Alibaba Cloud
Object Storage Service

Best Practices

Issue: 20180807

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.
5. By law, all the content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade

secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).

6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 **Danger:** Resetting will result in the loss of user configuration data.
	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 **Warning:** Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 **Note:** Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	 **Note:** You can use **Ctrl** + **A** to select all files.
>	Multi-level menu cascade.	**Settings** > **Network** > **Set network type**
Bold	It is used for buttons, menus, page names, and other UI elements.	Click **OK**.
`Courier font`	It is used for commands.	Run the `cd /d C:/windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich` *{stand | slave}*

Contents

1 Application server

1.1 Set up direct data transfer for mobile apps

Background

In the era of mobile Internet, mobile apps upload more and more data every day. By handing off their data storage issues to OSS, developers can focus more on their app logic.

This article describes how to set up an OSS-based direct data transfer service for a mobile app in 30 minutes. Direct data transfer is a service that allows a mobile app to directly connect to OSS for data upload and download, while only sending the control traffic to the app server.

Advantages

Setting up an OSS-based direct data transfer service for a mobile app offers the following advantages:

- More secure upload/download method (temporary and flexible permission assignment and authentication).
- Low cost. Fewer app servers. The mobile app is directly connected to the cloud storage and only the control traffic is sent to the app server.
- High concurrency and support for a massive amount of users (OSS has massive bandwidth for uploading and downloading use).
- Elasticity (OSS's storage space can be expanded unlimitedly).
- Convenience. You can easily connect to the MTS - video multiport adapter, Image Service, CDN download acceleration, and other services.

The architecture diagram is as follows:

Details:

- Android/iOS mobile app, which is the app installed on the end user's mobile phone
- OSS, short for Alibaba Cloud Object Storage Service, which stores app-uploaded data (For more information, see the *OSS description on Alibaba Cloud website*.
- RAM/STS, which generates temporary access credentials
- App server, which is the background service developed for the Android/iOS mobile app and used to manage the tokens used for data uploading/downloading by the app and the metadata of the app-uploaded data.

Steps:

1. Request for a temporary upload credential from the app server.

   The Android/iOS app cannot store AccessKeyID/AccessKeySecret directly, which may cause the risk of information leakage. Therefore, the app must request a temporary upload credential (a token) from the app server. The token is only valid for a certain period. For example, if a token is set to be valid for 30 minutes (editable by the app server), then the Android/iOS app can use this token to upload/download data to/from the OSS within the next 30 minutes. 30 minutes later, the app must request a new token to upload/download data.
2. The app server checks the validity of the preceding request and then returns a token to the app .
3. After the phone receives this token, it can upload or download data from the OSS.

This article mainly describes the content in the red circle and blue circle of the following figure.

- The blue circle shows how the app server generates a token.
- The red circle shows how the Android/iOS app receives the token.

The results are:

You can scan the QR code to install the sample app, as shown in the following figure. This tool is developed on Android. The app server in this document can also be used on iOS.

The interface for connecting the sample app to the OSS is shown in the following figure:

Note:
The address displayed in the app server is a sample address. You can deploy the app server on your own by referring to the STS app server code at the end of the article.

- App server: the background app server corresponding to the mobile app.
- Upload bucket: the bucket to which the mobile app uploads data.
- Region: the region in which a bucket is uploaded.

Steps for using the sample app:

- Click **Select Image** and upload an object to the OSS.
- You can choose normal upload or resumable upload.

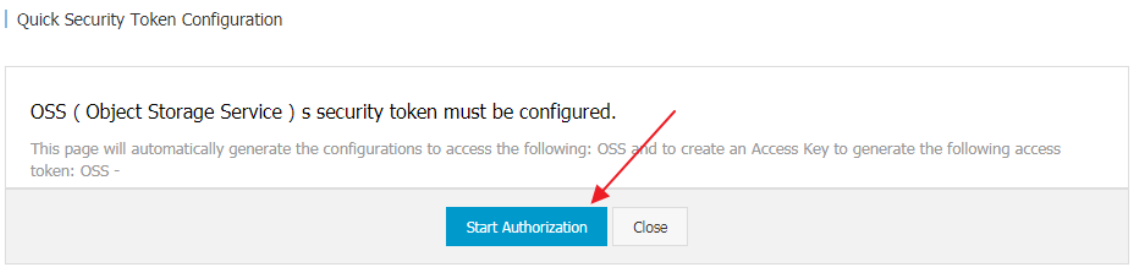
Note:
Resumable upload is recommended in poor network environments. You can use Image Service to scale down and add a watermark to the image to be uploaded. During initial use, do not modify the server URL and bucket name.

Prerequisites for setting up direct data transfer service

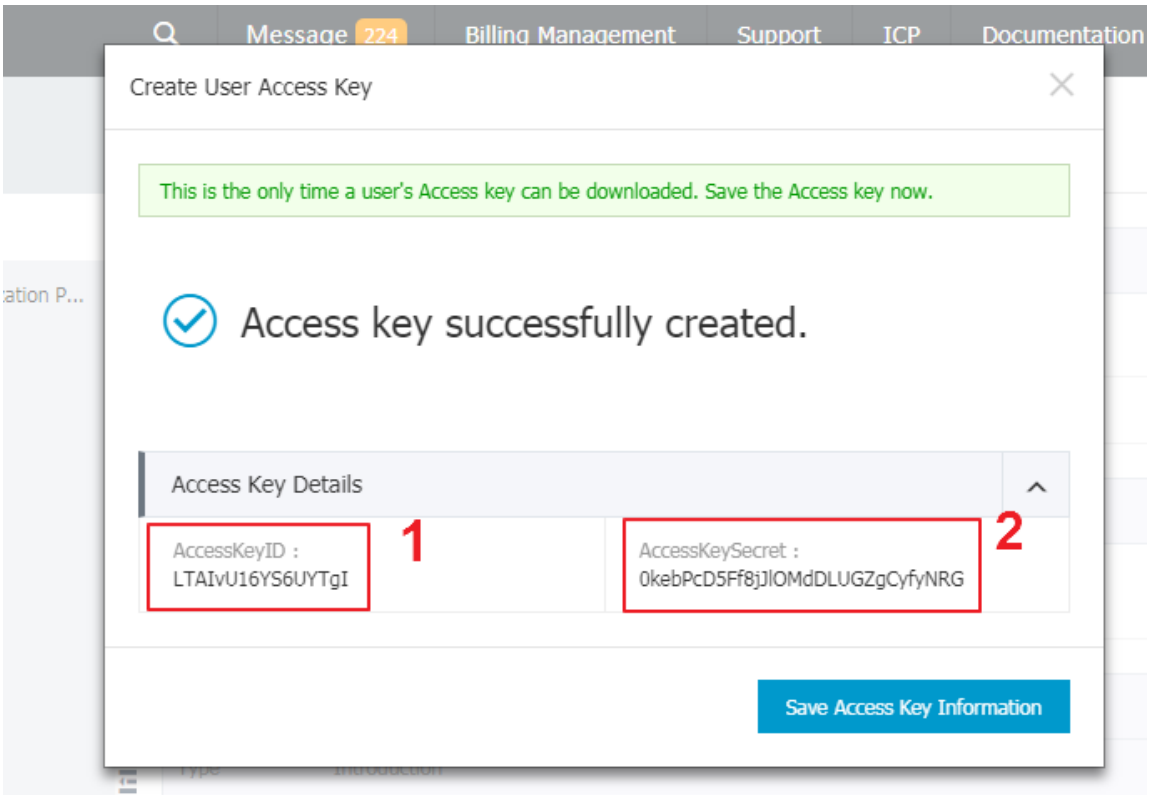
Preparations for setting up direct data transfer service:

1. *Activate the OSS service* and *create a bucket*.
2. Activate the STS service.
   a. Log on to the *OSS console*.
   b. On the OSS **Overview** page, find the **`Basic Settings`** area, and click **Security Token**, as shown in the following figure.
   c. Enter the **Quick Security Token Configuration** page.

