallel based on requests, thus implementing near-infinite scalability and eliminating the risk of service interruption inherent in single-availability zone operations.

Directions

1. Log in to the COS console, create a workflow, customize a filter rule, and create a custom function node.

2. In the function node pop-up window, click **Add Function**.

3. On the function creation page, select the **COS data workflow audio/video transcoding** template.

4. Based on the user's file size, configure the execution timeout duration in the basic settings and configure sufficient memory in the advanced settings.

5. Choose **Advanced Settings** > **Environment Configuration** and configure environment variables. The function template supports the following environment variables:
   - targetBucket: target bucket. Required.
   - targetRegion: region of the target bucket. Required.
   - targetKeyTemplate: target path template. Optional. Defaults to `${InputPath}${InputName}_transcode.${ext}.`
   - ffmpegTemplate: transcoding command template. Required. Example: `${ffmpeg} -loglevel error -i ${source} -r 10 -b:a 32k ${target}.`
   - localTmpPath: temporary save path. When CFS is bound, you can change the temporary path. Optional. Defaults to `/tmp.`

6. Enable permission configuration and bind a role that has the read/write permission of the current bucket. If you need to create an execution role, see Role and Authorization.

7. Click **Finish**.

8. Go back to the previous workflow page, select the custom transcoding function created just now, and save the workflow. Start the workflow on the workflow list page.

9. Upload the file. After the workflow processing is successful, you can see that the uploaded video is successfully transcoded and is saved as a new file.

Using Custom Function to Manage COS Files

Last updated：2022-05-23 14:36:39

Overview

COS workflow provides a series of processing capabilities for media files such as audios, videos, and images. You can flexibly and quickly create a media processing workflow as needed. To satisfy your needs for customization and guarantee the flexibility, COS workflow offers the custom function feature for you to implement custom logic in SCF by configuring function nodes in a workflow.
To make this feature easier to use, COS workflow provides common function feature templates and integrates their creation to node configuration steps, facilitating subsequent processing of source and output files. Currently, supported basic operations include modifying object attributes, moving objects, and deleting objects.

Use Cases

- You want to move or transition source files after media processing to reduce the storage costs.
- You want to tag output files or modify their headers after media processing to facilitate subsequent business use.

Solution Strengths

- Out-of-the-box service: You can use the service after simple configuration, with no need to develop the function logic or care about the complex deployment process.
- Flexible configuration: You can configure nodes of common features as needed to perform different business operations on source and output files.
- Easy extension: You can modify the function logic to meet more customization requirements.

Directions

1. Log in to the COS console.

2. Click **Bucket List** on the left sidebar.

3. Click the target bucket to enter the bucket details page.

4. On the left sidebar, select **Data Processing Workflow** > **Workflow** and click **Create Workflow**.

5. On the workflow creation page, configure the media processing node required by the business such as **Audio/Video Transcoding**. For more information, see Workflow.

6. On the workflow creation page, add the **Custom Function** node and select a required common feature function.

If you haven't created a function of this type, click **Create**.

Create a common feature function as follows:

1. Enter the basic function configuration: Enter the function name prefix, select **Authorize SCF Service**, and click **Next**.

2. Configure attributes: Set the storage class and custom header based on the business needs and click **Next**.

3. Select the object to be processed. You can perform this operation only on the workflow source file.

4. Click **OK**.

5. COS workflow encapsulates processes such as function creation, version release, and alias-based stream switch. Wait for the workflow to be created.

6. After the creation, select the function instance just created and click **OK**.

7. Click **Save**.

Operation Verification

1. Log in to the COS console.

2. Click **Bucket List** on the left sidebar.

3. Click the target bucket to enter the bucket details page.

4. On the left sidebar, select **Data Processing Workflow** > **Workflow** to enter the workflow management page.

5. Enable the workflow just created, go to the specified bucket, upload a media file, and wait for the workflow to be executed.

6. After the workflow execution is completed:
   You can see that media processing succeeded, and the output file was generated.

The storage class and custom header have been set for the source file.

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

Last updated：2022-06-09 12:31:38

Amazon Simple Storage Service (S3) is one of the earliest cloud services launched by AWS. After many years of development, the S3 protocol has become a de facto standard in the object storage field. Tencent Cloud Object Storage (COS) provides an S3-compatible implementation scheme, so you can directly use the COS service in most S3-compatible applications. This document describes how to configure such applications to use COS.

Prerequisites

Checking whether the application can use COS

- An application with `S3 Compatible` in its description can use COS in most cases. If you find that some of its features cannot work properly, contact us for assistance, and be sure to indicate that you followed the steps in this document and provide information such as the application name and screenshots.
- If your application description only states that `Amazon S3` is supported, it means that the application can use the S3 service, but whether it can use COS needs to be further evaluated in relevant configurations as detailed below.

Preparing COS service

Step 1. Sign up for a Tencent Cloud account

(If you already have a Tencent Cloud account, skip this step.)

Click here to sign up for a Tencent Cloud

Step 2. Verify your identity

(If you have already done so, skip this step.)

Click here to verify your identity

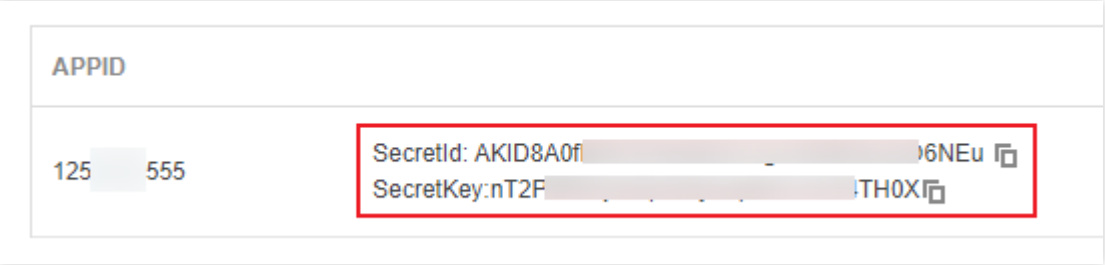
For more information on how to verify your identity, see Identity Verification Guide.

Step 3. Activate the COS service

Click here to activate the COS

Step 4. Prepare the APPID and access key

Get and note down the **APPID**, **SecretId**, and **SecretKey** on the API Key page in the CAM console.



Step 5. Create a bucket

Create a COS bucket as instructed in Creating a Bucket.

Some applications have a built-in process for creating buckets. If you want such applications to create buckets, skip this step.

Configuring COS Service in Application

Basic configuration

Most applications have similar configuration items for using a storage service. The common names and descriptions of these configuration items are as listed below:

> Note：
> If you have any questions during the configuration, contact us for assistance, and be sure to indicate that you followed the steps in this document and provide information such as the application name and screenshots.

Common Configuration Item Name	Description

Common Configuration Item Name	Description
Provider, service provider, storage service provider, storage provider, etc.	Select which storage service the application should use. There may be the following cases: • If an option has text like S3 Compatible Storage/S3 Compatible, then it will be used first. • If an option only has text like Amazon Web Services/AWS/Amazon S3, then use it but pay attention to our further instructions during configuration. • If there is no similar option, but the application description mentions that the application supports S3 services or S3-compatible services, then you can continue with the configuration below, but you also need to pay attention to our further instructions. • In other cases, the application may not be able to use COS.
Service endpoint, service address, service URL, endpoint, custom endpoint, server URL, etc.	This indicates the address of an S3-compatible service. If you use COS, enter the COS service address here in the format of `cos.<Region>.myqcloud.com` or `https://cos.<Region>.myqcloud.com`. Whether `https://` needs to be entered is determined by the application, and you can make some attempts by yourself. Here, `<Region>` indicates the availability region of COS. In the application, you can only create or select a bucket in the region specified in the service address. • For example, if your bucket is in Guangzhou region, the service address should be configured as `cos.ap-guangzhou.myqcloud.com`; otherwise, you cannot find the bucket in Guangzhou in the application. • If only `Amazon S3` can be selected as the application service provider and the service address can be configured, then you can change the service address to the aforementioned `cos.<Region>.myqcloud.com` or `https://cos.<Region>.myqcloud.com`. • If the service address cannot be configured or there is no such configuration item, the application cannot use COS.
Access key, access key ID, etc.	Enter the **SecretId** obtained in step 4.
Secret key, secret, secret access key, etc.	Enter the **SecretKey** obtained in step 4.
Region, etc.	Select "Default", "Auto", or "Automatic".

Common Configuration Item Name	Description
Bucket, etc.	You can select or enter the name of an existing bucket in the format of `<BucketName-APPID>`, such as `examplebucket-1250000000`. Here, `BucketName` is the name you entered when you created the bucket in step 5, and `APPID` is the **`APPID`** obtained in step 4. As described above, the bucket must be in the region specified by the service address, and buckets in other regions will not be listed or cannot work properly. If you need to create a bucket, the name of the new bucket should be in the format of `<BucketName-APPID>` as mentioned above; otherwise, it cannot be properly created.

Other advanced configuration items

In addition to the above basic configuration items, some applications have other advanced configuration items. The following describes some COS features for you to better use COS in such applications.

- Service port and protocol
  COS supports both HTTP and HTTPS protocols, with the default ports 80 and 443 used by default. For security considerations, we recommend you use COS over the HTTPS protocol preferably.
- Path-Style and Virtual Hosted-Style
  COS supports both styles.
- AWS v2 and AWS v4 signatures
  COS supports both signature formats.

Summary

COS does not guarantee full compatibility with S3. If you have any questions when using COS in your application, contact us for assistance, and be sure to indicate that you followed the steps in this document and provide information such as the application name and screenshots.

Storing Remote WordPress Attachments to COS

Last updated：2022-09-01 16:20:54

Overview

WordPress is a blogging platform built with PHP. You can use it to build your own website on a server that supports PHP and MySQL databases, or simply as a content management system (CMS).

WordPress is a powerful, scalable, and easy-to-expand platform with a wide range of plugins. With third-party plugins, it offers everything that a website should have.

This document describes how to use a plugin to store remote attachments from the WordPress media library in COS.

As 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 server.
- Accelerated user access to image attachments through CDN.

Preparations

1. Create a blog website with WordPress.

- You can download the latest version of WordPress and view installation instructions at Get WordPress.

2. 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.

3. Find the bucket you created on the **Bucket List** page and click its name to enter the details page.

4. Click the **Overview** tab on the left sidebar. Then, find and record the endpoint.

Installing and Configuring Plugin

Installing plugin

On the WordPress backend, click **Plugins** > **Add New**. You can install the plugin in one of the following two ways:

- Search for **tencentcloud-cos** directly and install it (recommended).
- Download the latest release of the source code from GitHub, and then upload it on the WordPress backend. You can also upload it directly to the plugin directory `wp-content/plugins` and run it on the backend.

Configuring plugin

1. Click **Tencent Cloud Settings** on the left sidebar of WordPress and configure relevant COS information as follows:

Configuration Item	Value
SecretId and SecretKey	Enter the access key information, which can be created and obtained on the Manage API Key page.
Region	The region selected during bucket creation
Space Name	The bucket name customized during bucket creation, such as `examplebucket-1250000000`
Access Domain Name	The COS bucket's default domain name, which is automatically generated when the bucket is created. Buckets in different regions have different default domain names. To view the default domain name, go to the COS console, click the name of the target bucket, click **Overview**, and find the **Domain Information** section.
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. We recommend you not enable it.
Forbid Thumbnail	After this option is enabled, corresponding thumbnail files will not be uploaded. We recommend you not enable it.
CI	After the CI service is enabled, you can perform various operations on images, such as editing, compression, format conversion, and watermarking. For more information, see Cloud Infinite.
Debugging	This feature logs errors, exceptions, and warnings. You can enable it as needed.

2. After completing the configuration, click **Save Configuration**.

3. Upload a new test file and confirm that its URL points to COS.

> Note：
> Once confirmed, sync your old WordPress resources to the COS bucket by using COSCMD or COS Migration. **This step must be performed; otherwise, you will not be able to view these resources in COS.** Then, you can optionally enable origin-pull as instructed below in Set origin-pull.

More Capabilities

1. Use CDN for access acceleration
   To configure CDN acceleration for your bucket, see Setting CDN Acceleration. You will need to change the URL prefix to the default or custom CDN acceleration domain name in the WordPress plugin settings.
2. Replace resource URLs in the database
   Unless your website is newly created, you will need to use the plugin to replace the old resource URLs in the database with new ones. We recommend you back up your data before replacing for the first time.

- Enter the old domain name for the resource, such as `https://example.com/` .
- Enter the current domain name for the resource, such as `https://img.example.com/` .

3. Configure cross-origin access
   When you reference a resource link in a document, you may receive the following prompt in the console: `No 'Access-Control-Allow-Origin' header is present on the requested resource` . This is most likely because you haven't configured the `HTTP Header` parameter in the CORS settings. You can configure this parameter in one of the following two ways:

- Configure in the COS console

> Note：
> For more information, see Setting CORS.

- Configure in 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. Set origin-pull

If you don't upload resources to the WordPress media library through the COS console, we recommend you use COS origin-pull. For detailed directions, see Setting Origin-Pull.

Once origin-pull is enabled, when a client accesses a source object in COS for the first time, if COS cannot hit the object, it will return a 302 HTTP status code and automatically redirect the request to the origin-pull address. Then, the origin will provide the object for access. Meanwhile, COS will copy this object from the origin and store it in the corresponding COS directory so that COS will be able to directly return the object to the client in subsequent requests.

Backing up Files from PC to COS

Last updated：2022-06-20 12:24:03

Background

Data matters, we all know that. Digital photos, e-documents, work products, saved game…none of them we can easily afford to lose. It will be a huge headache if we lose all of our files due to a disk failure, or a single file due to misoperation, computer shutdown, or software crash, or if we cannot provide a requested “callback version” just because we haven’t saved one. This is why backups are absolutely important.

When it comes to backups, the first idea that occurs to our mind is most probably using a portable hard drive or building a NAS within an individual network so that we can just move files into it. Well, is it really as simple as all that?

In fact, backup involves a lot of work to do. Copy our files onto backup media, check if the backups are correct, and we may have to do both of them regularly in order to minimize lost files. Besides, backup media require maintenance, so we need to replace our hard drives promptly once they are dead.

Given all this, is there an easier way to keep our files safe? The answer is Yes.

As Tencent Cloud’s business grows, it has already developed a suite of enterprise cloud storage services, including COS. Now, we need backup software to connect the files in our PC with cloud storage services. It will help us back up files automatically onto the cloud, and check backup correctness on a regular basis.

Software Introduction

Arq® Backup is commercial backup software for Windows and macOS. Running in the background, it automatically backs up specified directories at intervals you configure. Besides, it retains backup files for each point in time so that you can easily get an old version. This software offers minimal backup size and maximum backup speed by backing up only files that are different from those at the last point in time, and by backing up repeated files across paths only once. It encrypts backups with a password only you know before they leave your computer. Therefore, you can be assured that your sensitive data is well-protected from being stolen during the transfer over Internet or on-cloud storage.

To get a commercial license of Arq® Backup, each user needs to pay $49.99. This software, which is used on a single computer, offers a 30-day free trial which you may want to try before purchase.

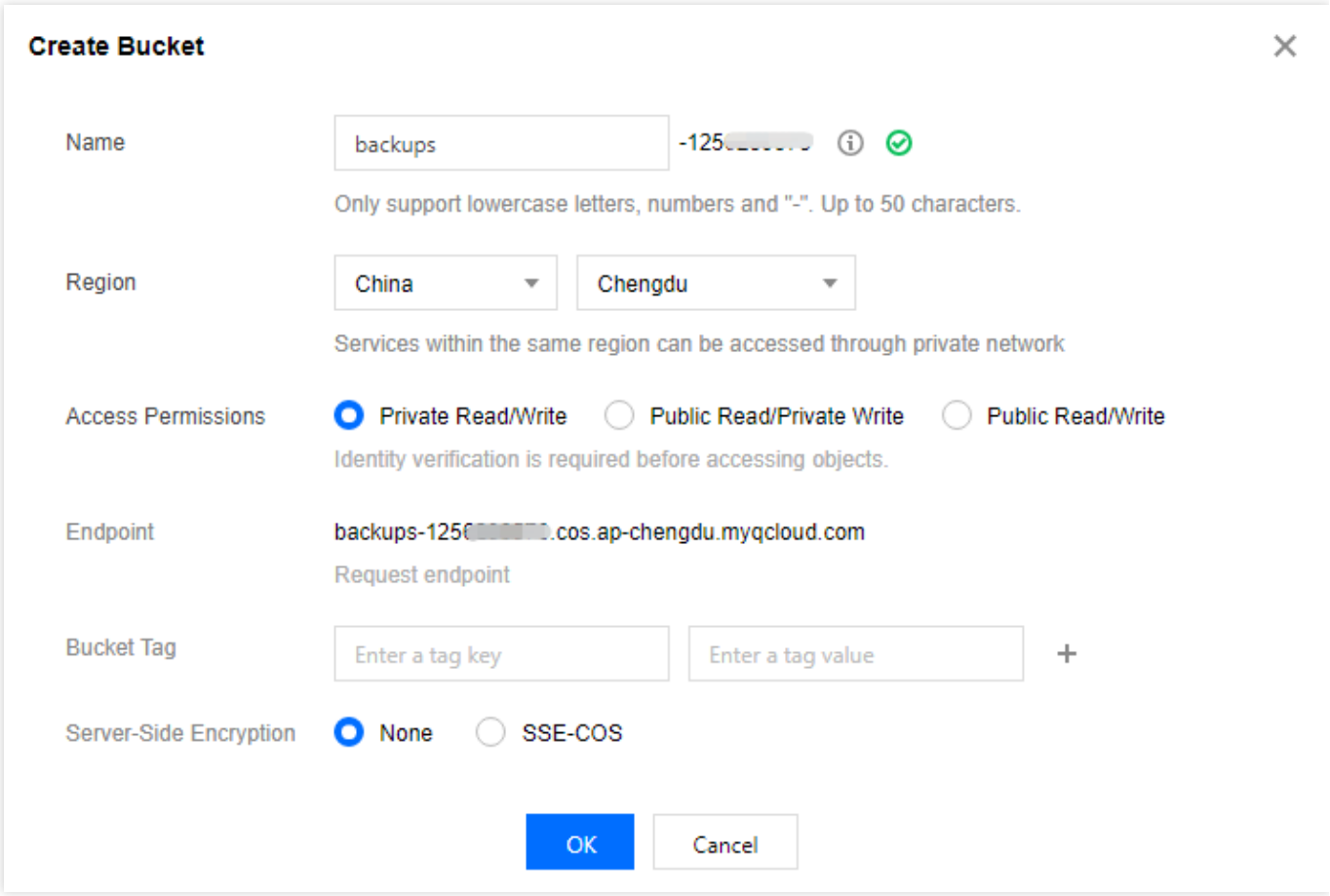
Note：

Arq® Backup currently does not support the simplified Chinese language. You can download, purchase it and read its instructions from the Official Website.