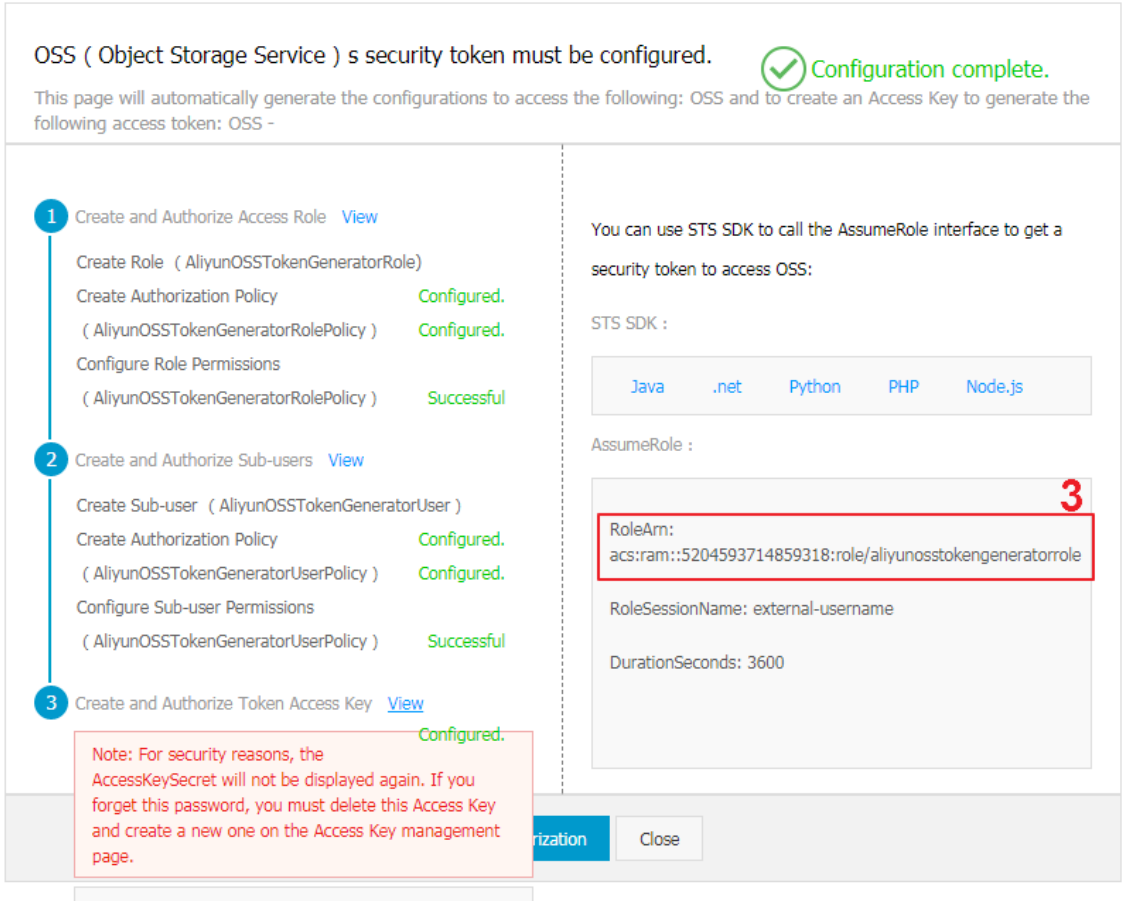
Note:
If RAM has not yet been activated, a prompt box to activate RAM appears. Click **Activate** and perform real-name verification. After the verification is finished, the following page appears. Click **Start Authorization**.



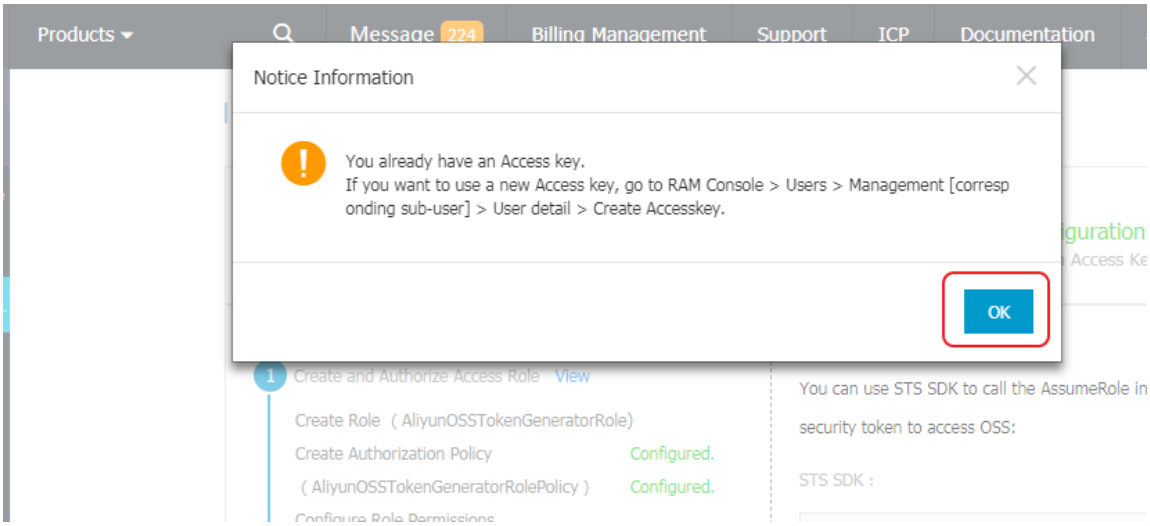
   d. The system performs authorization automatically. Be sure to save the parameters in the three red boxes in the following figures. Click **Save Access Key Information** to close the dialog box and complete STS activation.



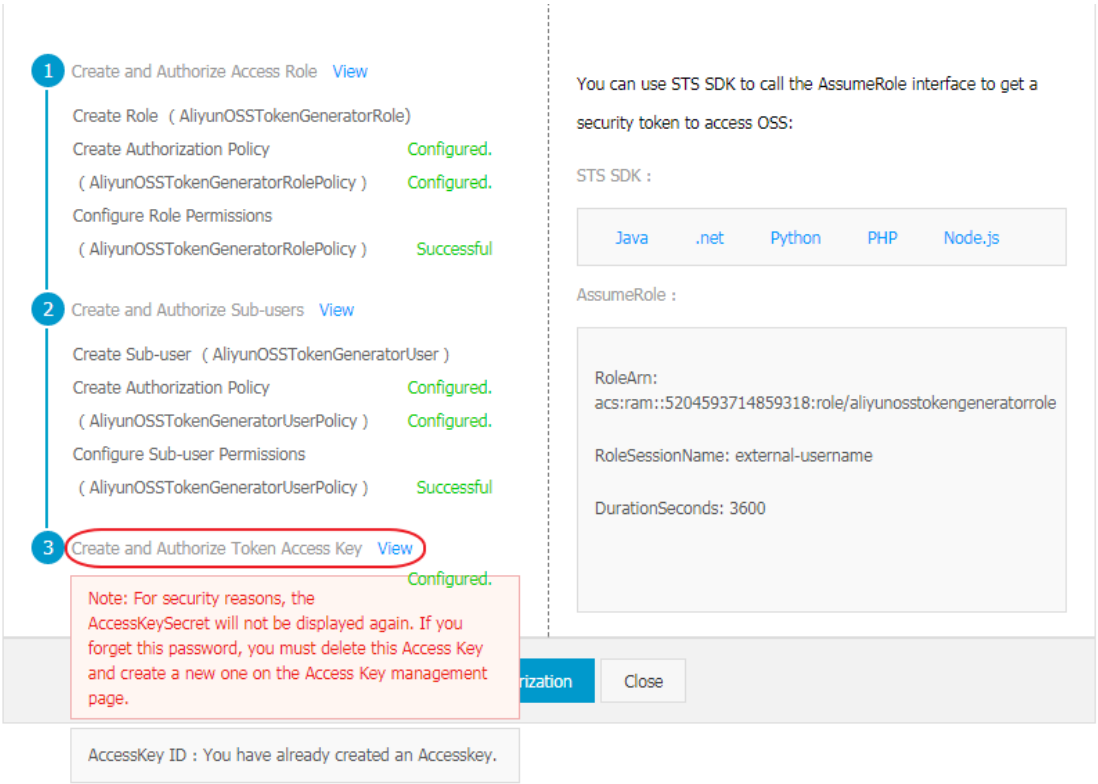
Quick Security Token Configuration



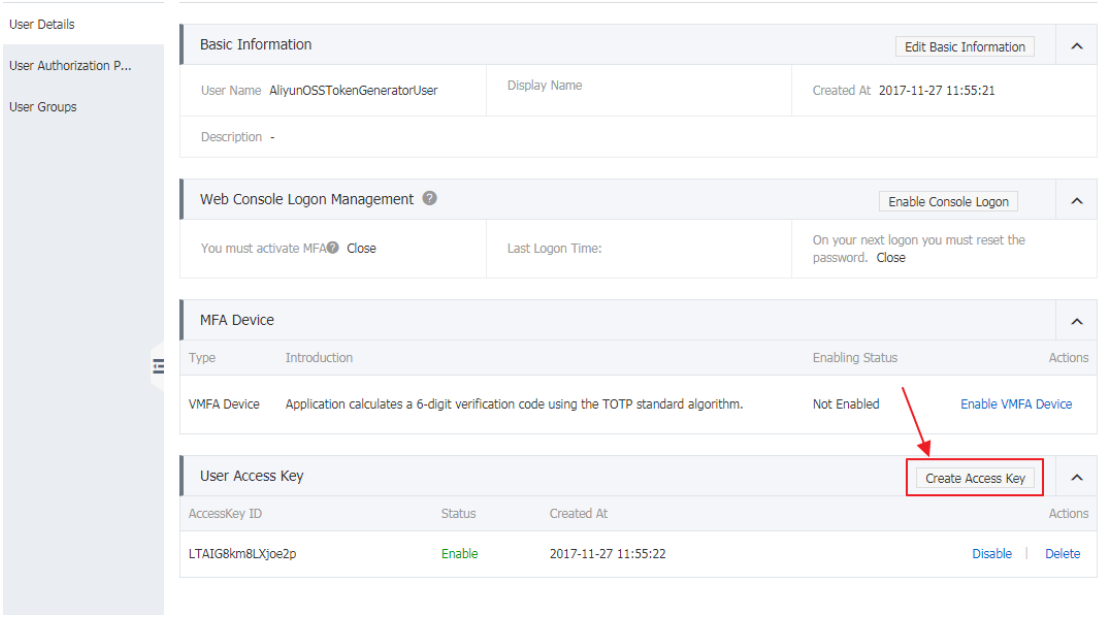
e. If you have already created an AccessKeyId/AccessKessKeySecret, the following prompt window appears:



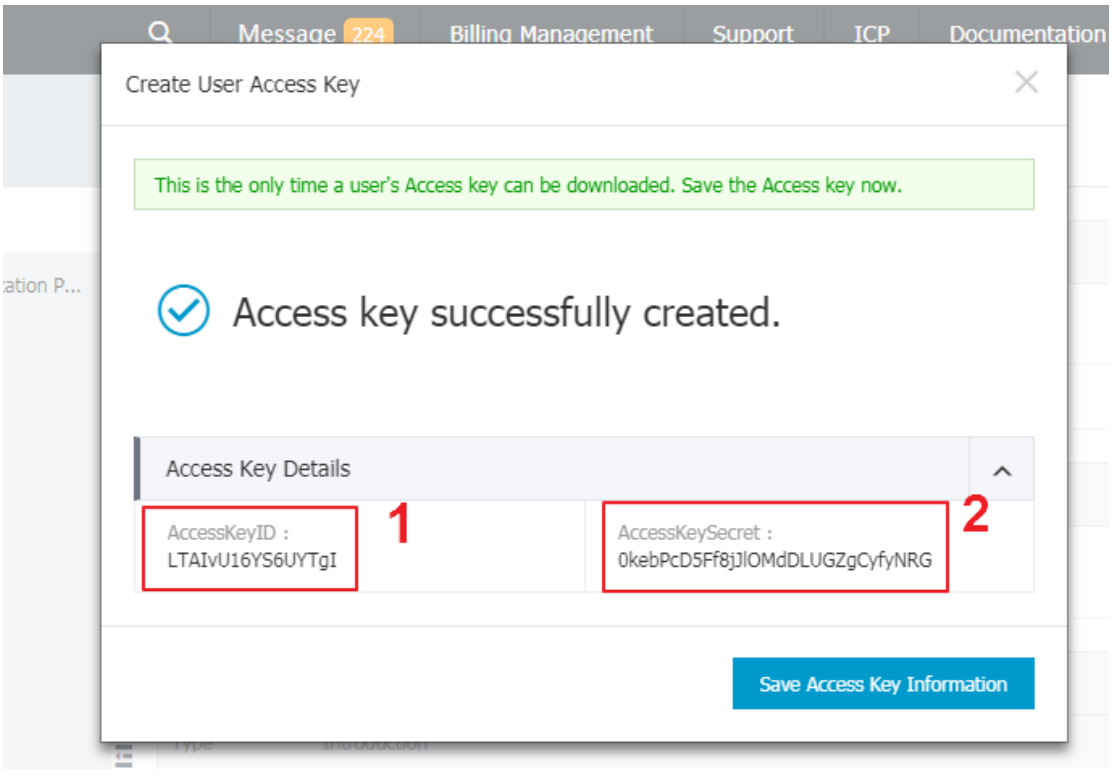
- Click **View**, as shown in the following figure.

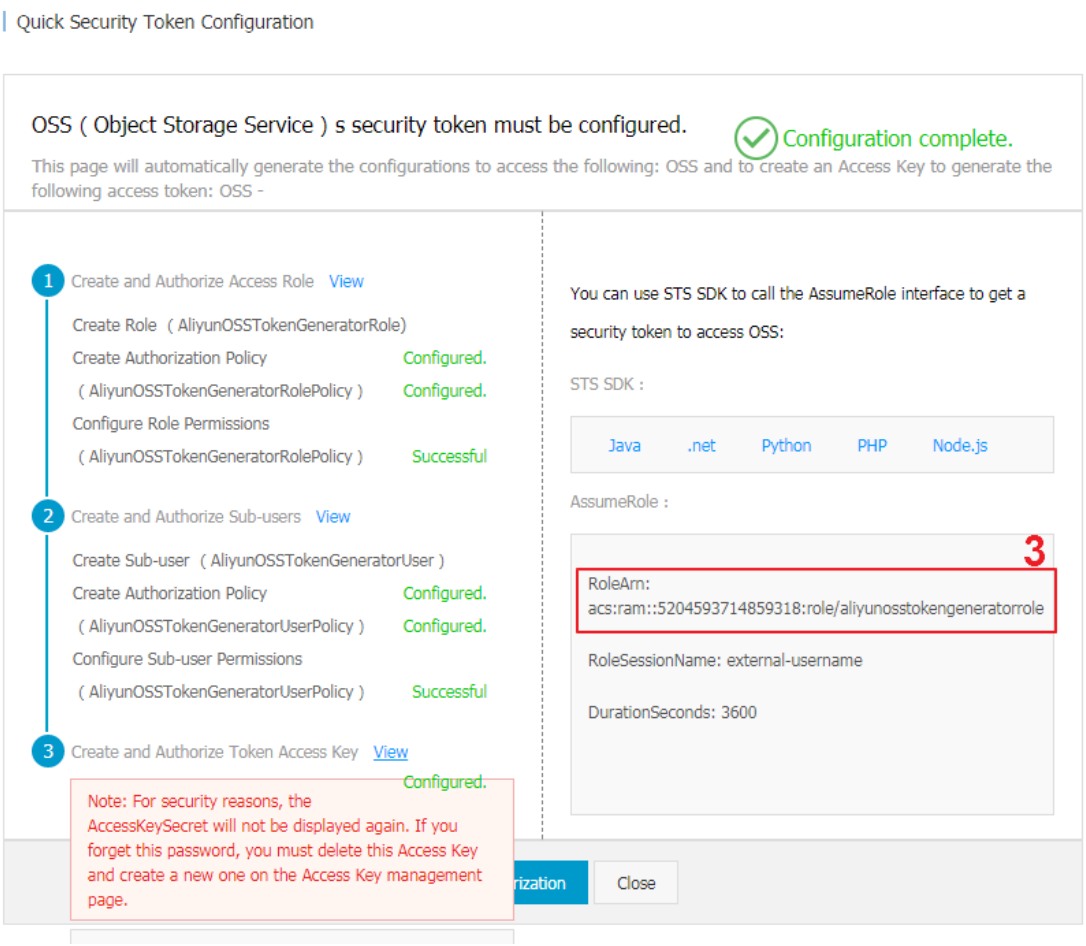


- Click **Create Access Key**, as shown in the following figure.