Preparing Tencent Cloud COS

Note：
Please skip Steps 1-2 if you are using COS.

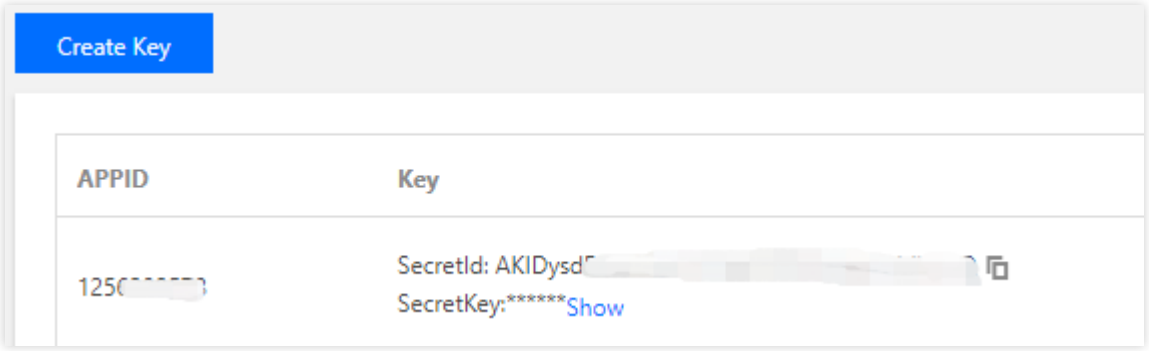
1. Sign up for Tencent Cloud, and complete Identity Verification.
2. Log in to the COS Console, and activate COS service as instructed.
3. In the COS console, click **Bucket List** in the left sidebar first, and then **Create Bucket** to add a new bucket.
   - Name: bucket name, e.g. “backups”.
   - Region: you can choose a region closest to your location. Currently, we offer price discounts for regions in southwest China, so you may alternatively choose “Chengdu” or “Chongqing” to enjoy this offer.



For the other fields, leave the default. Copy and save the **Request endpoint**, and click **OK**.

> Note：
> Now, you have created a bucket. For more information, see Creating a Bucket.

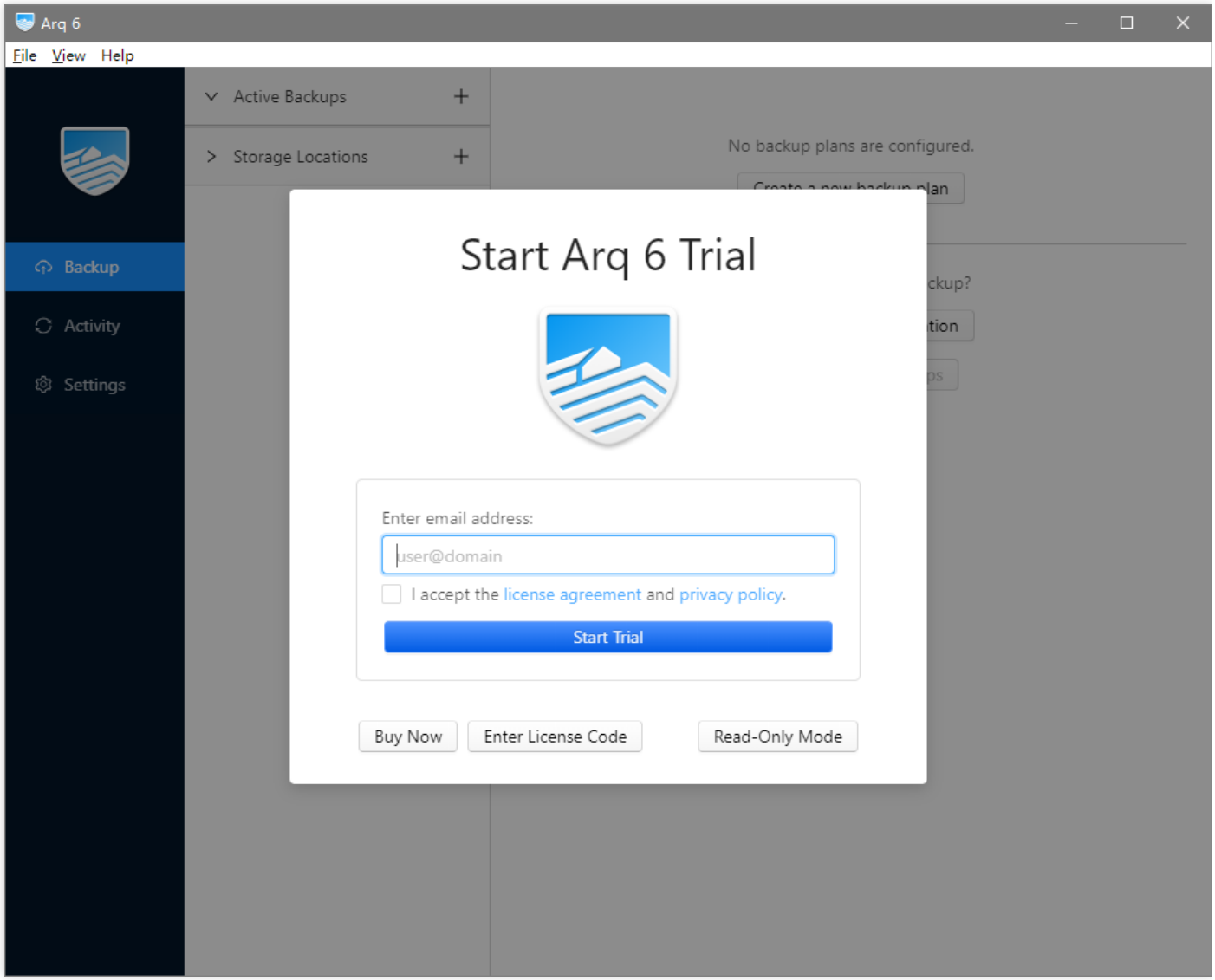
4. Log in to the API Key Management Console, and create and save your SecretId and SecretKey.



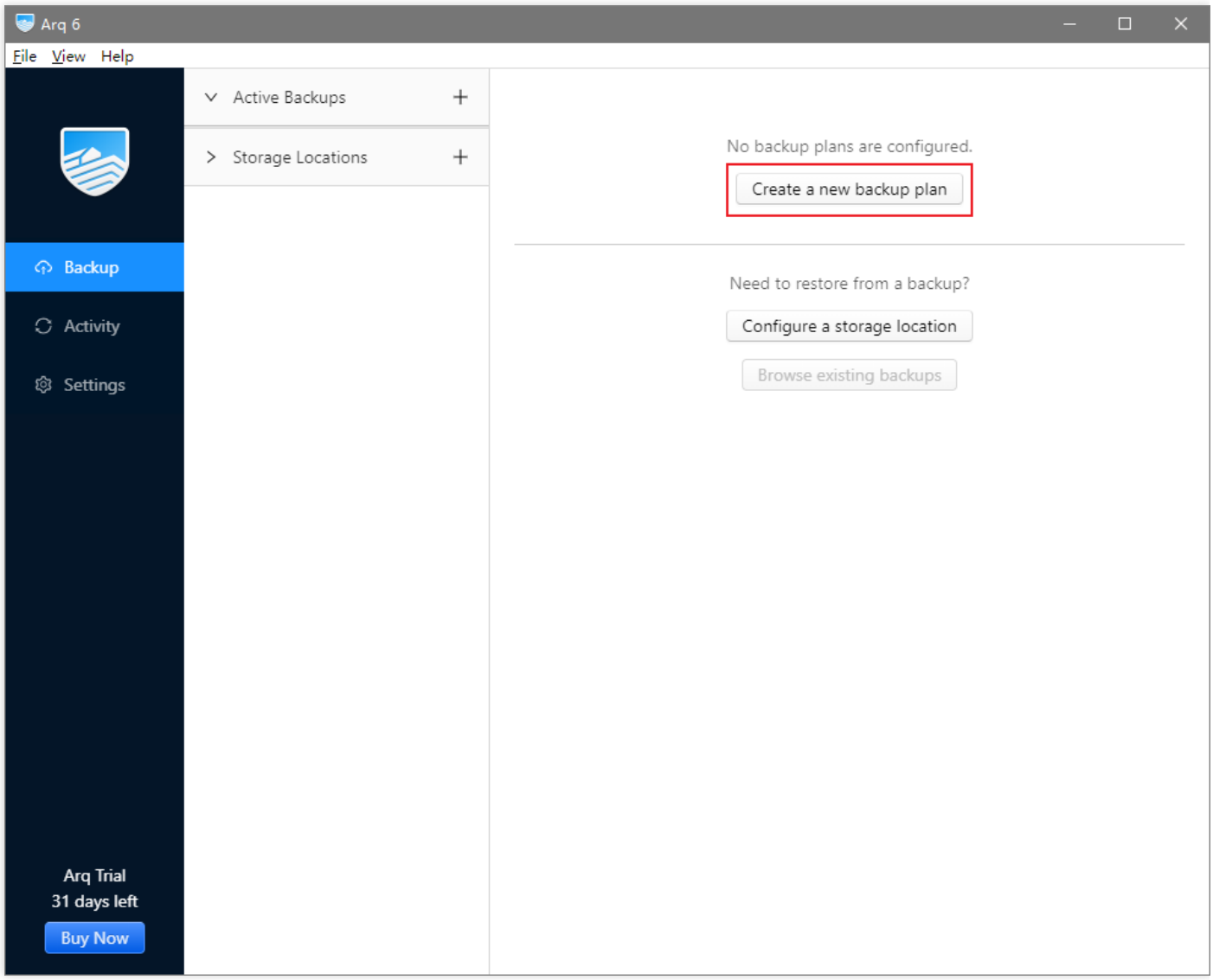
Installing and Configuring Arq® Backup

> Note：
> Take Arq® Backup Version 6.2.11 for Windows as an example.

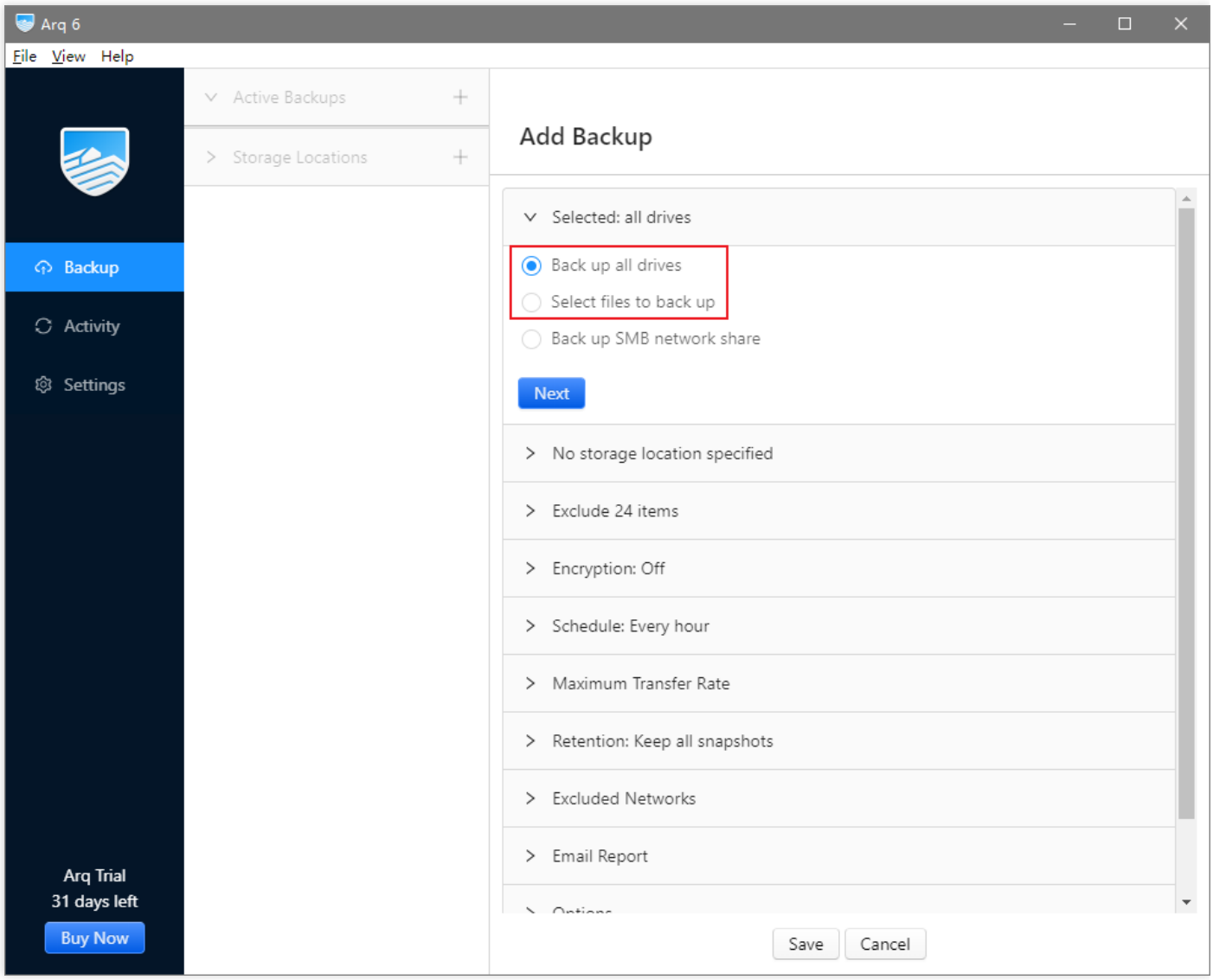
1. Download it from Arq® Backup Website.
2. Follow the wizard to install the software. Once completed, it will start automatically while prompting you to log in. Then, enter your email address and click **Start Trial**.



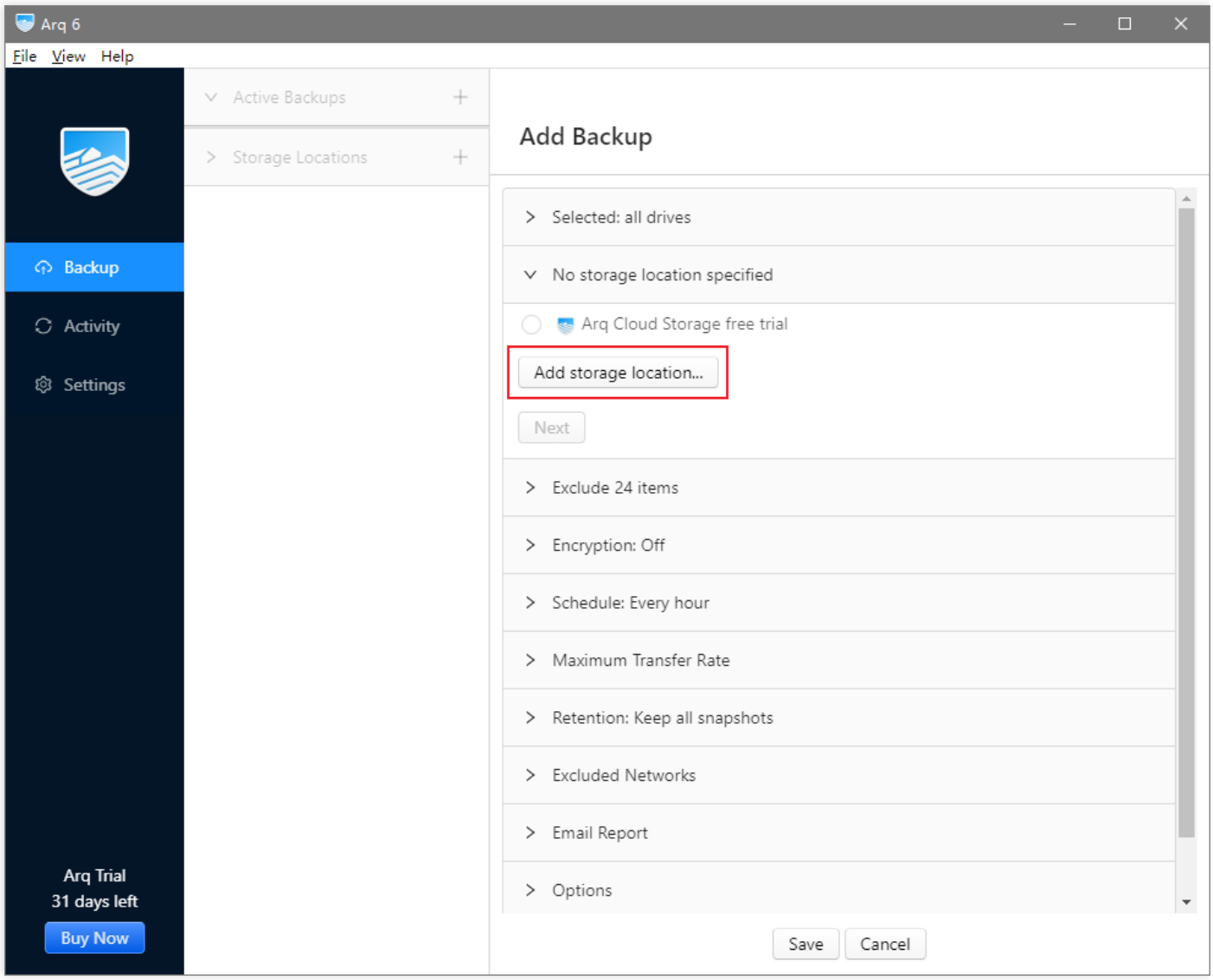
3. In the **Backup** pane, click **Create a new backup plan** to add a backup plan.