- Record parameters 1, 2, and 3, as shown in the following figure.





- Once you have saved the three parameters, STS activation is complete.

Set up an app server

Configuration of sample app server

> **Note:**
> The app in this example is written in PHP. You may write your app in your preferred language, such as Java, Python, Go, Ruby, Node.js, or C#.

This tutorial provides development sample programs available for download in multiple languages . The download addresses are shown at the end of this article.

The downloaded package in each language contains a configuration file named config.json.

```
"AccessKeyID" : "",
"AccessKeySecret" : "",
"RoleArn" : "",
"TokenExpireTime" : "900",
```

```
"PolicyFile": "policy/all_policy.txt"
```

Note:

- **1.** AccessKeyID: Set it to parameter 1 marked with a red box in the preceding figure.
  2. AccessKeySecret: Set it to parameter 2 marked with a red box in the preceding figure.
  3. RoleArn: Set it to parameter 3 marked with a red box in the preceding figure.
  4. TokenExpireTime indicates the expiration time of the token obtained by the Android/iOS app. The minimum value is 900s. The default value can be retained.
  5. PolicyFile indicates the file that lists the permissions the token grants. The default value can be retained.

This document has provided three token files defining the most common permissions in the policy directory. They are:

- all_policy.txt: specifying a token that grants permissions to create or delete a bucket, or upload , download, or delete a file for this account.
- bucket_read_policy.txt: specifying a token that grants permission to read the specified bucket for this account.
- bucket_read_write_policy.txt: specifying a token that grants permission to read and write the specified bucket for this account.

If you want to create a token to grant read and write permissions for the specified bucket, replace $BUCKET_NAME in the bucket_read_policy.txt and bucket_read_write_policy.txt files with the name of the specified bucket.

- Explanation of the formats of returned data:

```
//Correct result returned

    "StatusCode":200,
    "AccessKeyId":"STS. 3p***dgagdasdg",
    "AccessKeySecret":"rpnwO9***tGdrddgsR2YrTtI",
   "SecurityToken":"CAES+wMIARKAAZhjH0EUOIhJMQBMjRywXq7MQ/cjLYg80Aho
1ek0Jm63XMhr9Oc5s·∂·∂3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+mRR9KAbHUe
fqH+rdjZqjTF7p2m1wJXP8S6k+G2MpHrUe6TYBkJ43GhhTVFMuM3BZajY3VjZWOXBI
ODRIR1FKZjIiEjMzMzE0MjY0NzM5MTE4NjkxMSoLY2xpZGSSDgSDGAGESGTE
TqOio6c2RrLWRlbW8vKgoUYWNzOm9zczoqOio6c2RrLWRlbW9KEDExNDg5Mz
AxMDcyNDY4MThSBTI2ODQyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzMzMTQyNj
Q3MzkxMTg2OTExcglzZGstZGVtbzI=",
   "Expiration":"2015-12-12T07:49:09Z",

//Incorrect result returned

    "StatusCode":500,
    "ErrorCode":"InvalidAccessKeyId.NotFound",
```

```
    "ErrorMessage":"Specified access key is not found."
```

Explanation of correct result returned: (The following five variables comprise a token)

- StatusCode: The status indicates the result that the app retrieves the token. The app returns 200 for successful retrieval of the token.
- AccessKeyId: indicates the AccessKeyId the Android/iOS app obtains when initializing the OSS client.
- AccessKeySecret: indicates the AccessKeySecret the Android/iOS app obtains when initializing the OSS client.
- SecurityToken: indicates the token the Android/iOS app initializes.
- Expiration: indicates the time when the token expires. The Android SDK automatically determines the validity of the token and retrieves a new one as needed.

Explanation of incorrect result returned:

- StatusCode: The status indicates the result that the app retrieves the token. The app returns 500 for unsuccessful retrieval of the token.
- ErrorCode: indicates the error causes.
- ErrorMessage: indicates the detailed information about the error.

- Method for running sample code:
  - For PHP, download and unzip a pack, modify the config.json file, run php sts.php to generate a token, and deploy the program to the specified address.
  - For Java (based on Java 1.7), after downloading and unzipping a pack,

    Run this command: java -jar oss-token-server.jar (port). If you run java –jar oss-token-server .jar without specifying a port, the program listens to Port 7080. To change the listening port to 9000, run java –jar app-token-server.jar 9000. Specify the port number as needed.

How to upload files from your app to oss

1. After setting up the app server, write down the server address, which is `http://abc.com:8080`. Then, replace the app server address in the sample project with this address.
2. Specify the bucket and region for the upload in the sample apps.
3. Click **Set** to load the configuration.
4. Select an image file, set the object name to upload to OSS, and select Upload. Now you can experience the OSS service on Android. Data from the Android app can be uploaded directly to OSS.

5. After the upload is complete, check that the data is on OSS.

Explanation of core code

OSS initialization

- Android version

```
// We recommend you use OSSAuthCredentialsProvider because it
automatically updates expired tokens.
String stsServer = "http://abc.com:8080 is an example of an
application server address."
Osscredentialprovider credentialProvider = new ossauthcredentialspr
ovider (stsserver );
//config
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); /// Connection time-out. The
default value is 15 seconds.
conf.setSocketTimeout(15 * 1000); // Socket time-out. The default
value is 15 seconds.
conf.setMaxConcurrentRequest(5); // The maximum number of concurrent
 requests. The default value is 5.
conf.setMaxErrorRetry(2); // The maximum number of retry attempts
after each failed attempt. The default value is 2.
OSS oss = new OSSClient(getApplicationContext(), endpoint,
credentialProvider, conf);
```

- iOS version

```
OSSClient * client;

// We recommend you use OSSAuthCredentialProvider because it
automatically updates expired tokens.
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider
alloc] initWithAuthServerUrl:@"application server address, such as
http://abc.com:8080"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credential
Provider:credential];
```

Download source code

Example program

- Sample app source code for Android: *download address*
- Sample app source code for iOS: *download address*

Download sample code of app server

- PHP: *download address*
- Java: *download address*
- Ruby: *download address*
- node.js: *download address*

1.2 Permission control

This document elaborates how to configure different policies to implement different permission controls based on the app server mentioned in *Set up direct data transfer for mobile apps* by taking the app-base-oss bucket in the Shanghai region as an example.

Note:

- The following illustration assumes you have already activated STS and have thoroughly read the *Set up direct data transfer for mobile apps* document.
- The policies mentioned in the following content are covered in the specified policy file in the config.json file mentioned in the previous section.
- The operations on OSS upon retrieving the STS token indicate the process of specifying the policy for the app server, the app server retrieving a temporary credential from the STS and the app using the temporary credential to access OSS.

Common policies

- Full authorization policy

  For the ease of demonstration, the default policy is shown as follows. This policy indicates that the app is allowed to perform any operation on OSS.

  Note:
  This policy is neither secured nor recommended to use for mobile apps.

```
{
    "Statement": [
      {
        "Action": [
          "oss:*"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:*"]
      }
    ],
    "Version": "1"
  }
```

Operations on OSS upon retrieving STS token	Result
List all created buckets.	Successful
Upload the object without a prefix, test.txt.	Successful

Operations on OSS upon retrieving STS token	Result
Download the object without a prefix, test.txt.	Successful
Upload the object with a prefix, user1/test.txt.	Successful
Download the object with a prefix, user1/test.txt.	Successful
List the object without a prefix, test.txt.	Successful
List the object with a prefix, user1/test.txt.	Successful

- Read-only policies with or without any prefixes

  This policy indicates the app can list and download all objects in the bucket app-base-oss.

```
{
      "Statement": [
        {
          "Action": [
            "oss:GetObject",
            "oss:ListObjects"
          ],
          "Effect": "Allow",
          "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:
app-base-oss"]
        }
      ],
      "Version": "1"
    }
```

Operations on OSS upon retrieving STS token	Result
List all created buckets.	Failed
Upload the object without a prefix, test.txt.	Failed
Download the object without a prefix, test.txt.	Successful
Upload the object with a prefix, user1/test.txt.	Failed
Download the object with a prefix, user1/test.txt.	Successful
List the object without a prefix, test.txt.	Successful
List the object with a prefix, user1/test.txt.	Successful

- Read-only policies with a specified prefix

  This policy indicates the app can list and download all objects with the prefix of **user1/** in the bucket **app-base-oss**. However, the policy does not specify to download any objects

with another prefix. By this way, different apps corresponding to different prefixes are spatially isolated in the bucket.

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
  }
```

Operations on OSS upon retrieving STS token	Result
List all created buckets.	Failed
Upload the object without a prefix, test.txt.	Failed
Download the object without a prefix, test.txt.	Failed
Upload the object with a prefix, user1/test.txt.	Failed
Download the object with a prefix, user1/test.txt.	Successful
List the object without a prefix, test.txt.	Successful
List the object with a prefix, user1/test.txt.	Successful

- Write-only policies with no specified prefixes

  This policy indicates that the app can upload all objects in the bucket app-base-oss.

```
{
    "Statement": [
      {
        "Action": [
          "oss:PutObject"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
```

```
  }
```

Operations on OSS upon retrieving STS token	Result
List all created buckets.	Failed
Upload the object without a prefix, test.txt.	Successful
Download the object without a prefix, test.txt.	Failed
Upload the object with a prefix, user1/test.txt.	Successful
Download the object with a prefix, user1/test.txt.	Successful
List the object without a prefix, test.txt.	Successful
List the object with a prefix, user1/test.txt.	Successful

- Write-only policies with a specified prefix

  This policy indicates the app can upload all objects with the user1/ prefix in the bucket app-base-oss. The app cannot upload any object with another prefix. In this way, different apps corresponding to different prefixes are spatially isolated in the bucket.

```
{
    "Statement": [
      {
        "Action": [
          "oss:PutObject"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
  }
```

Operations on OSS upon retrieving STS token	Result
List all created buckets.	Failed
Upload the object without a prefix, test.txt.	Failed
Download the object without a prefix, test.txt.	Failed
Upload the object with a prefix, user1/test.txt.	Failed
Download the object with a prefix, user1/test.txt.	Successful

Operations on OSS upon retrieving STS token	Result
List the object without a prefix, test.txt.	Failed
List the object with a prefix, user1/test.txt.	Failed

- Read/write policies with or without any prefixes

  This policy indicates that the app can list, download, upload, and delete all objects in the bucket `app-base-oss`.

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:PutObject",
          "oss:DeleteObject",
          "oss:ListParts",
          "oss:AbortMultipartUpload",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
  }
```

Operations on OSS upon retrieving STS token	Result
List all created buckets.	Failed
Upload the object without a prefix, test.txt.	Successful
Download the object without a prefix, test.txt.	Successful
Upload the object with a prefix, user1/test.txt.	Successful
Download the object with a prefix, user1/test.txt.	Successful
List the object without a prefix, test.txt.	Successful
List the object with a prefix, user1/test.txt.	Successful

- Read/write policies with a specified prefix

  This policy indicates the app can list, download, upload, and delete all objects with a prefix of `user1/` in the bucket `app-base-oss`. The policy does not specify to read or write any objects

with another prefix. In this way, different apps corresponding to different prefixes are spatially isolated in the bucket.

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:PutObject",
          "oss:DeleteObject",
          "oss:ListParts",
          "oss:AbortMultipartUpload",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
  }
```

Operations on OSS upon retrieving STS token	Result
List all created buckets.	Failed
Upload the object without a prefix, test.txt.	Failed
Download the object without a prefix, test.txt.	Failed
Upload the object with a prefix, user1/test.txt.	Successful
Download the object with a prefix, user1/test.txt.	Successful
List the object without a prefix, test.txt.	Successful
List the object with a prefix, user1/test.txt.	Successful

Summary

With the help of preceding examples, we can understand that:

- You can create different policies for various app scenarios and then achieve differentiated permission control for different apps through slight modifications on the app server.
- You can also optimize apps to save the process of making another request to the app server before the STS token expires.
- Tokens are actually issued by the STS. An app server customizes a policy, requests for a token from the STS, and then delivers this token to the app. Here, token is only a shorthand expression. However, a "token" actually contains an "AccessKeyId", an "AccessKeySecret", an

"Expiration" value, and a "SecurityToken". These are used in the SDK provided by OSS to the app. For more information, see the implementation of the respective SDK.

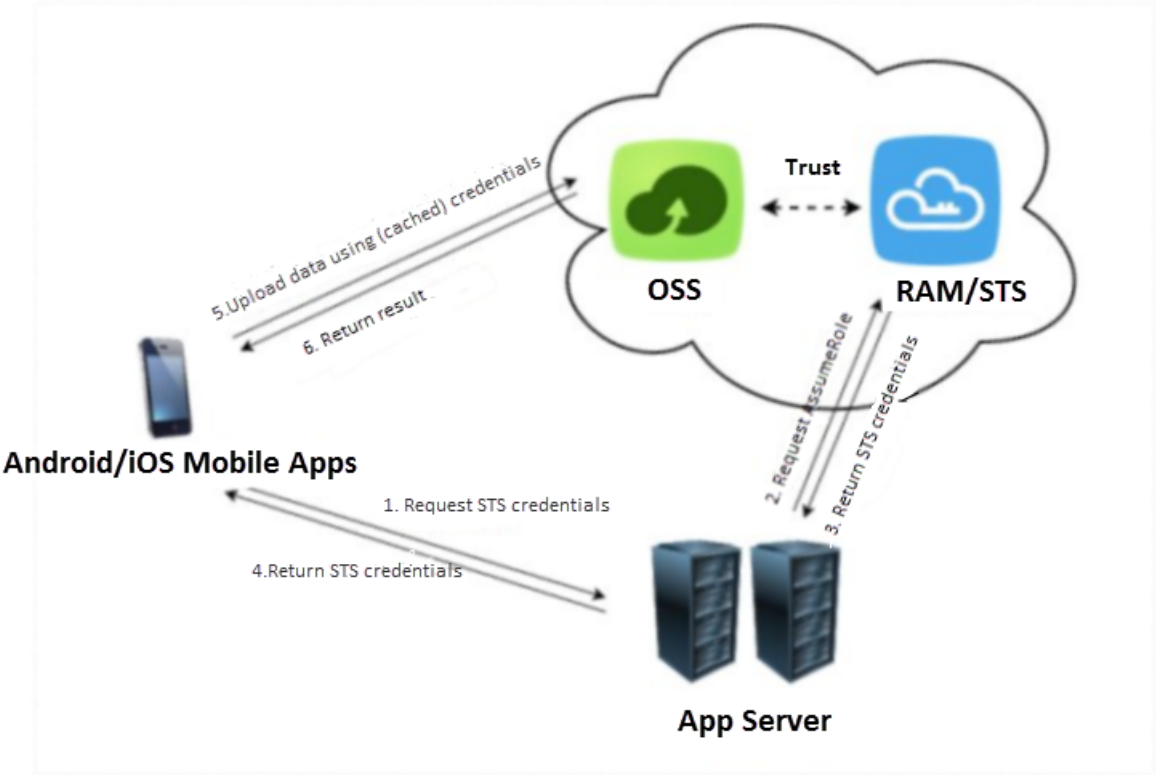
More references:

- *How to use RAM and STS in OSS*
- *RAM documentation* and *STS documentation*

1.3 Set up data callback for mobile apps

Background

Setting up direct data transfer for mobile apps describes how to set up OSS-based direct data transfer for mobile apps in 30 minutes. The following flowchart describes mobile app development:



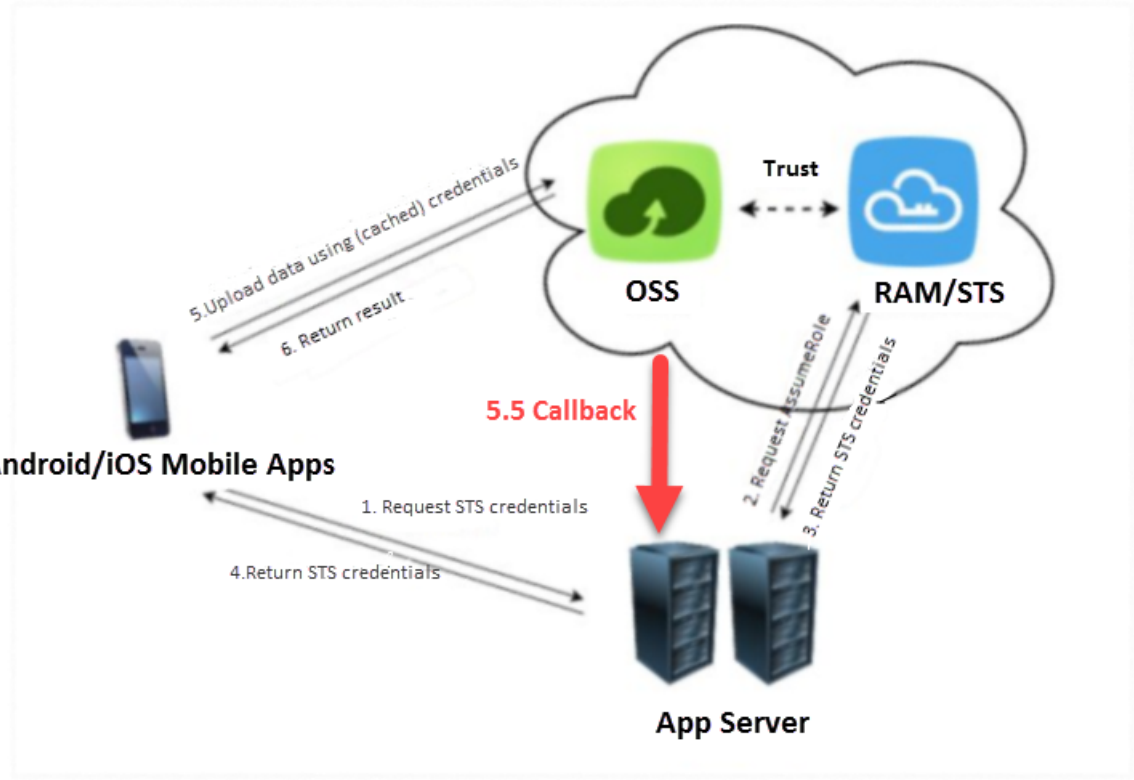
Role:

- The app server generates an STS credential for the Android/iOS mobile app.
- The Android/iOS mobile app applies for the STS credential from the app server and then uses the STS credential.
- The OSS processes requests from the Android/iOS mobile app.

After performing Step 1 (apply for an STS credential) in the preceding flowchart, the Android/iOS mobile app, can perform Step 5 (use the STS credential to upload data to OSS) repeatedly. In

this case, the app server does not know what data the app is uploading, and the app developer cannot manage the uploaded data. Is there any way to make the app server be aware of the data uploaded by the Android/iOS mobile app?

In this case, the OSS data callback service can be used to tackle these type of issues. You can see the following flowchart:



OSS triggers a callback after receiving data from the Android/iOS mobile app (Step 5 in the preceding flowchart) but before returning the upload result to the app (Step 6). The callback is marked as Step 5.5. OSS calls back data from the app server and obtains the content returned by the app server. Then OSS returns the content to the Android/iOS mobile app. For more information, see *Callback API Documentation*.

Data callback function

- Retrieving basic information about the data uploaded to the app server

  The following table shows the basic information. One or more of the following variables are returned, and the format of returned content is specified when the Android/iOS mobile app uploads data.

System variable	Meaning
bucket	Storage space (bucket) to which the mobile app uploads data
object	File name saved on OSS for the data uploaded by the mobile app
etag	etag of the uploaded file. It is the etag field returned to the mobile app
size	Size of the uploaded file
mimeType	Resource type
imageInfo.height	Image height
imageInfo.width	Image width
imageInfo.format	Image format, for example, JPG and PNG (only for recognized images)

- Transferring information through custom variables

  If you are a developer and want to know the app version, OS version, location, and mobile phone model of the user who is uploading data, you can specify the Android/iOS mobile app client to send the preceding variables when uploading files. For example,

  - x:version indicates app version.
  - x:system indicates OS version.
  - x:gps indicates location.
  - x:phone indicates mobile phone model.

    These values are attached when the Android/iOS mobile app uploads data to OSS. Then OSS includes the values in the CallbackBody and sends them to the app server. In this way , the information is transferred to the app server.

Data callback setup for the mobile app client

To enable OSS to trigger a callback when receiving an upload request, the mobile app must include the following two items in the request:

- callbackUrl indicates the app server to which data is called back, for example, `http://abc.com/callback.php`. Note that the server address must be accessible through the Internet.
- callbackBody indicates the content to be called back and sent to the app server. The content can include one or more of the variables OSS returns to the app server.

For example, assume that the data is called back and sent to the app server at `http://abc.com/callback.php`. You want to obtain the name and size of the file uploaded by the mobile phone. The defined variable "photo" gets the mobile phone model, and the variable "system" gets the OS version.

Two samples of upload callbacks are listed as follows:

- Data callback sample code for iOS apps:

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
request.uploadingFileURL = [NSURL fileURLWithPath:@<filepath>"];
// Set callback parameters
request.callbackParam = @{
                          @"callbackUrl": @"http://abc.com/callback.
php",
                          @"callbackBody": @"filename=${object}&size
=${size}&photo=${x:photo}&system=${x:system}"
                          };
// Set custom variables
request.callbackVar = @{
                        @"x:photo": @"iphone6s",
                        @"x:system": @"ios9.1"
                        };
```