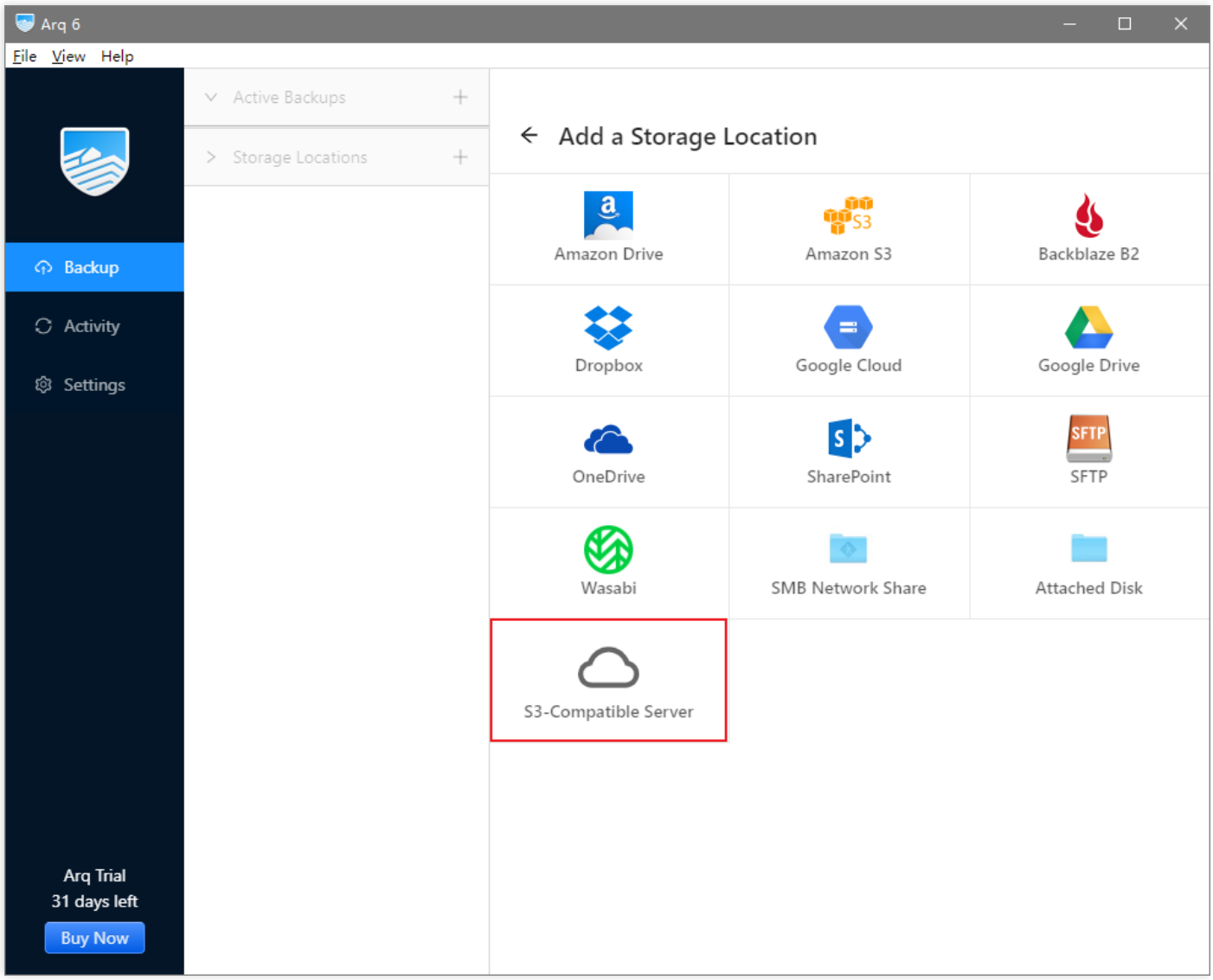
4. In the opened window, select *Back up all drives** or **Select files to back up**.



5. Click **Add storage location** to add a location for storing your backups as shown below:

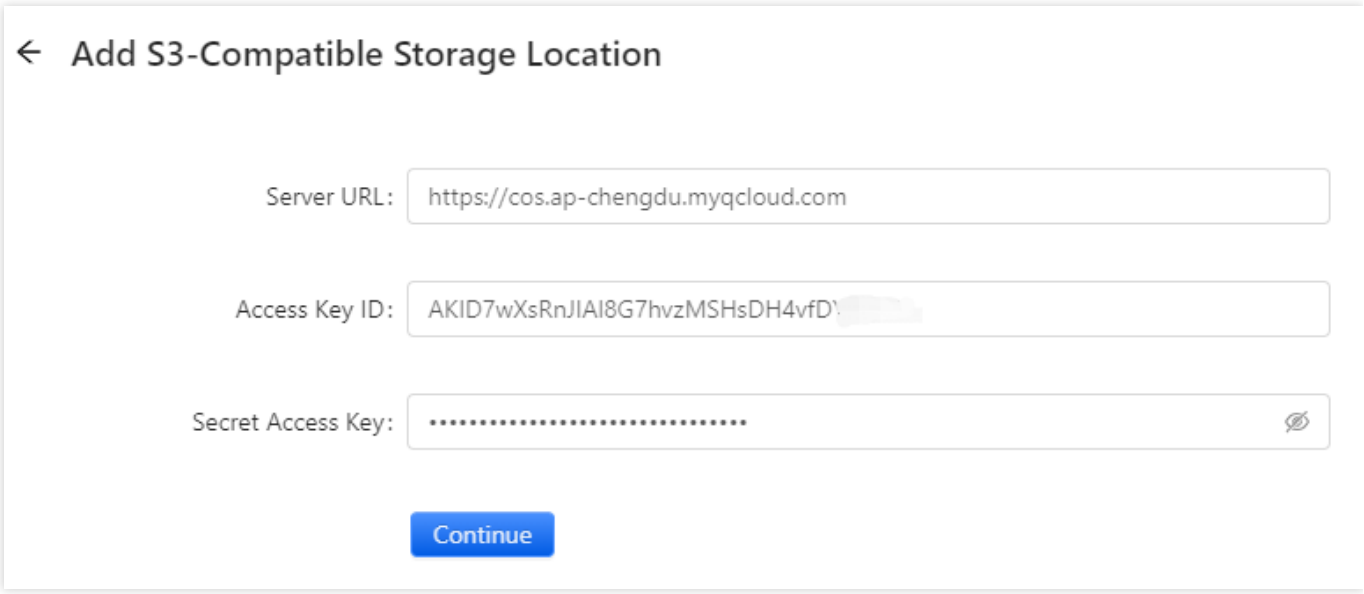


6. In this example, we select **S3-Compatible Server**.

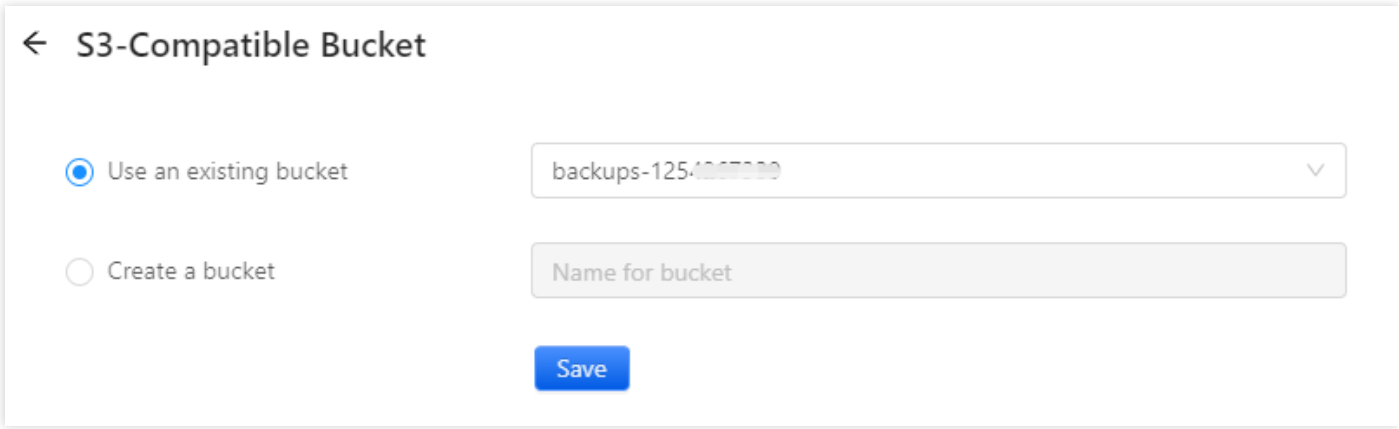


7. In the opened window, configure the following as instructed, and click **Continue**.
   - Server URL: enter the above-mentioned request endpoint, starting from `cos` and prepending `https://` to the beginning of it, such as `https://cos.ap-chengdu.myqcloud.com`. Note that the bucket name is excluded here.
   - Access Key ID: the above-mentioned SecretId.

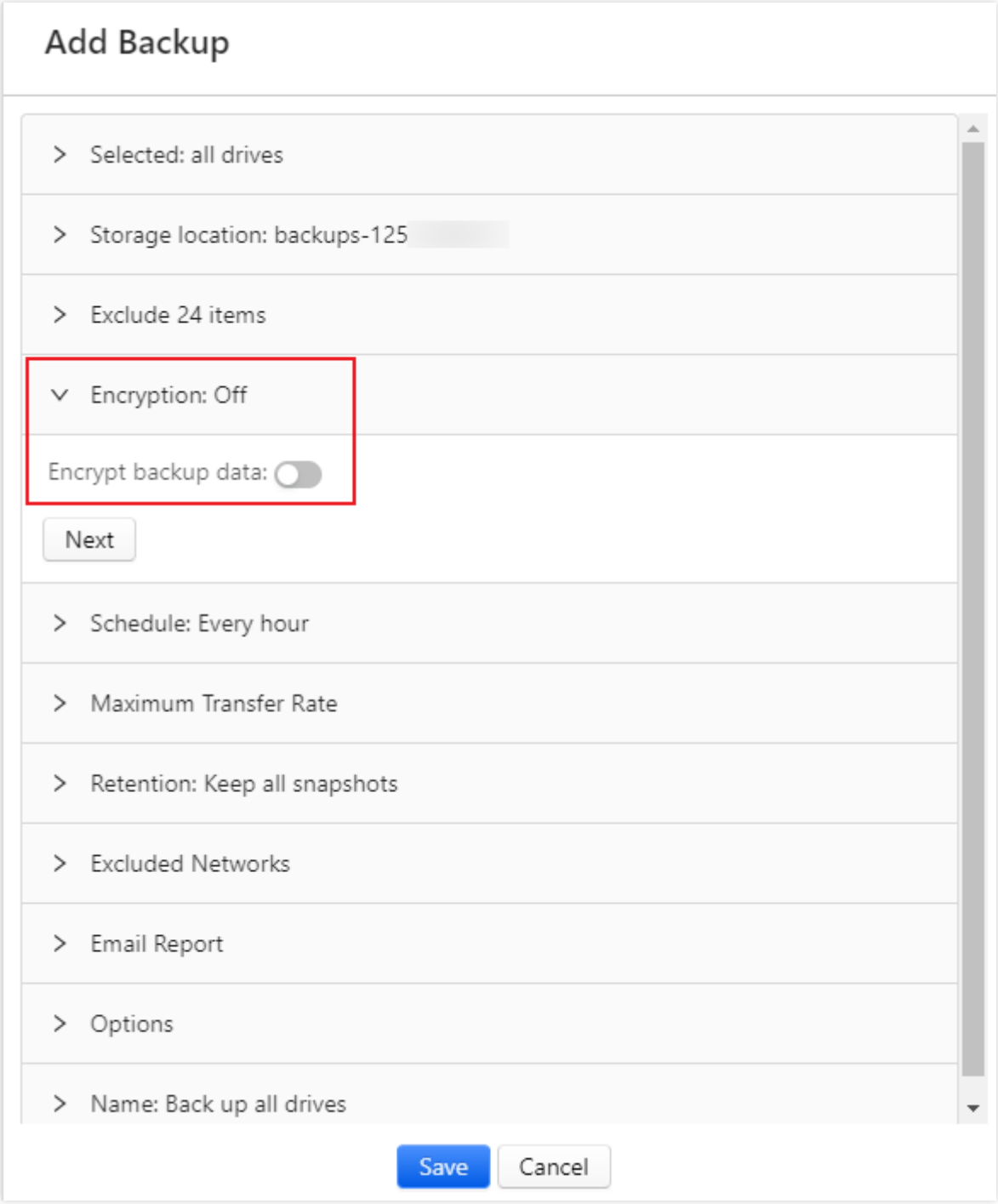
- Secret Access Key: the above-mentioned SecretKey.



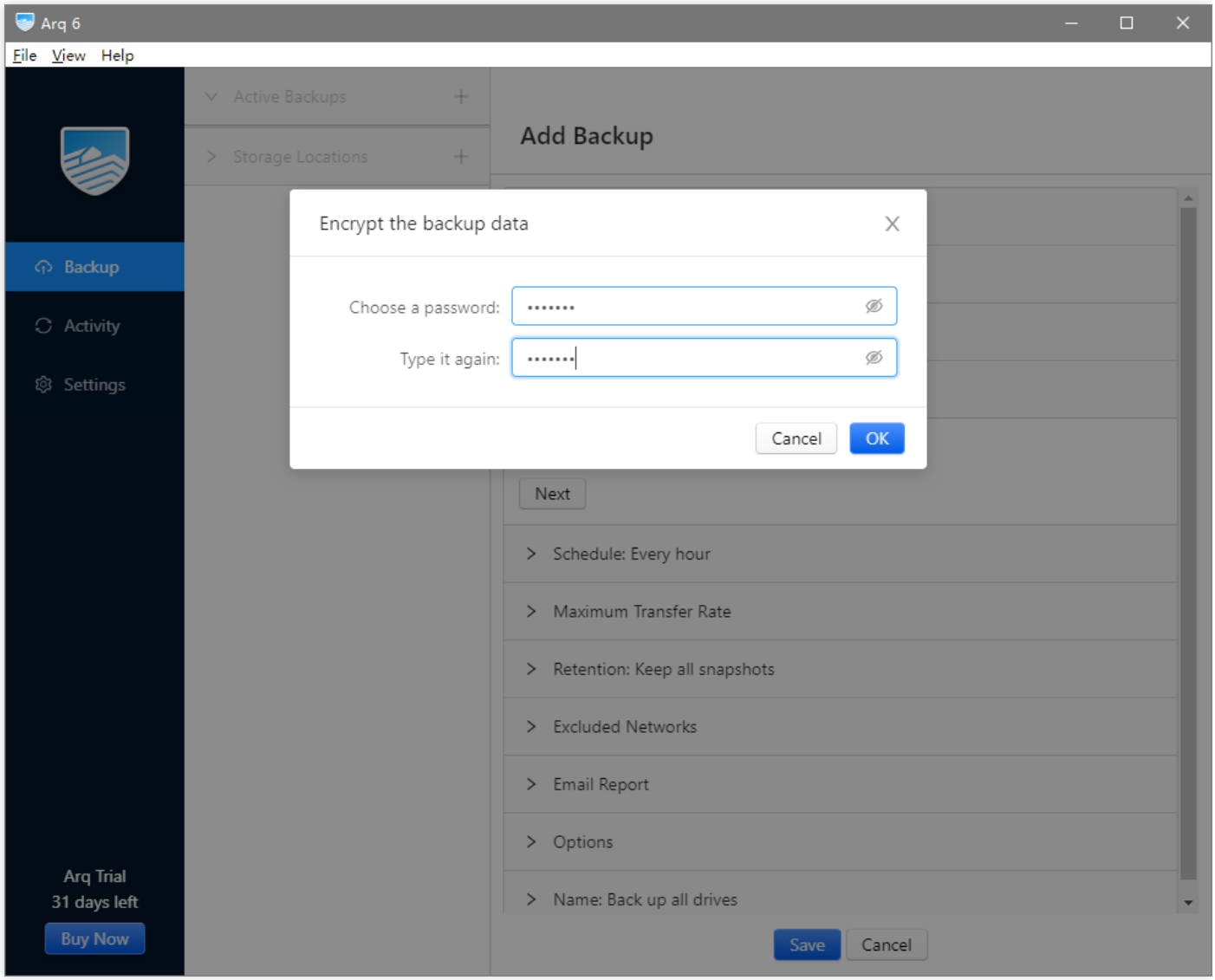
8. In the new window, click *Use an existing bucket*, select the above bucket you created, such as `backups-1250000000`, and click **Save**.