- Data callback sample code for Android apps:

```
PutObjectRequest put = new PutObjectRequest(testBucket, testObject,
uploadFilePath);
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("application/octet-stream");
put.setMetadata(metadata);
put.setCallbackParam(new HashMap<String, String>() {
    {
        put("callbackUrl", "http://abc.com/callback.php");
        put("callbackBody", "filename=${object}&size=${size}&photo=
${x:photo}&system=${x:system}");
    }
});
put.setCallbackVars(new HashMap<String, String>() {
     {
         put("x:photo", "IPOHE6S");
         put("x:system", "YunOS5.0");
     }
});
```

Data callback requirements for the app server

- You must deploy a service for receiving POST requests. This service must have a public address, for example, `www.abc.com/callback.php` (or an Internet IP address); otherwise, OSS cannot access this address.

- You must set the format of custom content returned to OSS to JSON. OSS delivers the content received from the app server as it is to the Android/iOS mobile app. (The Response header returned to OSS must carry the Content-Length header.)

The last section provides sample callback programs based on multiple programming languages, together with the download links and running methods.

Callback request received by the app server

The packet of a callback request the app server receives from OSS is as follows (the data varies with different URLs and callback content):

```
POST /index.html HTTP/1.0
Host: 121.43.113.8
Connection: close
Content-Length: 81
Content-Type: application/x-www-form-urlencoded
User-Agent: ehttp-client/0.0.1
authorization: kKQeGTRccDKyHB3H9vF+xYMSrmhMZjzzl2/kdD1ktNVgbWEfYTQG0G2
SU/RaHBovRCE8OkQDjC3uG33esH2txA==
x-oss-pub-key-url: aHR0cDovL2dvc3NwdWJsaWMuYWxpY2RuLmNvbS9jYWxsYmFja1
9wdWJfa2V5X3YxLnBlbQ==
filename=test.txt&size=5&photo=iphone6s&system=ios9.1
```

For more information, see *Callback API Documentation*.

How does the app server determine whether a callback request is sent by OSS?

The app server must determine whether a callback request is from OSS because the app server may receive invalid requests that affect its normal logic when the app server has a malicious callback during a network attack.

To determine request validity, the app server verifies the RSA checksum using the `x-oss-pub-key-url` and `authorization` parameters in the content OSS sends to the app server. Only requests that pass RSA checksum verification are sent by OSS. The sample programs in this document also provides implementation results, for your reference.

How does the app server process the received callback request?

After verifying a request from OSS, the app server processes the request based on its content. The Android/iOS mobile app specifies the format of the callback content when uploading the data, for example:

```
filename=test.txt&size=5&photo=iphone6s&system=ios9.1
```

The app server parses the OSS-returned content to obtain the expected data. Then the app server stores the data for subsequent management.

How does the app server return the callback request to OSS?

- The returned status code is 200.
- The returned content must use the JSON format.
- The returned content must carry the Content-Length header.

How does OSS process the content returned by the app server?

There are two scenarios:

- In case that the app server fails to receive the callback request or is not accessible, OSS returns a 203 status code to the Android/iOS mobile app. However, the uploaded data is already saved to OSS.
- If the app server receives a callback request and returns the correct status code, OSS returns content received from the app server as it is to the Android/iOS mobile app along with a 200 status code.

Sample callback programs for downloading

The sample program shows how to check the signature received by the application server. You must add the code for parsing the format of the callback content received by the application server .

- Java version:
  - *Download address*.
  - Running method: Extract the archive and run `java -jar oss-callback-server-demo.jar 9000` (9000 is the port number and can be changed as required).

    Note:
    This jar runs on java 1.7. If any problem occurs, you may make changes based on the provided code. This is a maven project.
- PHP version:
  - *Download address*.
  - Running method: Deploy the program to an Apache environment. Due to the characteristics of the PHP language, retrieving headers depends on the environment. You may make modifications to the example based on your own environment.
- Python version:
  - *Download address*.

- Running method: Extract the archive and directly run python callback_app_server.py. The program implements a simple HTTP server. To run this program, you may need to install the system environment on which the RSA depends.

- Ruby version:
  - *Download address*.
  - Running method: ruby aliyun_oss_callback_server.rb

2 Direct upload to OSS from Web

2.1 Overview of direct transfer on Web client

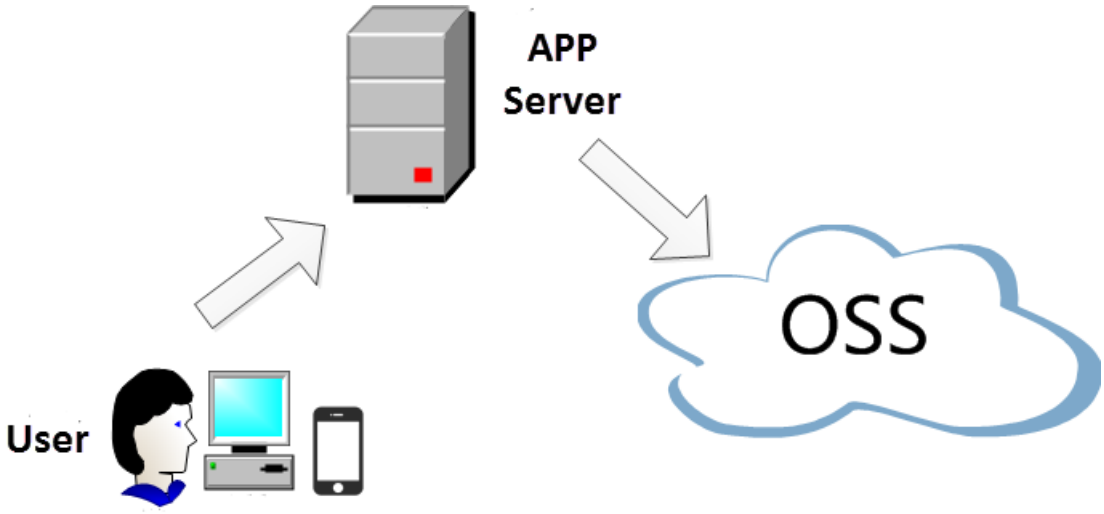
Purpose

This document with the help of two examples, elaborates how to transfer a file in HTML form directly to OSS.

- Example 1: Describes how to add a signature on a server (PHP) and then upload the file directly to OSS using a form.
- Example 2: Describes how to add a signature on the server (PHP), and set the callback upon uploading on the server. Then, upload the form directly to OSS. After that, OSS calls back the application server and returns the result to the user.

Background

Every OSS user may use the upload service. This is because, the data is uploaded using Web pages and it includes some HTML5 pages in some apps. The demand to upload these services is strong. Many users choose to upload files to the application servers through browsers/apps, and then the application server uploads the files to OSS.



However, the preceding method has following limitations:

- Low uploading speed: Intially, files are uploaded to the application server, and then to OSS. Therefore, the workload of transmission over the Internet is doubled. If the data is transferred directly to OSS without passing through the application server, the speed increases significan

tly. Moreover, OSS uses BGP bandwidth, thus ensuring a high speed for operators in different places.

- Poor scalability: As the number of users increases in future, the application server may constitute a bottleneck.
- High cost: The traffic consumed for uploading files directly to OSS is free of charge. If data is uploaded directly to OSS without passing through the application server, the costs of several application servers can be saved.

Basic

The application server uses PHP script language to return the signature. Click here for the example.

Advanced

The application server returns the signature using the PHP script language and implements uploading callback. Click here for the example.

2.2 Direct transfer after adding a signature on the server

Background

Direct signature by JS clients has a serious hidden security hazard in that the OSS AccessId /AcessKey are exposed on the front-end which may be accessible to others. This document explains how to get a signature from and upload a policy to the backend PHP code.

The logic for uploading a signature to the backend is as follows:

1. The client directly uploads the obtained signature to the OSS.
2. Signature sample uploaded to the backend

Signature sample uploaded to the backend

Download sample:

- Click *here* to download a test sample on a PC browser.
- You can test whether the upload was effective on a mobile phone. You can use a mobile phone app (such as WeChat, QQ, and mobile browsers) to scan the QR code. This is not an advertisement, but a QR code for the preceding URL. This operation allows you to see whether the service works as intended on mobile phones.

Download code:

Click *here* to download the code.

This example adopts the backend signature, and uses PHP language.

- Click *here* for the example of a backend signature using Java language.
- Click *here* for the example of a backend signature using Python language.
- Click *here* for the example of a backend signature using Go language.

Usage of other languages:

1. Download the corresponding language example.
2. Modify the example code. For instance, set the listening port, and then start running.
3. At upload.js in `oss-h5-upload-js-php.zip`, change the variable serverUrl to the address configured at step 2. For example, serverUrl = `http://1.2.3.4:8080` or serverUrl=`http://abc.com/post/`.

Principle of constructing a Post signature on the server end

The OSS PostObject method is used for uploads. You can construct a PostObject request in the browser using Plupload and send the request to the OSS. Signatures are implemented on the server in PHP. In the same principle, the server can be compiled in Java, .NET, Ruby, Go, or Python language. The core logic is to construct a Post signature. The Java and PHP examples are provided here. The following steps are required:

1. The webpage requests the signature through JavaScript from the server end.
2. After JavaScript gets the signature, it uploads the signature to the OSS through Plupload.

Implementation

1. Populate the fields with your ID, key, and bucket.

   Modify php/get.php:

   - Set the variable $id to AccessKeyId.
   - Set $key to AccessKeySecret.
   - Set $host to bucket+endpoint.

   Note:

   For information on the endpoint, see *Basic OSS concepts*.