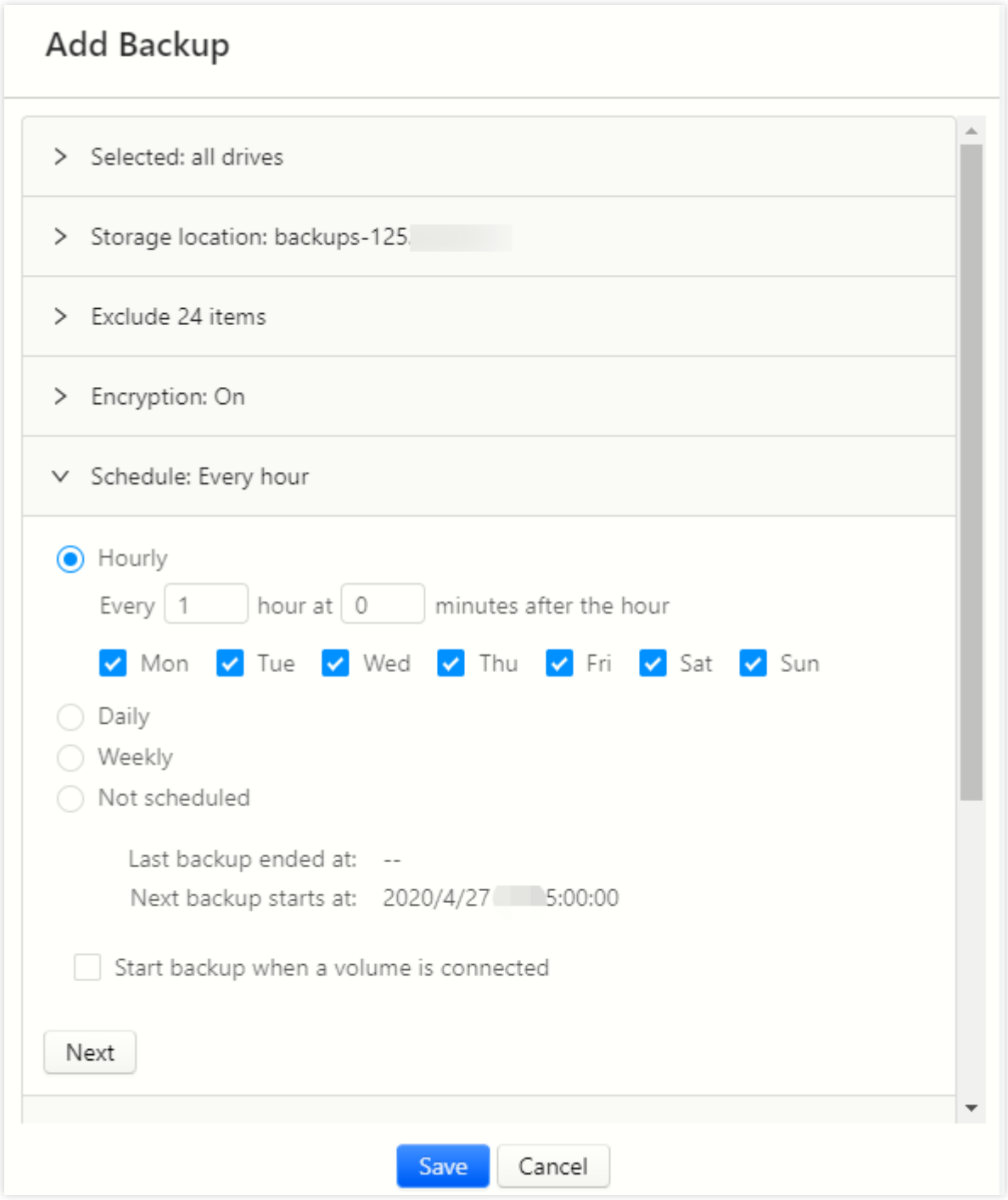
9. (Optional) You may choose to encrypt backup data. Here, we select **On**.



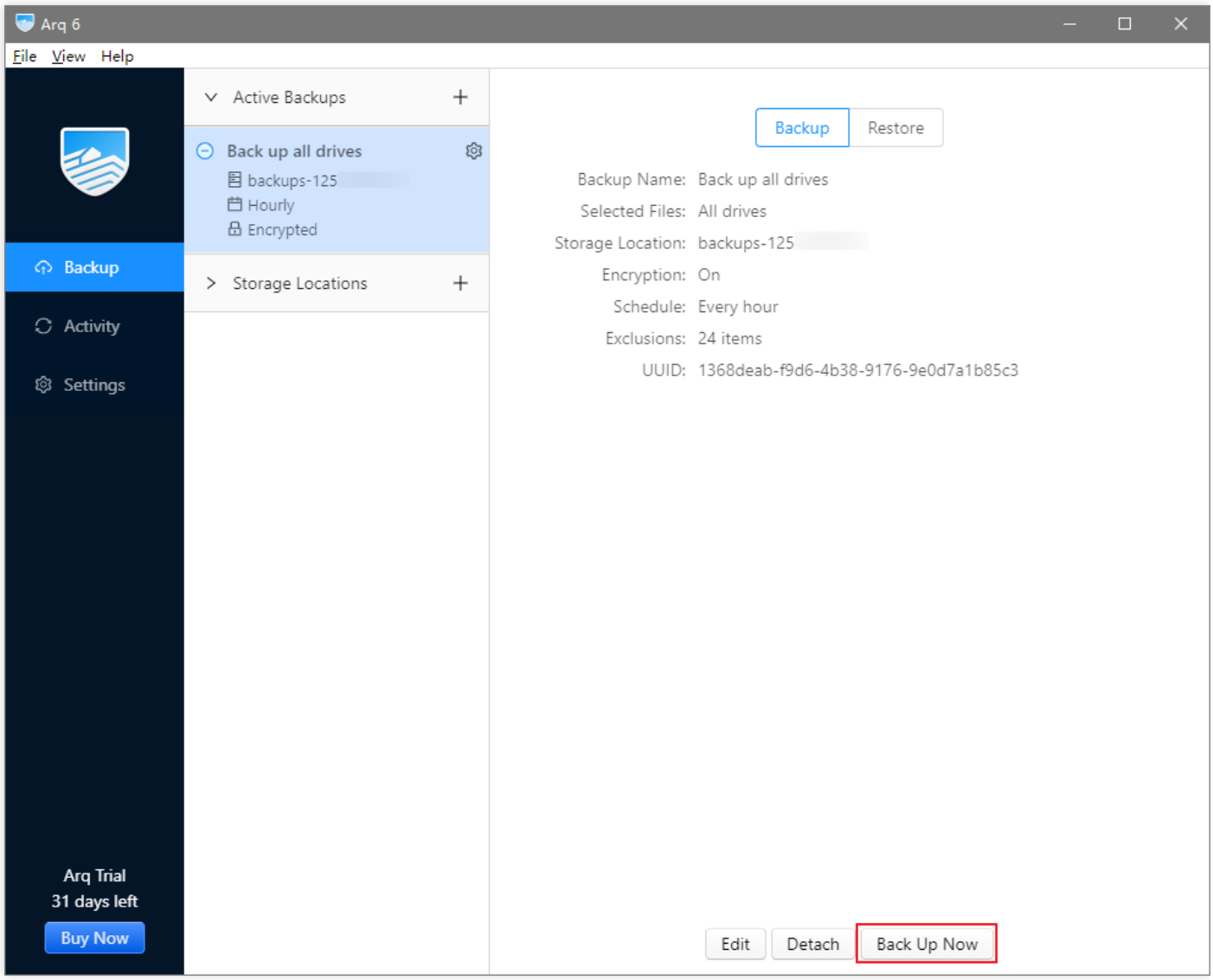
0. In the pop-up window, set your encryption password. Enter it twice and click **OK**. **Please keep your password in mind, otherwise, you may not be able to restore your files from backup.**



1. (Optional) You may configure a backup schedule.

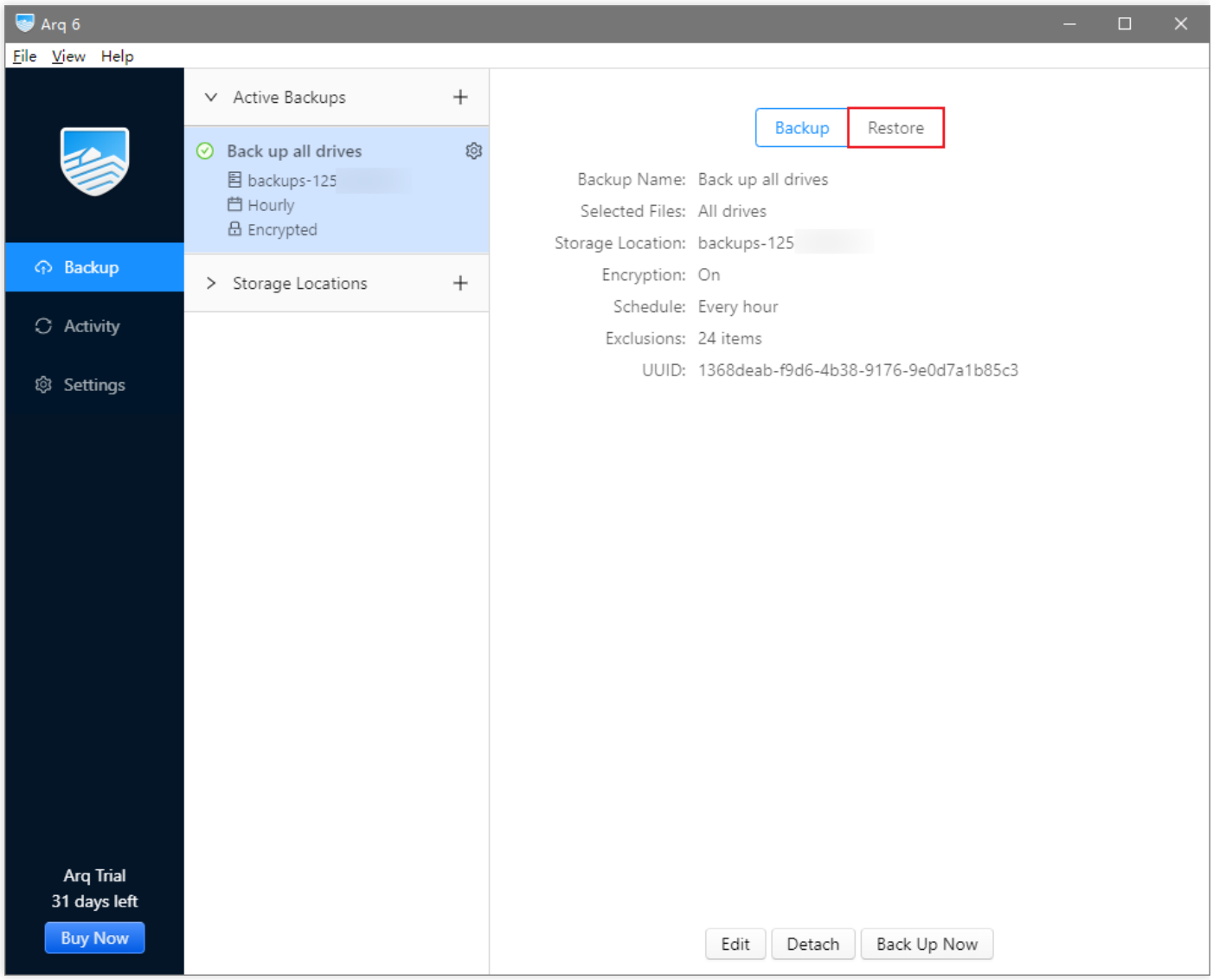


2. Click **Save**, and then **Back Up Now** to start the backup.

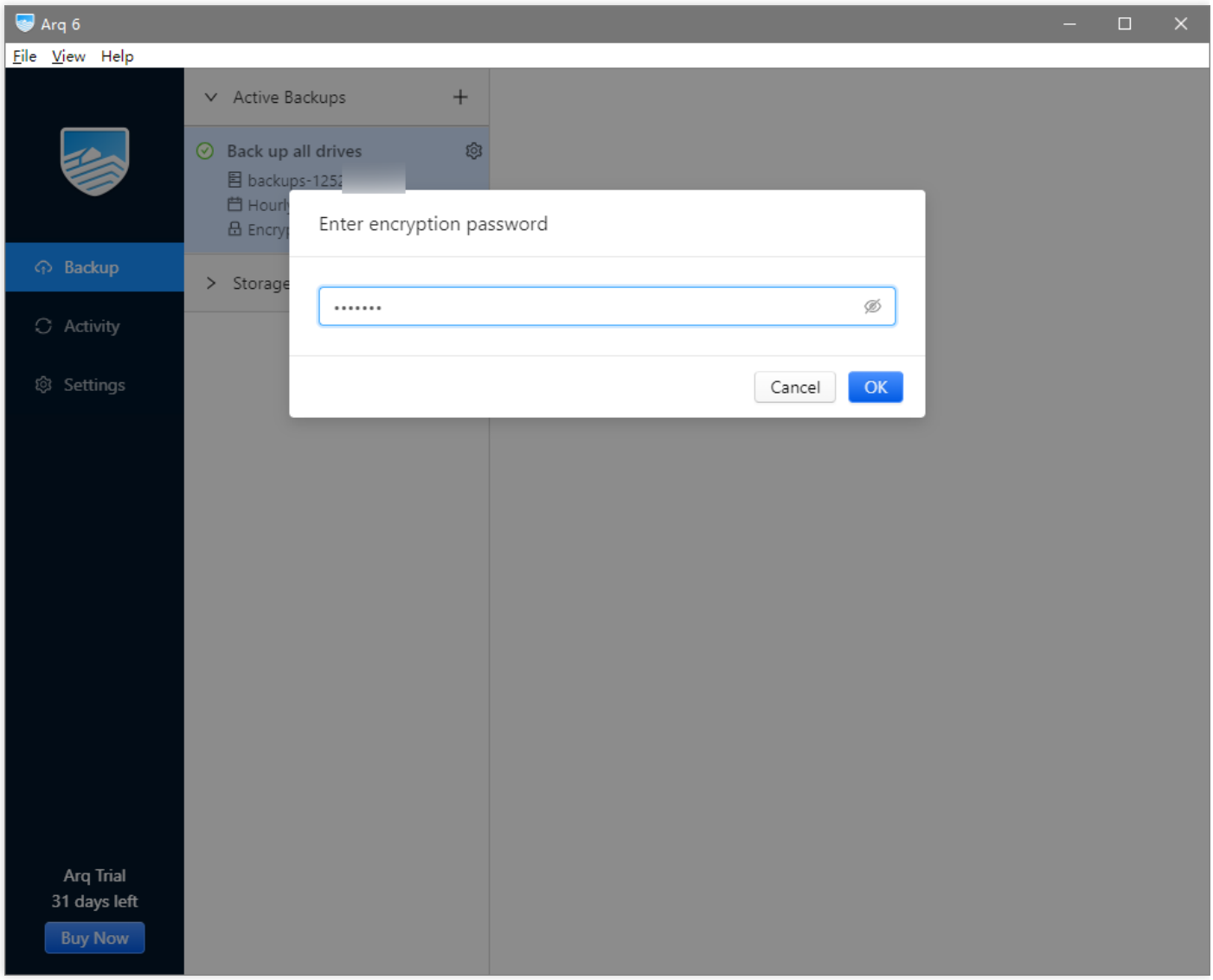


Restoring Files from Backup

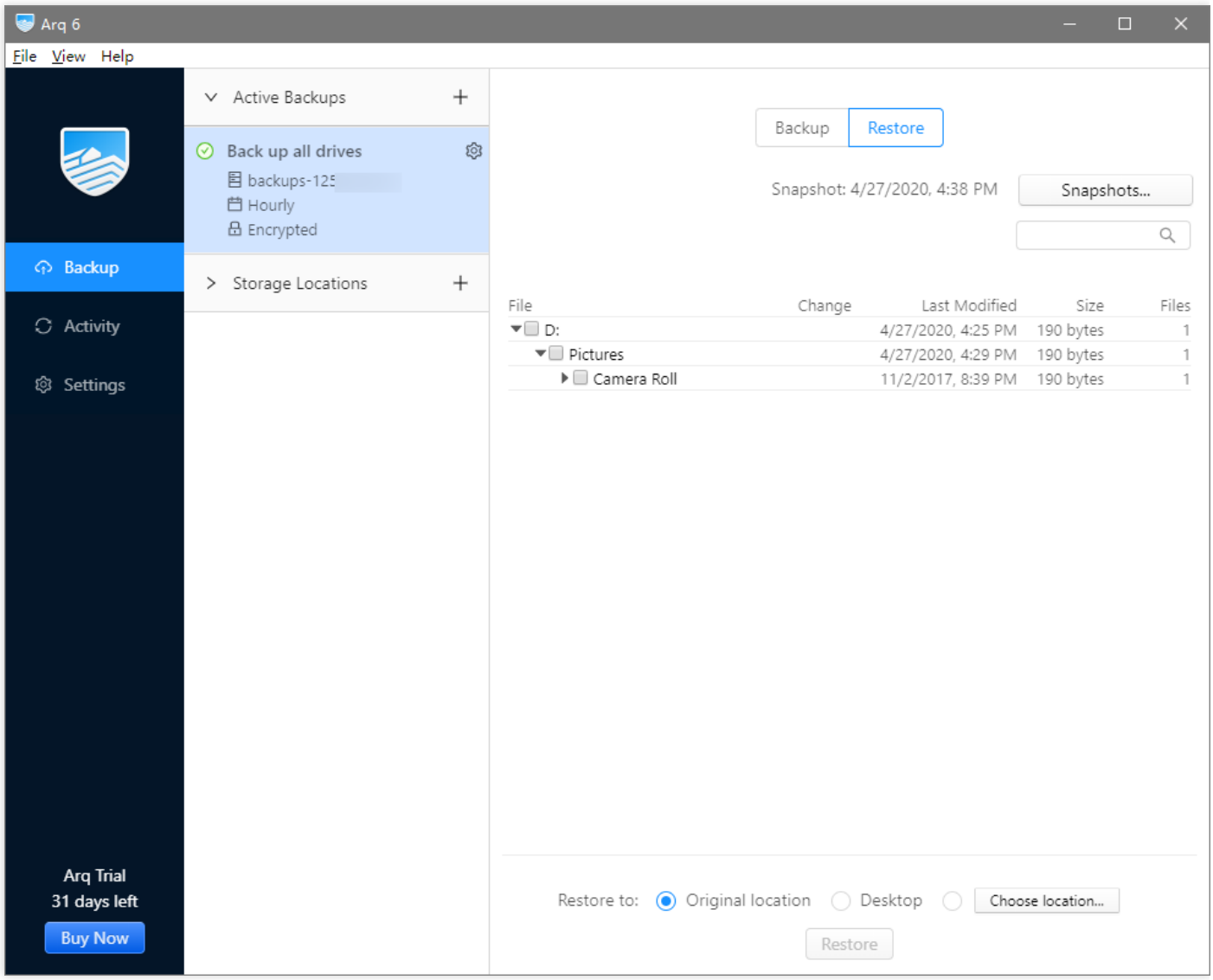
1. Click **Restore** in the list of **Backup** in the left sidebar.



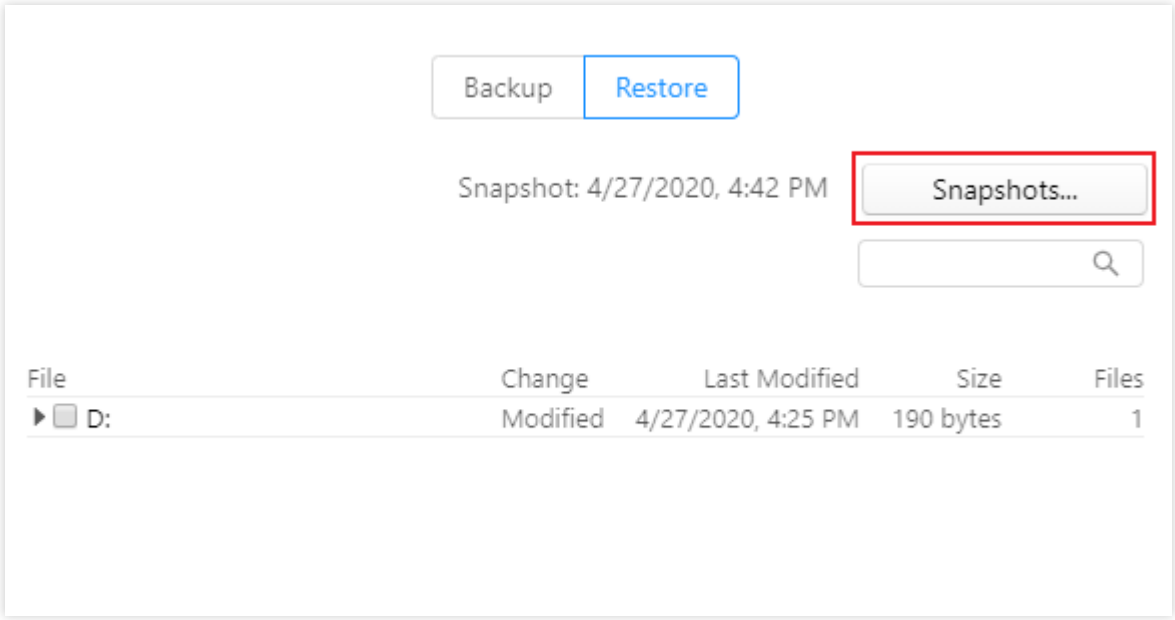
2. Enter your password if you have enabled **Encrypt backup data** in Step 9.



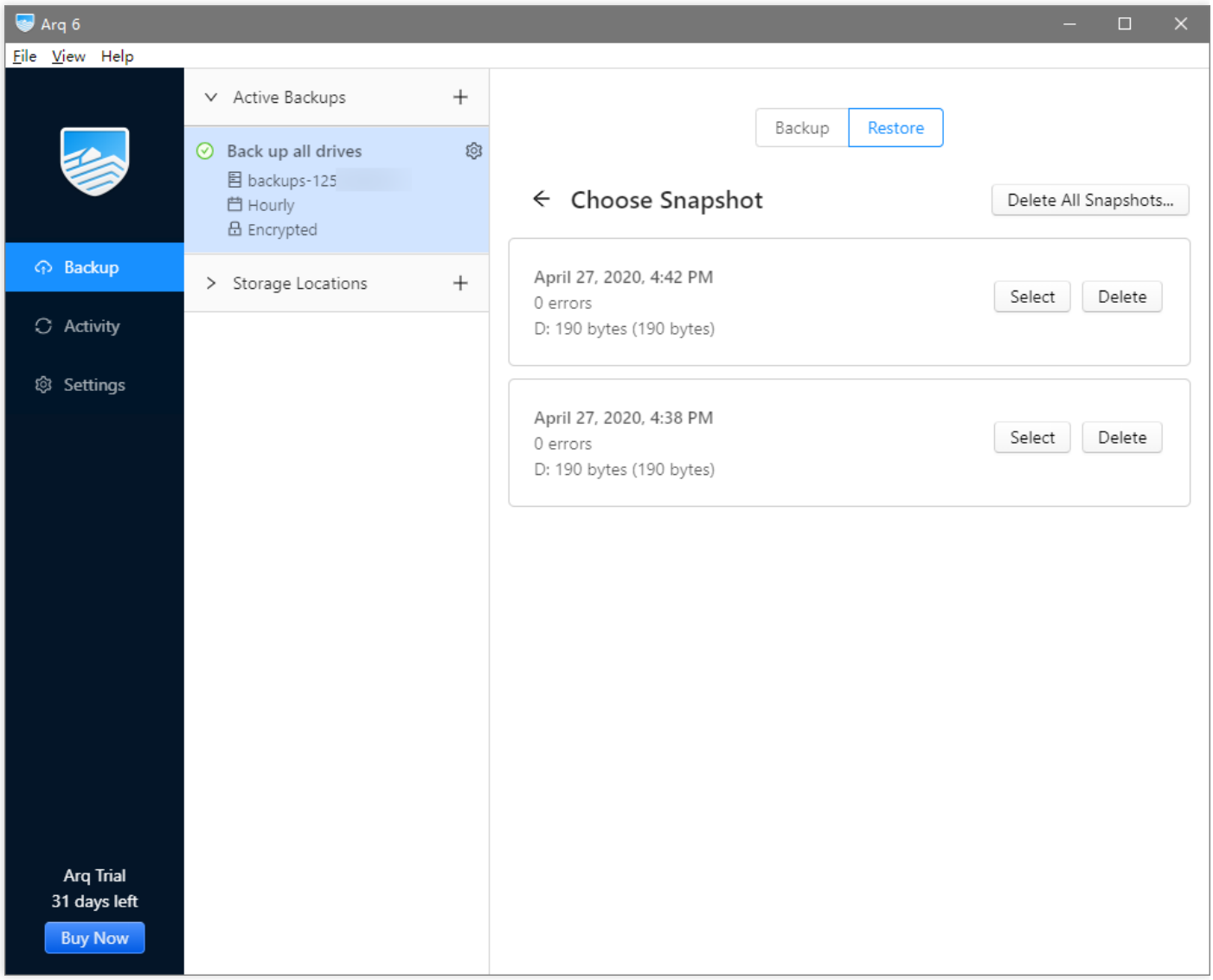
3. Select the directories or files to restore, and the location to save them, and then click **Restore**.



4. By default, files are restored from their latest backup. If necessary, you can choose to restore from an old version of backup in snapshots, which can be viewed by clicking **Snapshots**.



5. Choose a snapshot.



6. Select the snapshot directories or files to restore, and the location to save them, and then click **Restore**.
7. Once prompted that the restore operation is completed, you can go to the specified directories and view your restored files.

Mounting COS to Windows Server as Local Drive

Last updated：2022-08-30 15:12:13

Overview

COS can be used on Windows mainly with APIs, COSBrowser, or COSCMD.

However, if you are a Windows Server user, you can only use COSBrowser for cloud file storage, which is not friendly for running programs or performing operations. In this case, you can mount the cost-effective COS to Windows Server as local drive as instructed in this document.

> Note：
> Examples given in this document apply only to Windows 7 or Windows Server 2012 and later.

Directions

Download and installation

The following installation procedure is for your reference only. You should perform the installation based on your actual OS:

- Go to GitHub to download WinFsp.
  You can install it with the default options.
- Go to Git or GitHub to download Git.
  `Git-2.31.1-64-bit.exe` is downloaded as an example. You can install it with the default options.
- Go to Rclone or GitHub to download Rclone.
  `rclone-v1.55.0-windows-amd64.zip` is downloaded as an example. You can decompress it to any directory. In this example, the package is decompressed to `E:\AutoRclone` .

> Note：
> If GitHub cannot be opened or is slow, you can try another way on your own for the download.

Rclone configuration

Note：
The following configuration process takes Rclone v1.55.0 as an example. Note that the configuration process of other versions may be different.

1. Open any folder, find **This PC** in the left navigation pane, right-click and select **Properties** > **Advanced system settings** > **Environment Variables** > **System variables** > Path*, and click **New*.
2. In the pop-up window, enter the path where Rclone is decompressed (E:\AutoRclone) and click **OK**.
3. Open Windows PowerShell and run the `rclone --version` command to see whether Rclone is installed successfully.
4. If Rclone is installed successfully, run the `rclone config` command in Windows PowerShell.
5. In Windows PowerShell, enter **n** and press **Enter** to create a new remote.
6. In Windows PowerShell, enter the name of the drive (for example, `myCOS`) and press **Enter**.
7. In the options that are displayed, enter **4** (the one that contains Tencent Cloud) and press **Enter**.
8. In the options that are displayed, enter **11** (the one that contains Tencent Cloud COS) and press **Enter**.
9. When `env_auth>` is displayed, press **Enter**.
0. When `access_key_id>` is displayed, enter the `SecretId` of COS and press **Enter**.

Note：
We recommend you use sub-account permissions here. You can go to the Manage API Key page to view your `SecretId` and `SecretKey` .

1. When `secret_access_key>` is displayed, enter the `SecretKey` of COS and press **Enter**.
2. Choose the region of the bucket based on the gateway addresses of the Tencent Cloud regions that are displayed. Guangzhou region is used as an example here. Therefore, you can enter **4** (`cos.ap-guangzhou.myqcloud.com`) and press **Enter**.
3. Select the object permission (`private` or `public-read`) as needed, which takes effect only for objects that are uploaded subsequently. `public-read` is used as an example here. Therefore, you can press **2** and then press **Enter**.
4. Select the storage class for your objects uploaded to COS. `Default` is used as an example here. Therefore, you can press **1** and then press **Enter**.

- Default: Default option
- Standard storage class: STANDARD

- Infrequent access storage mode: Standard_IA
- Archive storage mode: ARCHIVE

> Note：
> To use the INTELLIGENT TIERING or DEEP ARCHIVE storage class, **modify the configuration file** and set the value of `storage_class` to `INTELLIGENT_TIERING` or `DEEP_ARCHIVE`.

5. When `Edit advanced config? (y/n)` is displayed, press **Enter**.
6. After confirming the information, press **Enter**.
7. Enter **q** to complete the configuration.

Modifying configuration file

After performing the configuration above, the Rclone configuration file `vrclone.conf` can be found in the `C:\Users\Username\.config\rclone` directory. You can directly modify the file to update the Rclone configuration.

Mounting COS as local drive

1. Open the installed Git CMD 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 it with the user-defined drive name.
     - Y: Replace it with the drive letter you want to assign to the drive. Make sure that it does not conflict with other drive letters.
     - E:\temp: Local cache directory, which can be configured as needed.

   If "The service rclone has been started" is displayed, the mount is successful.
2. Enter **exit** to close the terminal.
3. Find the `myCOS(Y:)` drive in **This PC**.
   If you open this drive, you can see all buckets in Guangzhou region. You can upload, download, create, and delete files and do more in the drive as needed.

Note：

- If any error is reported during the process, you can view the error messages from Git Bash.
- If you delete a bucket from the drive, the bucket will be deleted from COS, no matter whether there are objects stored in it or not.
- If you change the name of a bucket in the drive, the bucket name in COS will also be changed.

Running 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 automatically run at startup.

1. Create `startup_rclone.vbs` and `startup_rclone.bat` files in the `E:\AutoRclone` directory.
2. In `startup_rclone.bat` , write the following mount command:

- To mount COS as a shared drive on LAN, enter 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, enter 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
```

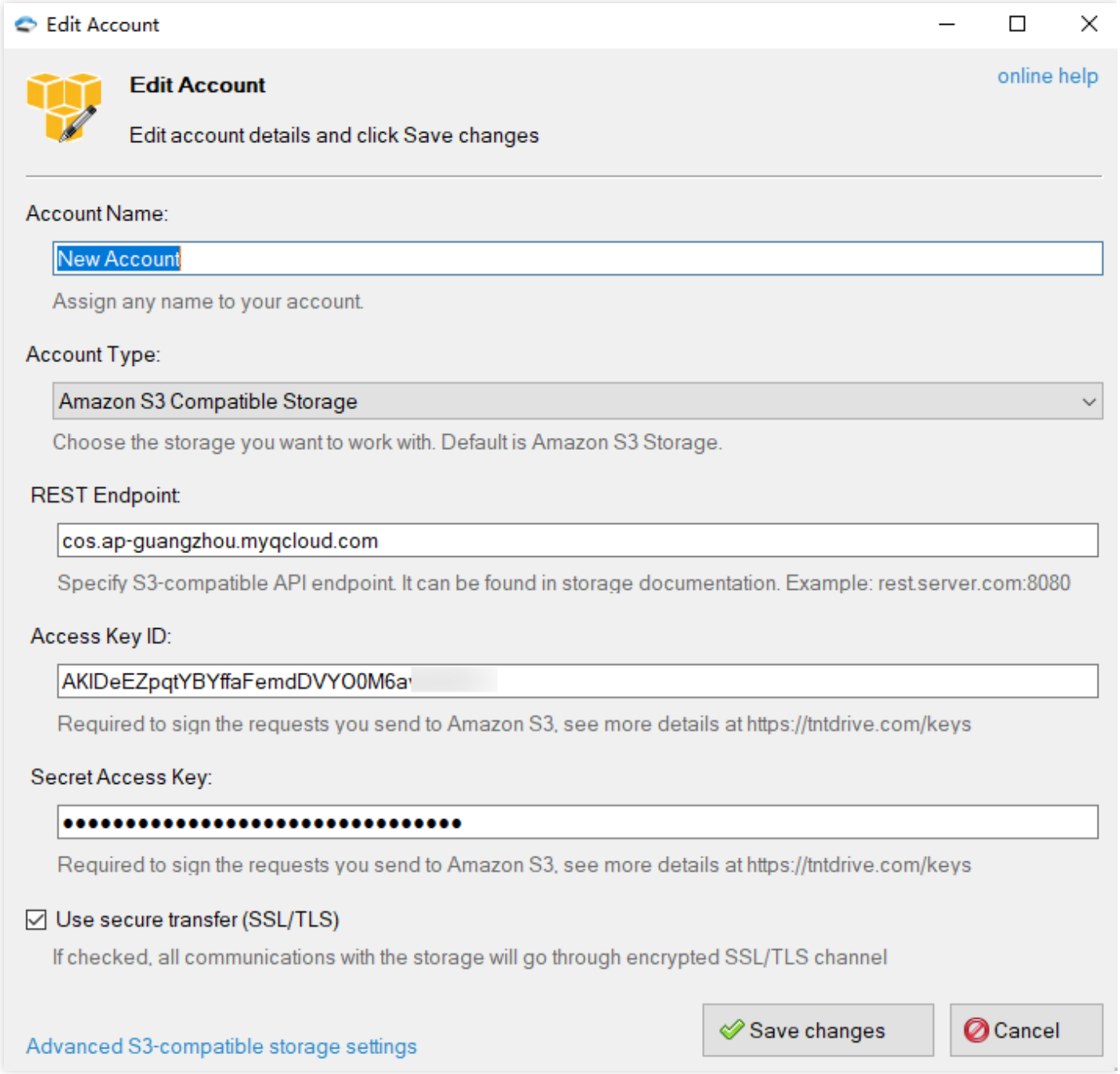
Note：
Replace the path with the actual one.

4. Cut the `startup_rclone.vbs` file to the `%USERPROFILE%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup` directory.
5. Restart the server.

Related Operations

You can also use a third-party tool to mount COS to a Windows server as local drive. The following shows the mounting procedure with the TntDrive tool:

1. Download and install TntDrive.
2. Open TntDrive and click **Account** > **Add New Account** to create an account.

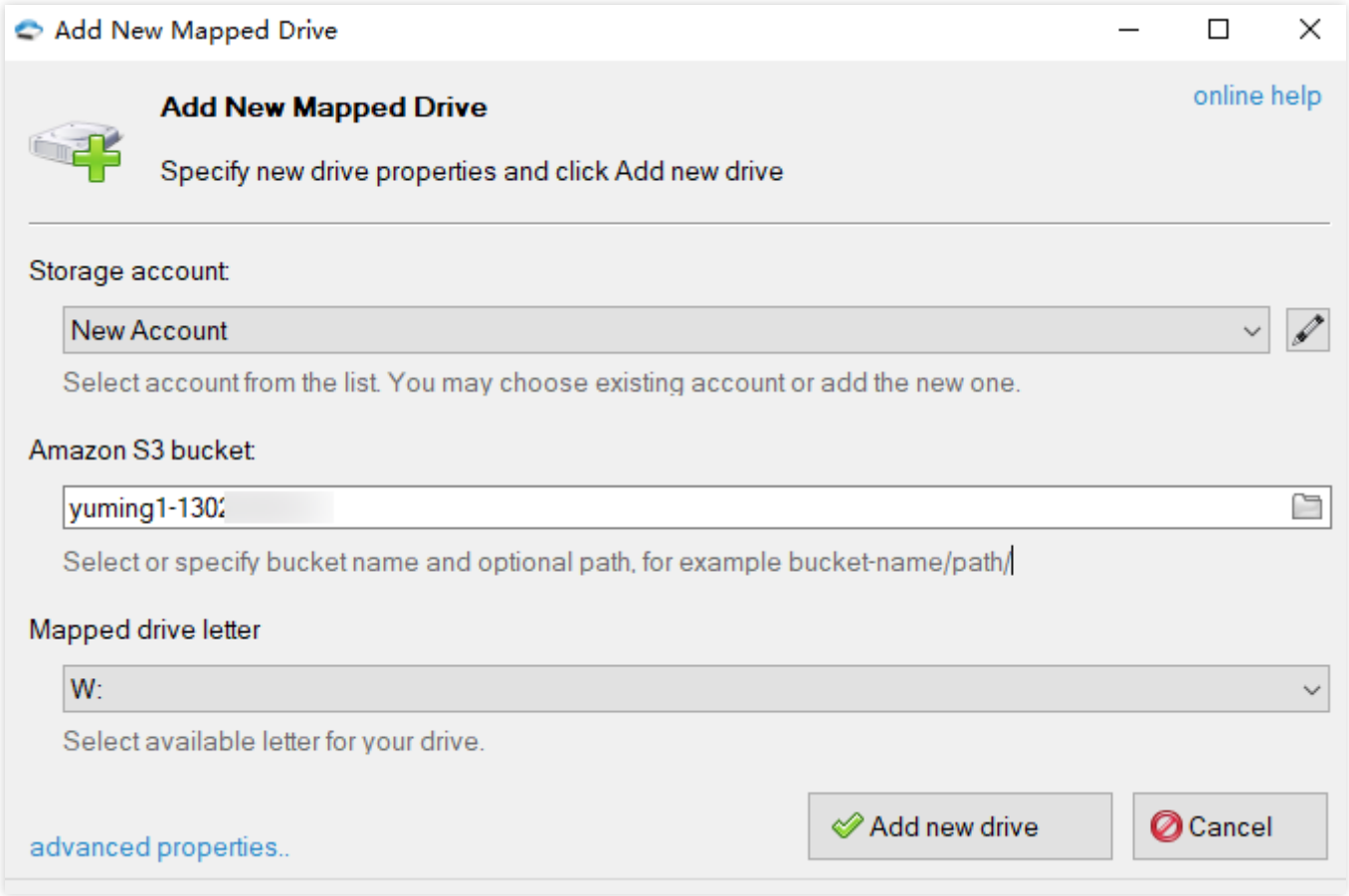


Main parameters are as described below:

- **Account Name**: User-defined account name.

- **Account Type**: Account type. As COS is compatible with S3, you can select **Amazon S3 Compatible Storage**.
- **REST Endpoint**: Enter the region of your bucket. For example, if your bucket resides in Guangzhou region, set this parameter to `cos.ap-guangzhou.myqcloud.com` .
- **Access Key ID**: Enter the value of your `SecretId` , which can be created/obtained on the Manage API Key page.
- **Secret Access Key**: Enter the value of your `SecretKey` .

3. Click **Add new account**.
4. In the TntDrive window, click **Add New Mapped Drive** to create a mapped drive.



Main parameters are as described below:

- **Amazon S3 bucket**: Enter the path or name of the bucket. You can click the icon on the right to select a bucket. In this example, the bucket in Guangzhou region set in step 2 is selected (mapping a bucket as a drive).
- **Mapped drive letter**: Set the drive letter of the mapped drive, which cannot conflict 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.

Backing Up Oracle Databases to COS with RMAN

Last updated：2021-09-30 12:29:07

Overview

COS offers unlimited storage and can automatically transition objects to ARCHIVE or DEEP ARCHIVE for cost reduction, making it ideal for data backup and archive.

More and more customers are backing up data in the cloud. To facilitate this, Oracle's backup module has integrated with COS so that users can back up and restore databases at a lower cost.

Connecting OSB to COS

The Oracle Secure Backup (OSB) Cloud Module is part of the OSB product family. It enables you to directly back up Oracle databases to COS. The OSB Cloud Module has integrated with RMAN. Therefore, you can customize an RMAN script to directly back up Oracle databases to COS efficiently.

Oracle 9i or later versions support the OSB Cloud Module. For more information, please see Oracle Documentation.

Installing OSB

1. Go to Oracle to obtain the latest version of OSB and install it.
2. Decompress the downloaded `osbws_installer.zip` and run the following command with the actual `SecretId`, `SecretKey`, region, and endpoint of COS to install OSB:

```
java -jar osbws_install.jar -AWSID <SecretId> -AWSKey <SecretKey> -walletDir $O
RACLE_HOME/osbws_wallet -libDir $ORACLE_HOME/lib -location <Region> -awsEndPoin
t <endpoint>
// Replace the location with the actual directory of the package.
```

Example:

```
java -jar osbws_install.jar -AWSID AKIDxxxx -AWSKey XXXX -walletDir $ORACLE_HOM
E/osbws_wallet -libDir $ORACLE_HOME/lib -location ap-guangzhou -awsEndPoint cos
.ap-guangzhou.myqcloud.com
```

Note：
Oracle 12 and later versions have the OSB cloud module by default. If the installation does not work, you can download the latest OSB module and install it by yourself or use a proxy.

Backing Up Databases to COS Using RMAN

1. Log in to the database and run the following command to connect to RMAN:

```
rman target /
```

2. Back up the databases to a COS bucket using the commands below, where `lib/libcos.so` and `cosorcl.ora` are about the database names and should be modified as needed.

```
run {
allocate channel ch1 type
sbt parms='SBT_LIBRARY=/u01/app/oracle/product/11.2.0/dbhome_1/lib/libcos.so,
SBT_PARMS=(OSB_WS_PFILE=/u01/app/oracle/product/11.2.0/dbhome_1/dbs/cosorcl.or
a)';
backup channel=ch1 database format='ora_%d_%I_%T_%s_%t_%c_%p.dbf' plus archivel
og;
backup channel=ch1 current controlfile format='%d_%I_%T_%s_%t_%c_%p.conf';
backup channel=ch1 spfile format='ora_%d_%I_%T_%s_%t_%c_%p.spf' ;
release channel ch1;
}
```

Restoring a Database from COS Using RMAN

1. Shut down the database and make its status “unmount”.
   - Shut down the database:

     ```
     shutdown immediate;
     ```

   - Make the database status “unmount”:

     ```
     startup nomount;
     ```

2. Run the RMAN command `list backup` to list all backups and select the desired one, whose handle and tag should be recorded.

3. Run the following `restore` command to connect to the backup.
   Connect to the backup whose handle is `ORACLE_1880733115_20190507_5_1007656283_1_1.conf`, which is obtained from `list backup`. `lib/libcos.so` and `cosorcl.ora` are about the database names. They should be modified as needed and be the same as those used in the backup process.

```
run {
allocate channel ch1 type
sbt parms='SBT_LIBRARY=/u01/app/oracle/product/11.2.0/dbhome_1/lib/libcos.so,
SBT_PARMS=(OSB_WS_PFILE=/u01/app/oracle/product/11.2.0/dbhome_1/dbs/cosorcl.or
a)';
restore controlfile from 'ORACLE_1880733115_20190507_5_1007656283_1_1.conf';
release channel ch1;
}
```

4. Run the `restore` command below and change the database status to `mount` with `alter database mount`:

```
run {
allocate channel ch1 type
sbt parms='SBT_LIBRARY=/u01/app/oracle/product/11.2.0/dbhome_1/lib/libcos.so,
SBT_PARMS=(OSB_WS_PFILE=/u01/app/oracle/product/11.2.0/dbhome_1/dbs/cosorcl.or
a)';
restore database from tag='TAG20190507T163102';
recover database from tag='TAG20190507T163102';
release channel ch1;
}
```

   - The tag value **TAG20190507T163102** is a backup's ID number, which is obtained using `list backup`. Ensure that the value is the same as that obtained from the control file of the database to restore, that is, the handle of the backup should be **ORACLE_1880733115_20190507_5_1007656283_1_1.conf**.
   - `lib/libcos.so` and `cosorcl.ora` are about the database names and should be modified as needed.
5. Open the database to view the data restored from COS.

Modifying RMAN Concurrence

> Note：
> By default, RMAN does not have concurrence. To use concurrence, you need to set it manually.

Log in to RMAN and modify the concurrence configuration. The following sets the number of concurrences to 15:

```
run {
configure channel device type sbt parms='SBT_LIBRARY=/u01/app/oracle/product/11.
2.0/dbhome_1/lib/libcos.so ENV=(OSB_WS_PFILE=/u01/app/oracle/product/11.2.0/dbhom
e_1/dbs/cosorcl.ora)';
configure default device type to SBT;
configure device type SBT parallelism 15;
}
```

References

For more information, please see Oracle Documentation.

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

Last updated：2022-11-08 14:36:47

Overview

The image hosting service provides various features such as image storage, processing, and distribution to sustain countless blog sites and community forums worldwide on the backend. COS is a distributed storage service launched by Tencent Cloud to store massive numbers of files, with higher performance and reliability guaranteed. You can use **COS** to set up an image hosting service.

The strengths of COS in the image hosting scenarios include:

- **Low costs**: The unit price of storage is low, and you only need to pay for what you use.
- **Unrestricted speed**: Upload and download speeds are not restricted, so users no longer need to wait for slow loading, enjoying a better access experience.
- **High availability**: COS offers an SLA for high availability, where stored data has a guaranteed durability of up to 99.9999999999%.
- **Unlimited capacity**: COS stores high numbers of files in a distributed manner for on-demand capacity use.

Practice Scenarios

Scenario 1: Adding images to set up an image hosting service with COS

The following tools are used in this scenario:

- PicGo: A tool that supports multiple cloud storage configurations and quickly generates image URLs.
- Typora: A lightweight Markdown file editor that supports multiple output formats and allows you to quickly upload local images to an image hosting service.

Directions

1. Install PicGo and set relevant COS parameters.

> Note：
> PicGo 2.3.0 is used in this scenario. Note that the configuration process may vary by version.

After downloading PicGo from PicGo website and installing it, find **Tencent Cloud COS** in the image hosting service settings and configure the following parameters:

- COS Version: Select COS v5.
- Set SecretId: It is a developer-owned secret ID used for the project, which can be created and obtained on the Manage API Key page.
- Set SecretKey: It is a developer-owned secret key used for the project, which can be obtained on the Manage API Key page.
- Set APPID: It is a unique user-level resource identifier for COS access, which can be obtained on the Manage API Key page.
- Set Storage Space Name: It is a bucket, i.e., a container used for data storage. For more information, see Bucket Overview.
- Confirm Storage Region: It is the region information. For enumerated values such as `ap-beijing`, `ap-hongkong`, and `eu-frankfurt`, see Regions and Access Endpoints.
- Specify Storage Path: It is the path where the image is stored in the COS bucket.
- Set Custom Domain Name: 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.

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:

3. In **Image** of Typora's preferences, configure the following:
   - Select **Upload image** for **When Insert...**.
   - In **Image Upload Settings**, select **PicGo.app** and set the location of PicGo.exe you just installed.

4. Restart Typora for the settings to take effect.

5. Enter the Typora editor area, drag and drop or paste an image directly to upload it and automatically replace it with a COS file URL. (If it is not automatically replaced with a COS URL after pasting, check whether the server in PicGo is enabled.)

Scenario 2: Quickly migrating images in an image hosting repository to COS

Taking an image hosting service as an example, you can find the local image hosting folder, or download the entire folder from the internet, and then transfer all the images in the folder to a COS bucket. Then, uniformly replace the URL domain name to restore the website.

Directions

Step 1. Download images in the original image hosting service

Log in to the original image hosting website and download the previously uploaded image folder.

Step 2. Create a COS bucket and set up hotlink protection

1. Sign up for a Tencent Cloud account and create a bucket with access permissions of **public read/private write** as instructed in Creating a Bucket.
2. After the bucket is created, enable hotlink protection in the bucket as instructed in Setting Hotlink Protection to avoid images from being hotlinked.

Step 3. Upload the folder to the bucket

In the COS bucket you just created, click **Upload Folder** to upload the prepared image folder to the bucket.

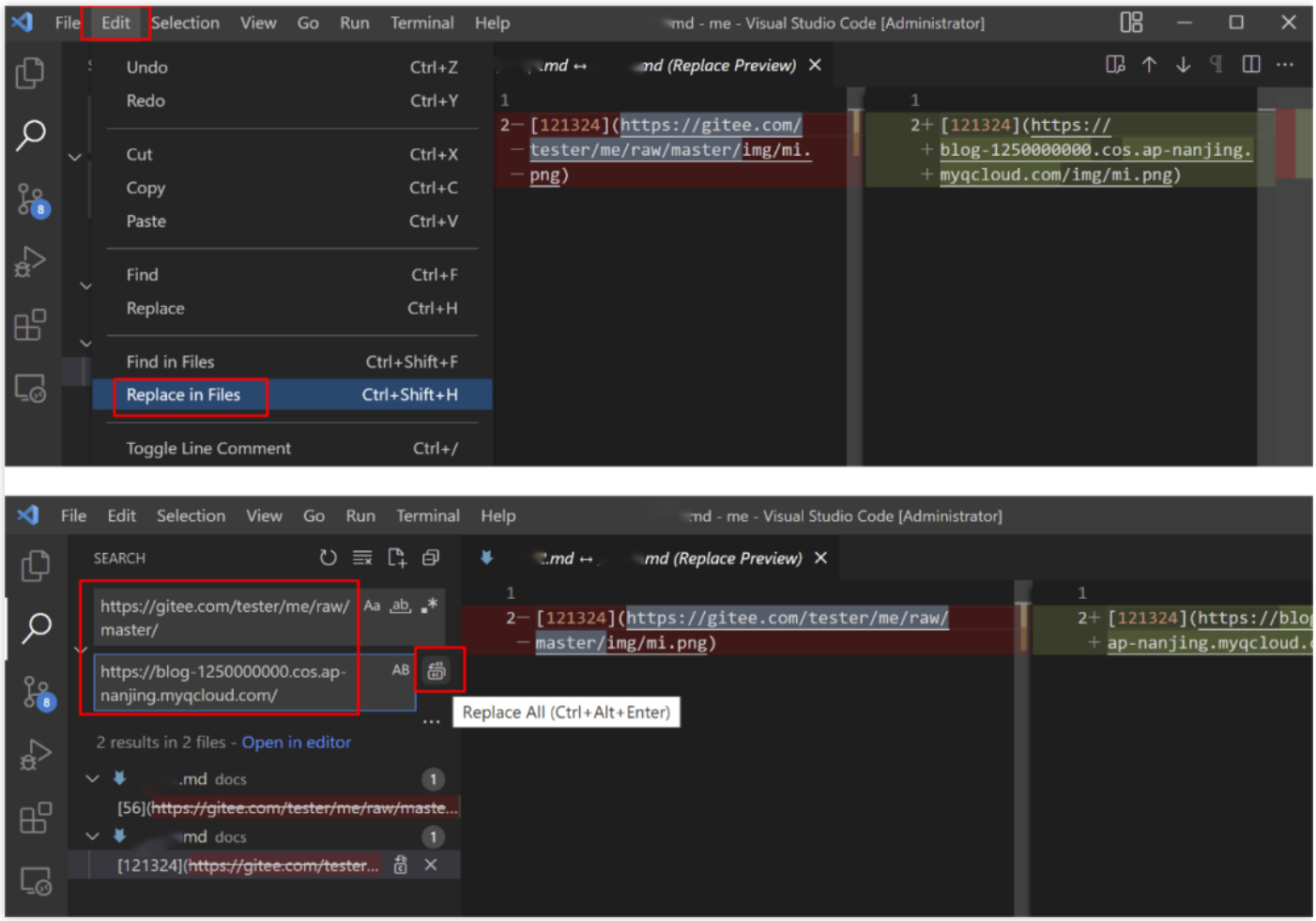
> Note：
> If the number of images is high, you can also use COSBrowser to upload images quickly.

Step 4. Globally replace the domain name

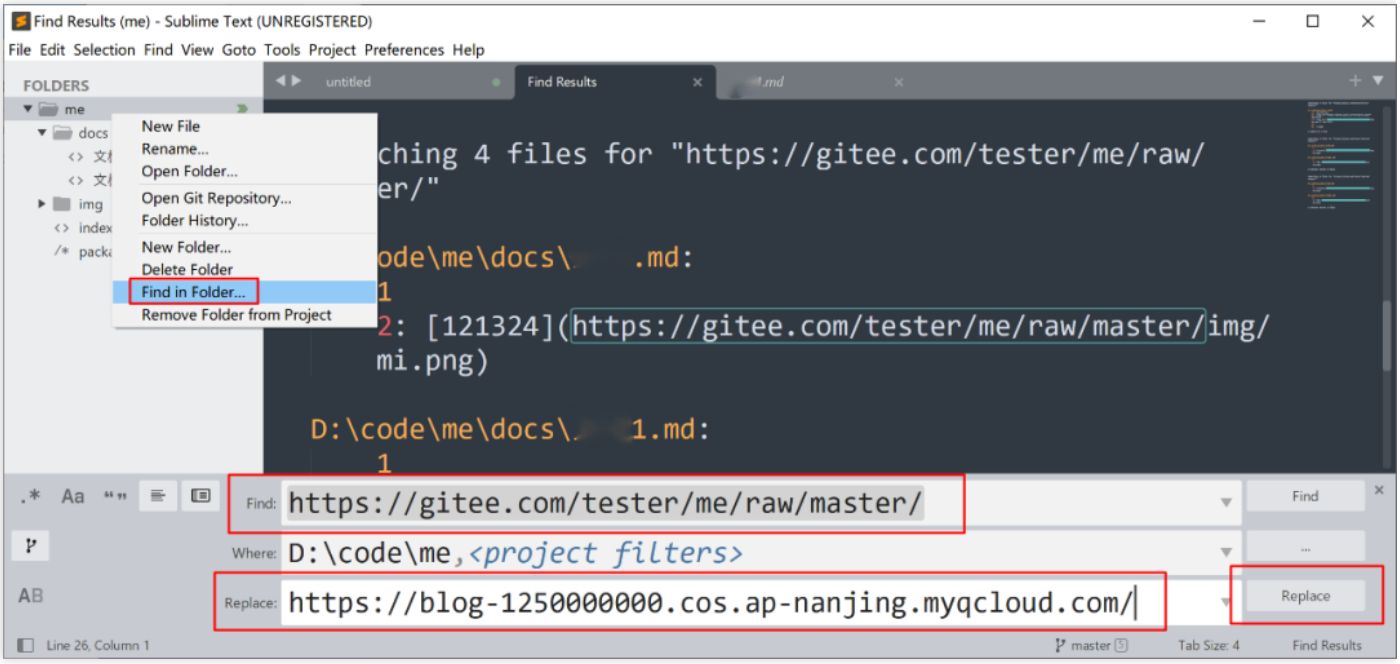
On the bucket overview page in the COS console, copy the default domain name of the bucket (you can also associate a custom CDN acceleration domain name). Then, use a common code editor to search for and replace the invalid URL prefix globally with the default domain name of the COS bucket.

> Note：
> For more information on the default domain name, see Regions and Access Endpoints.

- Example search-and-replace with Visual Studio Code:



- Example search-and-replace with Sublime Text:



Managing COS Resource with CloudBerry Explorer

Last updated：2022-10-25 14:47:40

Overview

CloudBerry Explorer is a client tool for COS management. It can mount COS on Windows and other operating systems for you to easily access, move, and manage files in COS.

Supported Systems

Windows and macOS.

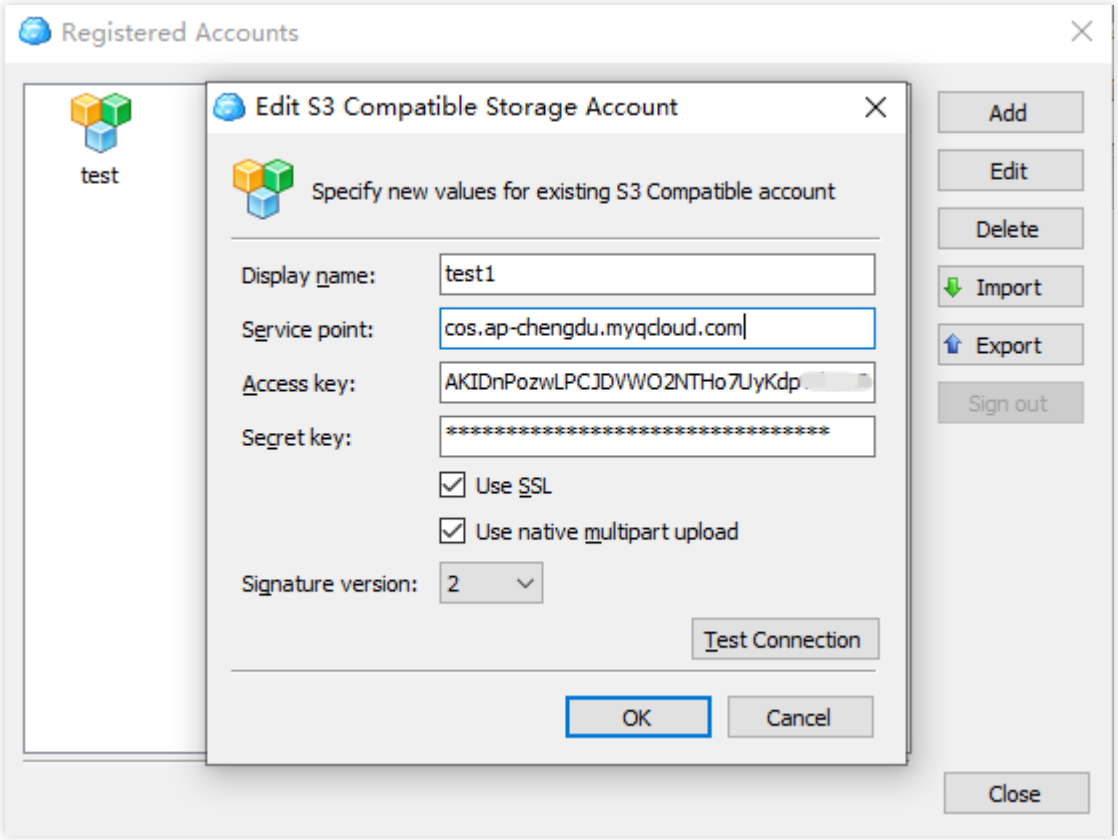
Download Address

Download CloudBerry Explorer here.

Installation and Configuration

> Note：
> The following configuration process takes CloudBerry Explorer Windows v6.3 as an example. Note that the configuration process may vary by version.

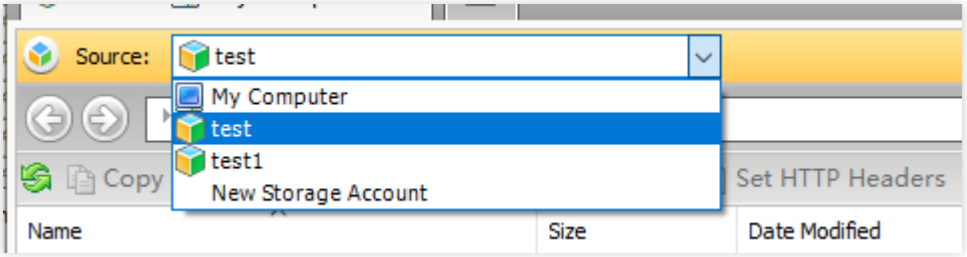
1. Double-click the installation package and complete the installation as prompted.
2. Open the tool and double-click **S3 Compatible**.
3. Configure the following information in the pop-up window, click **Test Connection**, and wait until the connection is successful.



The configuration items are as described below:

- Display name: Enter a custom username.
- Service point: The format is `cos.<region>.myqcloud.com`; for example, to access a bucket in Chengdu region, enter `cos.ap-chengdu.myqcloud.com`. For applicable region abbreviations, see Regions and Access Endpoints.
- Access key: Enter the `SecretId`, which can be created and obtained on the Manage API Key page.
- Secret key: Enter the `Secretkey`, which can be created and obtained on the Manage API Key page.

4. After adding the account information, select the configured username in **Source** to view the list of buckets under the username. At this point, the configuration is completed.



Managing COS File

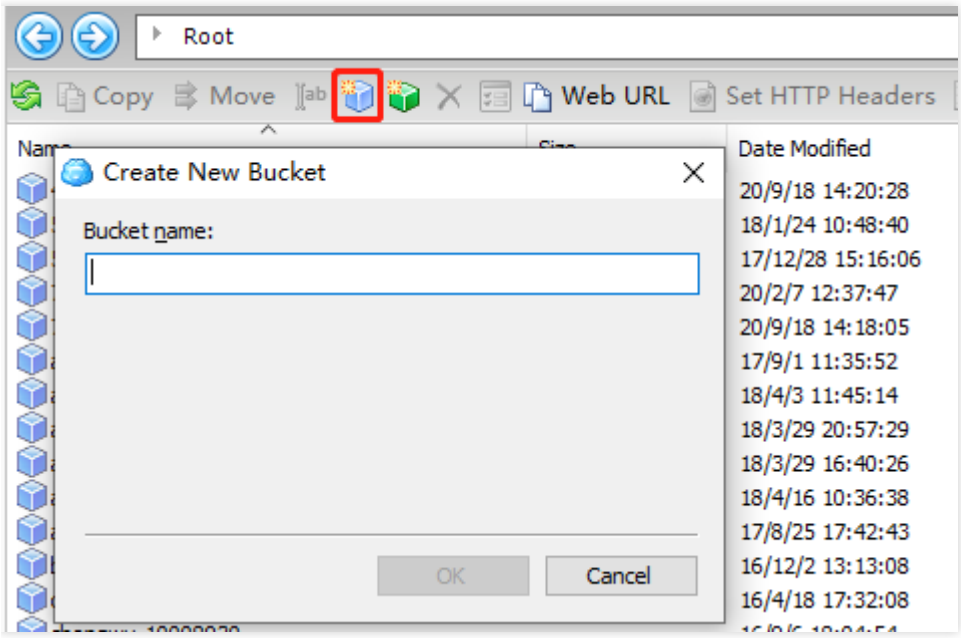
Querying bucket list

Select the configured username in **Source** to view the list of buckets under the username.

> Note：
> With this operation, you can only view the buckets in the region configured by the **Service point**. To view buckets in other regions, click **File** > **Edit Accounts**, select a username, and change **Service point** to another region.

Creating a bucket

Click the icon as shown below, enter the full bucket name in the pop-up window such as `examplebucket-1250000000` , and click **OK**.
For bucket naming conventions, see Bucket Overview.

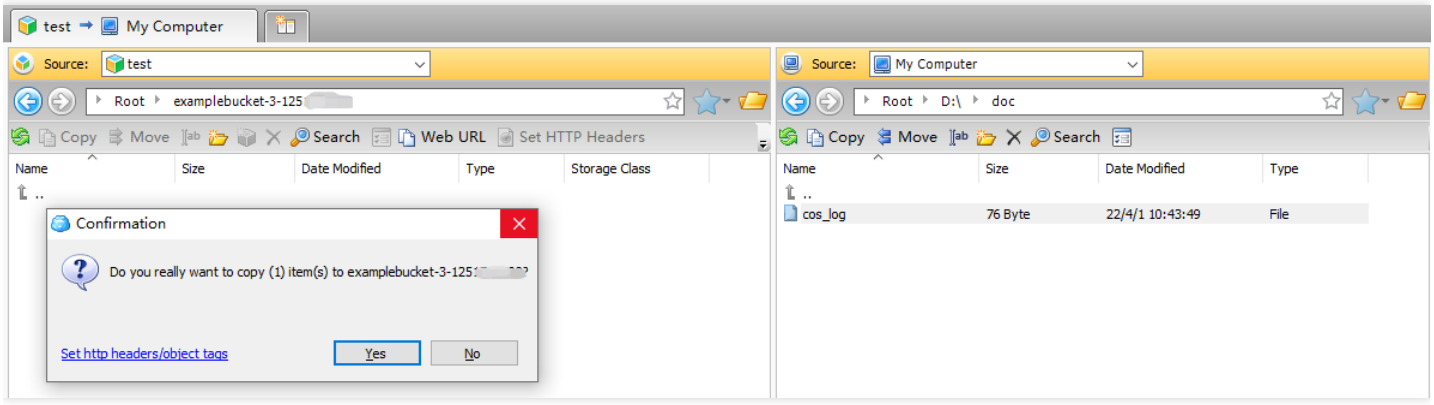


Deleting a bucket

Right-click the target bucket in the bucket list and select **Delete** in the context menu.

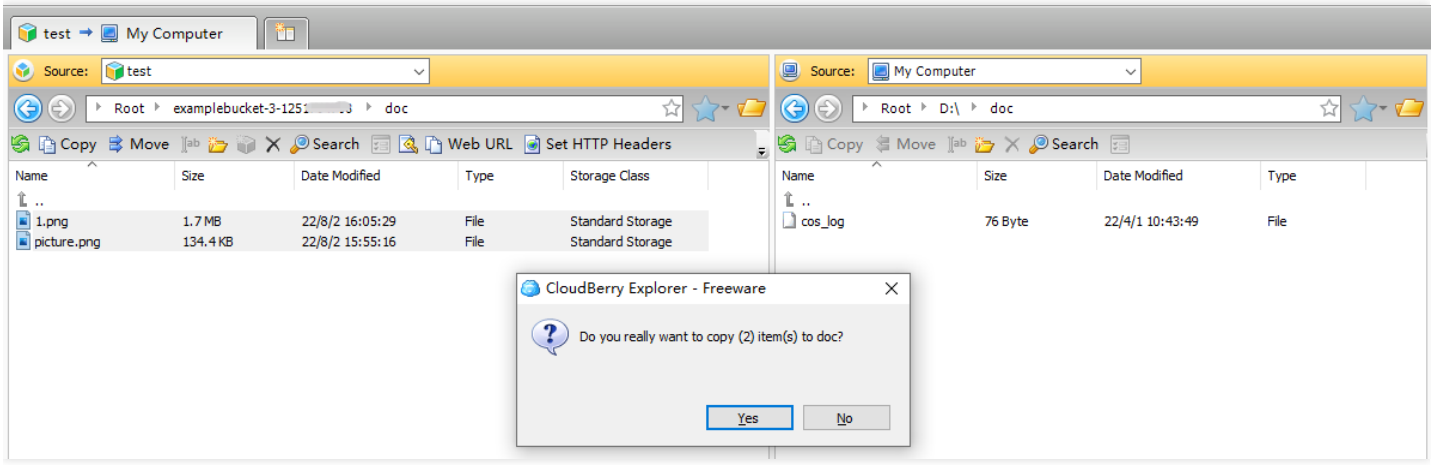
Uploading an object

In the bucket list, select the destination bucket or path, select the object to be uploaded on the local computer, and drag and drop it to window on the left.



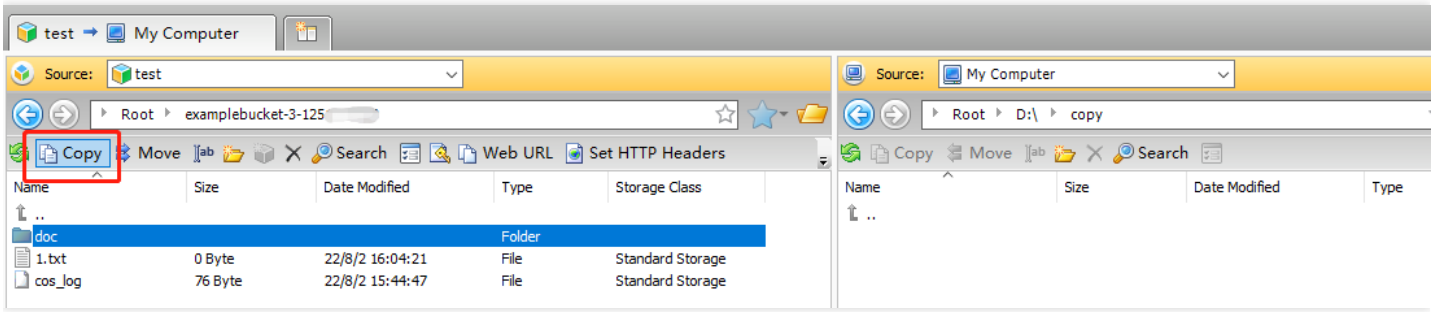
Downloading an object

Select the target object in the window on the left and drag and drop it to a folder on the local computer on the right.



Copying an object

Select the destination path in the right window of the tool, right-click the target object in the left window, select **Copy**, and confirm in the pop-up window.



Renaming an object

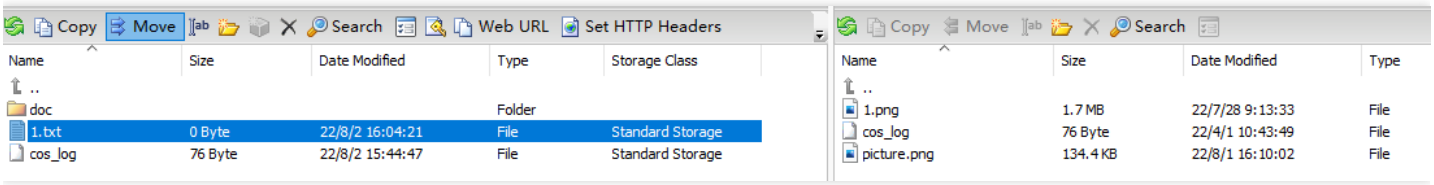
Right-click the target object in the bucket, select **Rename**, and enter a new name.

Deleting an object

Right-click the target object in the bucket and select **Delete**.

Moving an object

Select the destination path in the right window of the tool, right-click the target object in the left window, select **Move**, and confirm in the pop-up window.



Other features

In addition to the above features, CloudBerry Explorer also allows you to set object ACLs, view object metadata, customize headers, and get object URLs.

Managing COS Resource with DragonDisk

Last updated：2022-10-25 14:47:41

Overview

DragonDisk is a free file manager with a GUI similar to Windows File Explorer and supports data backup and sharing. You can use it to easily and quickly manage files in COS.

Supported Systems

Windows, macOS, and various Linux distributions.

Download Address

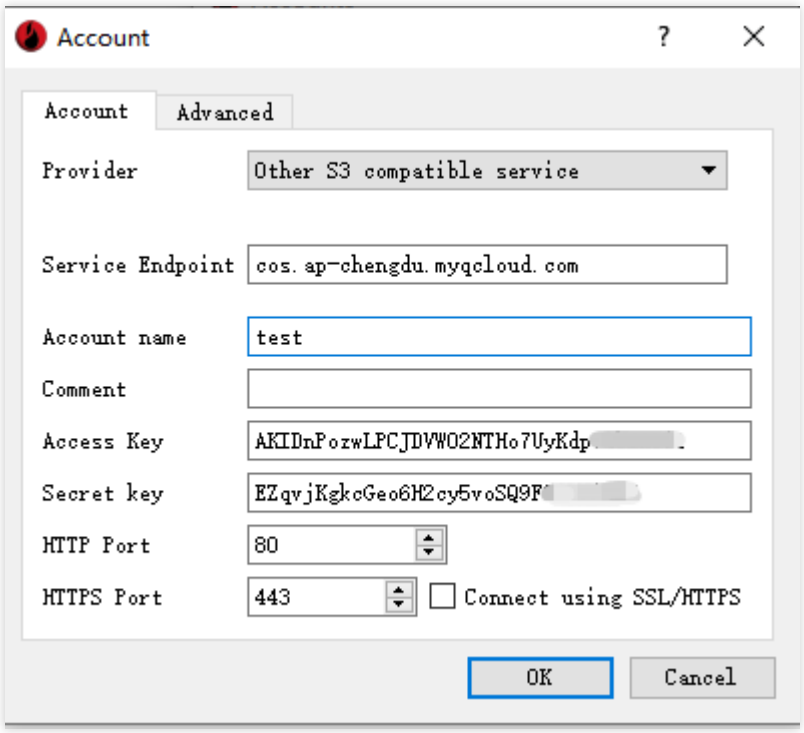
Go to the DragonDisk Download page and download it.

Installation and Configuration

> Note：
> The following configuration process takes DragonDisk Windows v1.05 as an example. Note that the configuration process may vary by version.

1. Double-click the installation package and complete the installation as prompted.
2. Open the tool, select **File** > **Accounts**, and click **New** in the pop-up window to add the account configuration information.

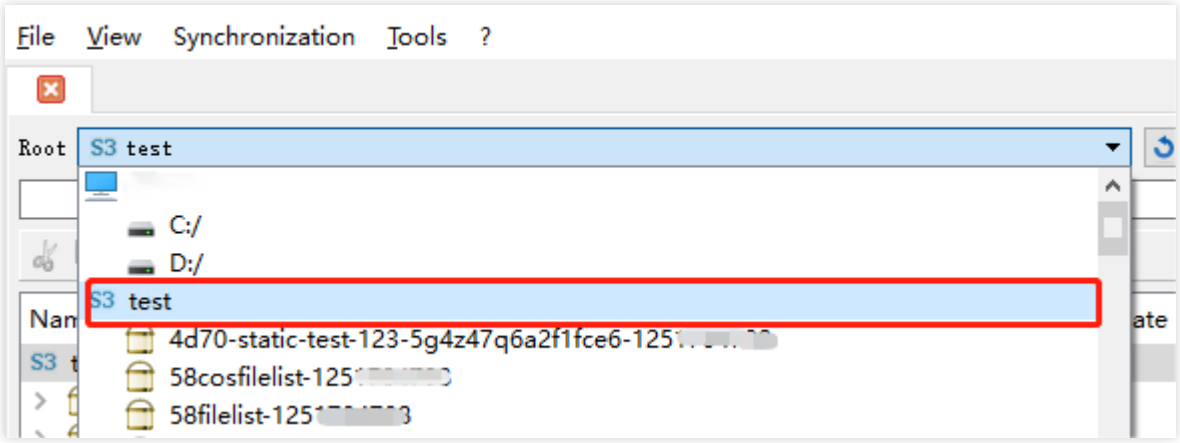
3. Configure the following information in the pop-up window.



The configuration items are as described below:

- Provider: Select **Other S3 compatible service**.
- Service Endpoint: The format is `cos.<region>.myqcloud.com` ; for example, to access a bucket in Chengdu region, enter `cos.ap-chengdu.myqcloud.com` . For applicable region abbreviations, see Regions and Access Endpoints.
- Account name: Enter a custom username.
- Access key: Enter the `SecretId` , which can be created and obtained on the Manage API Key page.
- Secret key: Enter the `Secretkey` , which can be created and obtained on the Manage API Key page.

4. After adding the account information, select the configured username in **Root** to view the list of buckets under the username. At this point, the configuration is completed.



Managing COS File

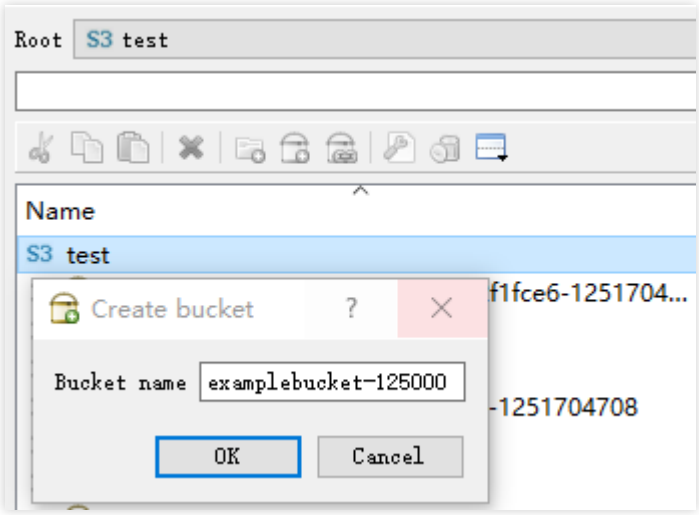
Querying bucket list

Select the configured username in **Root** to view the list of buckets under the username.

> Note：
> With this operation, you can only view the buckets in the region configured by the **Service Endpoint**. To view buckets in other regions, click **File** > **Accounts**, select a username, and change **Service Endpoint** to another region.

Creating a bucket

1. Right-click the username and enter the full bucket name in the pop-up window such as `examplebucket-1250000000`.

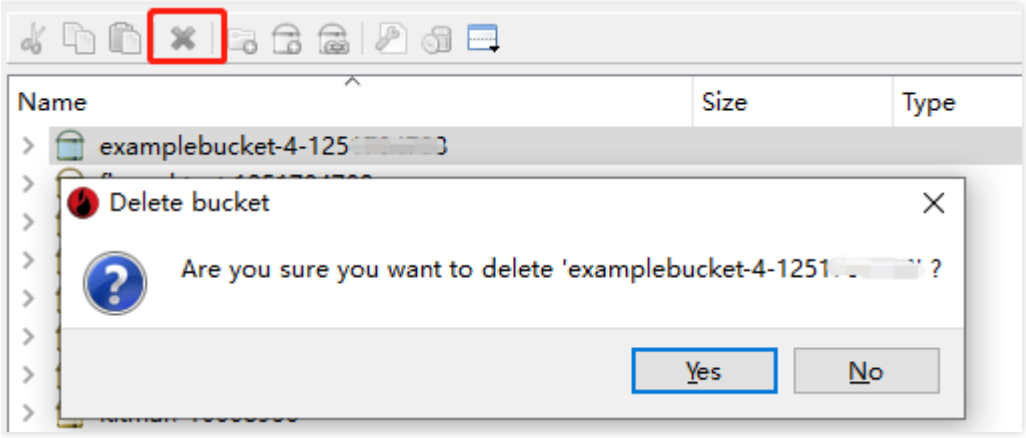


2. After confirming that everything is correct, click **OK**.

   For bucket naming conventions, see Bucket Overview.

Deleting a bucket

Right-click the target bucket in the bucket list and select **Delete** in the context menu.

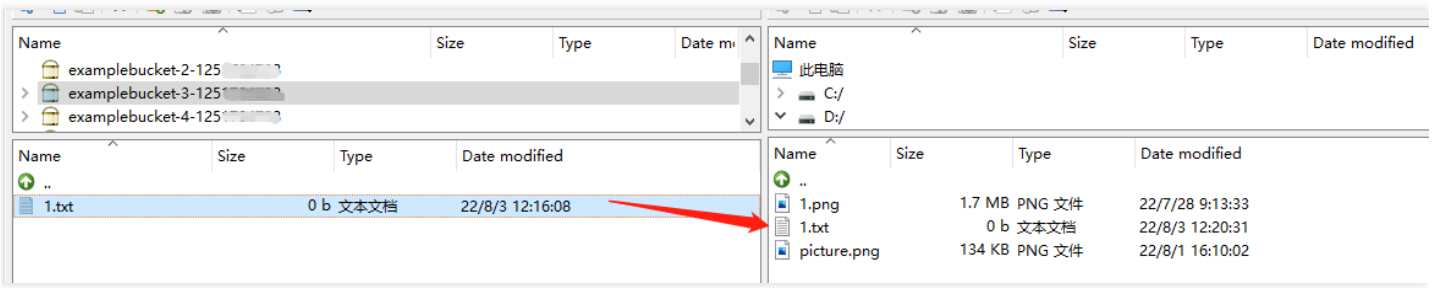


Uploading an object

In the bucket list, select the destination bucket or path, select the object to be uploaded on the local computer, and drag and drop it to the bucket or path.

Downloading an object

Find the target bucket in the bucket list and drag and drop the object to a folder on the local computer on the right.



Copying an object

Right-click the target object in the left window, select **Copy**, right-click under the destination path, and select **Paste**.

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

Right-click the target object in the left window, select **Cut**, right-click under the destination path, and select **Paste**.

Other features

In addition to the above features, DragonDisk also allows you to set object ACLs, view object metadata, customize headers, and get object URLs.

Using APIs to Zip Files

Last updated：2022-12-28 14:31:34

Preparations

1. Multi-File Zipping is implemented with Tencent Cloud Serverless Cloud Function (SCF). You need to log in to the COS console and create a **multi-file zipping** function.For the creation guide, see Multi-File Zipping.
2. After creating the function, click **Instructions** on the right of the function to configure it. The configurations are a **JSON string**, which will be described in detail in this document.

- If your function needs SCF authentication, you need to call the `Invoke` API provided by SCF to run your cloud function, where the `ClientContext` parameter is passed in JSON format (see Parameter Configuration Sample for details).
- For authentication-free functions, you can directly make HTTP requests to the corresponding API gateway to call the function.

Parameter Configuration Sample

Note：
In actual use, remove the comments from the code.

```
{
"bucket": "examplebucket-1250000000", // Bucket to deliver the final ZIP package
"region": "ap-guangzhou", // Region where the bucket resides
"key": "mypack.zip", // Name of the final ZIP package
"flatten": false, // Whether to flatten source file paths
/**
* “sourceList” (a JSON array) is used to specify the list of source files that ne
ed to be zipped.
* Each item includes the source file URL, “renamePath”, and more.
*
* If the source file list is too long, you can JSON stringify the “sourceList” pa
rameter.
* Write the .json file, upload it to COS, and specify it with the “sourceConfigLi
st” parameter.
*
```

```
* You only need to specify either “sourceList” or “sourceConfigList”.
*/
"sourceList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir1/file1.jpg",
"renamePath": "dir1_rename/file1.jpg"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir2/file2.mp4",
"renamePath": "file2.mp4"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file3.md"
}
],
"sourceConfigList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceList.json"
}
]
}
```

The parameters are described as follows:

Parameter	Description	Type	Required
bucket	Bucket to store the final ZIP package, formatted as `BucketName-APPID` (e.g., `examplebucket-1250000000`)	String	Yes
region	Region where the bucket that stores the final ZIP package resides. For more information, please see Region and Access Endpoints.	String	Yes
key	Name (i.e., object key that uniquely identifies an object in the bucket) of the final ZIP package. For more information, please see Object Overview.	String	Yes

Parameter	Description	Type	Required
flatten	Whether to flatten source file paths (i.e., flatten the original directory structure). For example, if a source file's URL is `https://domain/source/test.mp4`, its path will be `source/test.mp4`. If you set this parameter to `true`, its path in the ZIP package will be `test.mp4`. If you set this parameter to `false` (default), its path will be `source/test.mp4`.	Boolean	No
sourceList	A list of source files. **Either `sourceList` or `sourceConfigList` must be specified.**	Array	Yes
sourceList[].url	URL of a source file	String	Yes
sourceList[].renamePath	Renames the path of a source file path in the final ZIP package. For example, you can rename `dir1/file1.jpg` to `dir1_rename/file1.jpg`. Note: `renamePath` has a higher priority over `flatten`, which means the flattening operation will not take effect to the renamed path.	String	No
sourceConfigList	A list of `sourceList` configuration files. If you don't want to include the entire `sourceList` in a request, you can JSON stringify the `sourceList` parameter to generate a JSON configuration file, upload it to COS, and specify the URL of that configuration file in `sourceConfigList` (multiple configuration files can be specified). **Either `sourceList` or `sourceConfigList` must be specified**.	Array	No
sourceConfigList[].url	URL of a `sourceList` configuration file	String	No

Function Response Sample

```
{
code: 0,
data: {
Bucket: "examplebucket-1250000000",
ETag: "\"35bb5e5f050e22bed8f443d8da5dbfb8-1\"",
Key: "mypack.zip",
Location: "examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/mypack.zip"
```

```
},
error: null,
message: "cos zip file success"
}
```

The response parameters are as follows:

Parameter	Description	Type
code	Service error code. `0` indicates successful execution. Other numbers indicate failure.	Number
message	Message for the execution results, which may be `null`	String
data	Message for successful execution. If the execution is successful, this parameter includes the URL of the ZIP package.	Object
error	Error message. If the execution is successful, the value is `null` .	Object/String

Samples

Sample 1: simple use case

Parameter configuration

```
{
"bucket": "examplebucket-1250000000",
"region": "ap-guangzhou",
"key": "mypack.zip",
"flatten": false,
"sourceList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir1/file
1.jpg"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir2/file
2.mp4"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file3.md"
}
]
}
```

ZIP package structure

```
mypack.zip
├── dir1/file1.jpg
├── dir2/file2.mp4
└── file3.md
```

Sample 2: flattening the source file paths

Parameter configuration

```
{
"bucket": "examplebucket-1250000000",
"region": "ap-guangzhou",
"key": "mypack.zip",
"flatten": true, // Set “flatten” to “true” to flatten the source file paths.
"sourceList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir1/file
1.jpg"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir2/file
2.mp4"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file3.md"
}
]
}
```

ZIP package structure

```
mypack.zip
├── file1.jpg
├── file2.mp4
└── file3.md
```

Sample3: renaming source file paths

Parameter configuration

```
{
"bucket": "examplebucket-1250000000",
```

```
"region": "ap-guangzhou",
"key": "mypack.zip",
"flatten": false,
"sourceList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir1/file
1.jpg",
// Rename “dir1/file1.jpg” to “dir1_rename/file1.jpg”.
"renamePath": "dir1_rename/file1.jpg"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir2/file
2.mp4",
// Rename “dir2/file2.mp4” to “file2.mp4”.
"renamePath": "file2.mp4"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file3.md"
}
]
}
```

ZIP package structure

```
mypack.zip
├── dir1_rename/file1.jpg
├── file2.mp4
└── file3.md
```

Sample 4: renaming and flattening source file paths

Parameter configuration

```
{
"bucket": "examplebucket-1250000000",
"region": "ap-guangzhou",
"key": "mypack.zip",
"flatten": true, // Set “flatten” to “true” to flatten source file paths.
"sourceList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir1/file
1.jpg"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/dir2/file
```

```
2.mp4"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file3.md",
// Rename “file3.md” to “dir3/file3.md”. As “renamePath” has a higher priority ov
er “flatten”, the renamed path will not be flattened.
"renamePath": "dir3/file3.md"
}
]
}
```

ZIP package structure

```
mypack.zip
├── file1.jpg
├── file2.mp4
└── dir3/file3.md
```

Using APIs to Merge Files

Last updated：2021-11-18 14:16:27

Preparations

1. The object concatenation feature is implemented via Serverless Cloud Function (SCF). Before using it, you need to go to the COS console to create an **object concatenation** function on Application Integration - Object Concatenation.
2. After creating the function, set the function parameters according to the **instructions** in the operation column of the function list. The parameters are in **JSON string** format and detailed below.
   - If you select SCF authentication for the function, you need to call the Invoke API provided by SCF to run the object concatenation function, where the `ClientContext` parameter is passed in JSON format. For more information, please see Function Parameter Configuration Sample.
   - If you configure the function to be authentication free, you can directly send HTTP requests to the corresponding API to call the function.

Function Parameter Configuration Sample

> Note：
> In actual practice, delete the comments in the code.

```
{
"bucket": "examplebucket-1250000000", // Bucket of the output merged file for fin
al delivery
"region": "ap-guangzhou", // Region of the bucket of the output merged file for f
inal delivery
"key": "concat.txt", // Name of the output merged file for final delivery
/**
* `sourceList` specifies the list of source files that need to be packaged and is
a JSON array.
* Each item contains information such as the source file URL, and more parameters
may be extended in the future.
*
* If the source file list is excessively long, you can convert the value of `sour
ceList` into a JSON string,
* write the JSON string into a .json file, upload the file to COS, and specify th
```

```
e URL of the file in the `sourceConfigList` parameter.
*
* You only need to specify either `sourceList` or `sourceConfigList`.
*/
"sourceList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file1.txt"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file2.txt"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file3.txt"
}
],
"sourceConfigList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/sourceLis
t.json"
}
]
}
```

The parameters are described as follows:

Parameter	Description	Type	Required
bucket	Bucket of the output merged file for final delivery, in the format of `BucketName-APPID`, for example, `examplebucket-1250000000`.	String	Yes
region	Region of the bucket of the output merged file for final delivery. For the enumerated values, please see Regions and Access Endpoints.	String	Yes
key	Name (object name) of the output merged file for final delivery. It is the unique ID of an object in a bucket. For more information, please see Object Overview.	String	Yes
sourceList	List of source files. **`sourceList` and `sourceConfigList` cannot be empty at the same time.**	Array	Yes
sourceList[].url	URL of the source files.	String	Yes

Parameter	Description	Type	Required
sourceConfigList	Configuration file list of `sourceList`. If you do not want the request to carry the entire `sourceList`, you can convert the value of the `sourceList` into a JSON string, generate a JSON configuration file, upload the file to COS, and specify the URL of the file in the `sourceConfigList` parameter. * *`sourceList`* *and* *`sourceConfigList`* *cannot be empty at the same time.* *	Array	No
sourceConfigList[].url	URL of the `sourceList` configuration file.	String	No

Function Response Sample

```
{
"code": 0,
"message": "cos concat file success",
"data": {
"Location": "examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/concat.txt",
"Bucket": "examplebucket-1250000000",
"Key": "concat.txt",
"ETag": "\"152958e4f4bfded94c0b30f03343d6b8-1\""
}
}
```

Response parameters are described as follows:

Parameter	Description	Type
code	Business error code. `0`: the execution is successful. Other values: execution failed.	Number
message	Text description of the execution result. The message may be `null`.	String
data	Execution success information. If the execution is successful, this parameter contains the URL of the output merged file.	Object
error	Execution failure information. If the execution is successful, this parameter is `null`.	Object or String

Example

Parameter configuration

```
{
"bucket": "examplebucket-1250000000",
"region": "ap-guangzhou",
"key": "concat.txt",
"sourceList": [
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file1.txt"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file2.txt"
},
{
"url": "https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/file3.txt"
}
]
}
```

Structure of the final output of object concatenation

```
concat.txt
├── content of file1.txt
├── content of file2.txt
└── content of file3.txt
```