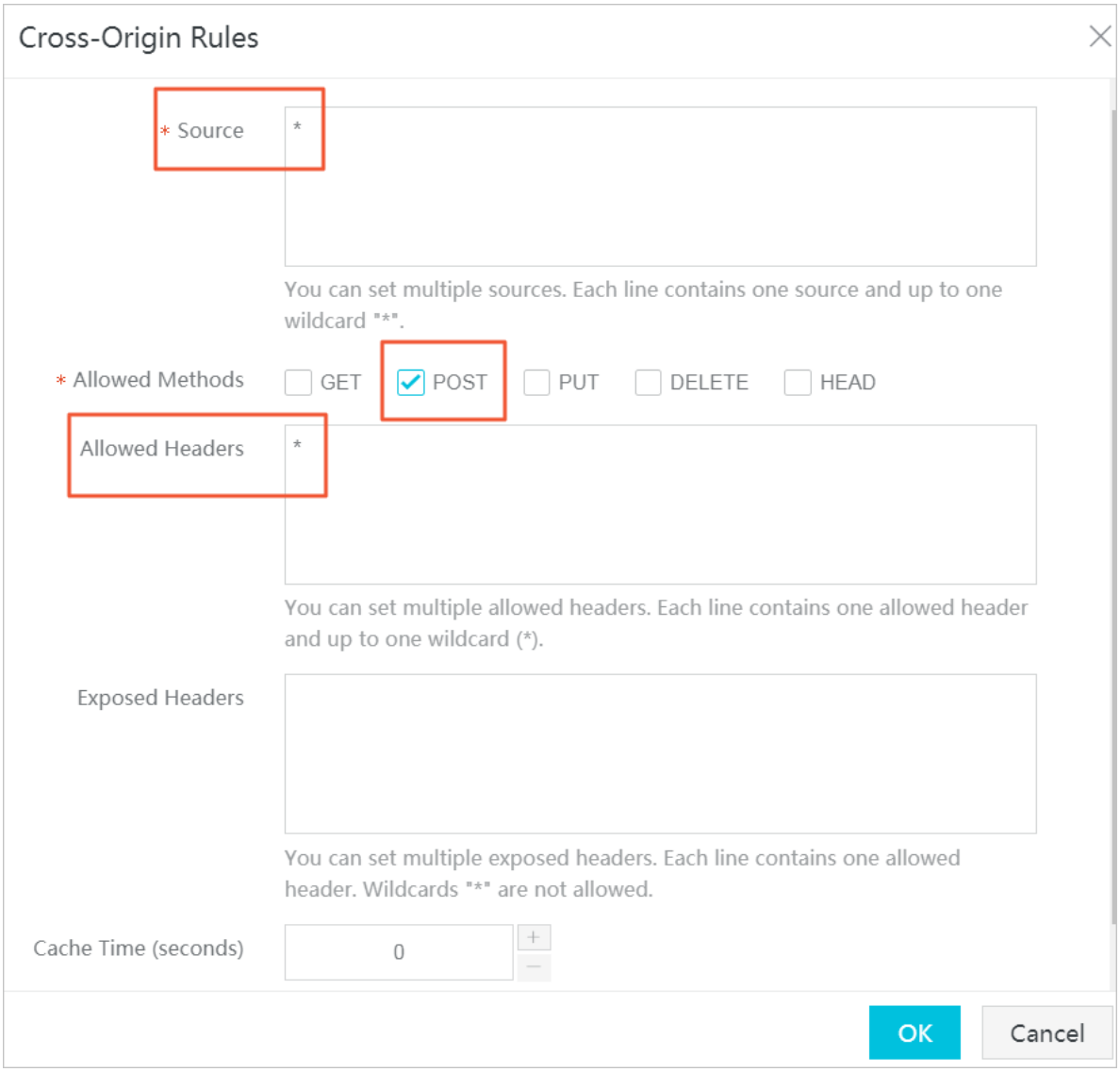
```
$id= 'xxxxxx';
  $key= 'xxxxx';
```

```
$host = 'http://post-test.oss-cn-hangzhou.aliyuncs.com
```

2. You must set CORS for the bucket to guarantee browser safety.

> **Note:**
> Make sure that the CORS settings of the bucket attribute support the POST method. This is because, HTML directly uploads data to OSS and produces a cross-origin request in the process. Hence, you must allow cross-original requests in the bucket attributes.

For procedure, see *Set CORS*. The settings are as follows:



> **Note:**
> In earlier-version IE browsers, Plupload is executed in flash. You must set crossdomain.xml. For the setting method, *click here*.

Details of core logic

Set random object names

You often need to name uploaded objects randomly, if they have the same suffix as the objects on the client. In this example, two radios are used to differentiate. If you want to fix the settings to apply random names to the uploaded objects, you can change the function to the following:

```
function check_object_radio() {
    g_object_name_type = 'random_name';
```

If you want to set uploads to the user's objects, you can change the function to the following:

```
function check_object_radio() {
    g_object_name_type = 'local_name';
```

Set the upload directory

The upload directory is specified by the server end (in PHP), which enhances security. Each client is only allowed to upload objects to a specific directory. This guarantees security by isolation. The following code changes the upload directory address to abc/ (the address must end with `/`).

```
$dir = 'abc/';
```

Set the filtering conditions for uploaded objects

We often need to set the filtering conditions for uploads. For example, only allowing image uploads, setting the size of uploaded objects, and disallowing repeated uploads. You can use the filters parameter for this.

```
var uploader = new plupload.Uploader({
    ……
    filters: {
        mime_types : [ //Only images and zip objects are allowed to be
 uploaded
        { title : "Image files", extensions : "jpg,gif,png,bmp" },


        max_file_size : '400kb', //Only objects with a maximum size of
 400 KB are allowed to be uploaded.
        MAID: True //Repeated objects are not allowed to be selected
```

Use the Plupload attribute filters to set filtering conditions.

Explanations of the preceding setting values:

- mime_types: Restrict extensions of the uploaded objects.
- max_file_size: Restrict the size of the uploaded objects.

- prevent_duplicates: Restrict repeated uploads.

> **Note:**
> The filter conditions are not required. You can comment out the filtering condition, if you do not need it.

Get uploaded object names

If you want to know the name of the uploaded object, you can use plupload to call the FileUploaded event, as follows:

```
FileUploaded: function(up, file, info) {
            if (info.status == 200)

                document.getElementById(file.id).getElementsByTagName
('b')[0].innerHTML = 'upload to oss success, object name:' +
get_uploaded_object_name(file.name);

            else

                document.getElementById(file.id).getElementsByTagName
('b')[0].innerHTML = info.response;
```

You can use the following functions to get the names of the objects uploaded to OSS. The file.name property records the names of the uploaded local objects.

```
get_uploaded_object_name(file.name)
```

Upload signatures

JavaScript can get the policyBase64, accessid, and signature variables from the backend. The following is the core code for getting the three variables:

```
phpUrl = './php/get.php'
        xmlhttp.open( "GET", phpUrl, false );
        xmlhttp.send( null );
        var obj = eval ("(" + xmlhttp.responseText+ ")");
        host = obj['host']
        policyBase64 = obj['policy']
        accessid = obj['accessid']
        signature = obj['signature']
        expire = parseInt(obj['expire'])
        key = obj['dir']
```

Parse xmlhttp.responseText (the following only serves as an example. The actual format may vary, but the values of signature, accessid, and policy must exist).

```
{"accessid":"6MKOqxGiGU4AUk44",
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
```

```
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDoyMzoyM1oiLCJjxb25kaXR
pb25zIjpbWyJjcb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4NTc2MDAwXSxbInN0YXJ0
cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8iXV19",
"signature":"I2u57FWjTKqX/AE6doIdyff151E=",
"expire":1446726203,"dir":"user-dir/"}
```

- accessid: It is the Accessid of the user request. However, disclosing Accessid does not impact data security.
- host: The domain name to which the user wants to send an upload request.
- policy: A policy for uploading user forms. It is a Base64-encoded string.
- signature: A signature string for the policy variable.
- expire: It is the expiration time of the current upload policy. This variable is not sent to OSS, because it is already indicated in the policy.

Parse policy. The decoded content of the policy is as follows:

```
{"expiration":"2015-11-05T20:23:23Z",
"conditions":[["content-length-range",0,1048576000],
["starts-with","$key","user-dir/"]]
```

There is a key point here, And policytext specifies the final time that the policy upload fails. The key content of the PolicyText specifies the final expiration time of this policy. Before its expiry, this policy may be used to upload objects. Therefore, it is not necessary to obtain a signature from the backend for each upload.

Here, we use the following designs: For initial uploads, a signature is obtained for each object upload. For subsequent uploads, the current time is compared with the signature time to see whether the signature has expired. If the signature expires, a new signature is obtained. If the signature has not expired, the same signature is used. The expired variable is used here. The core code is as follows:

```
now = timestamp = Date.parse(new Date()) / 1000;
[color=#000000]//This determines whether the time specified by the
 expire variable is earlier than the current time. If so, a new
signature is obtained. 3s is the buffer duration.[/color]
    if (expire < now + 3)


    phpUrl = './php/get.php'
    xmlhttp.open( "GET", phpUrl, false );
    xmlhttp.send( null );


return .
```

We see that starts-with has been added to the policy content. This indicates the name of the object to be uploaded must start with the user-dir (this string can be customized).

This setting is added because, in many scenarios, one bucket is used for one app and contains the data of different users. To prevent the data from being overwritten, a specific prefix is added to the objects uploaded by a specific user to OSS. However, an issue occurs. Once the users obtains this policy, they can modify the upload prefix before the expiration time to upload objects to another user's directory. To resolve this issue, you can set the application server to specify the prefix of the uploaded objects by a specific user at the time of upload. In this case, no one can upload objects with another user's prefix even after obtaining the policy. This guarantees data security.

Summary

In the sample mentioned in this document, the webpage end requests the signature from the server end during uploads from the webpage end, and then objects are uploaded directly, with no pressure on the server end. This approach is safe and reliable. However in this sample, the backend program is not immediately aware of the number or identity of objects uploaded. You can use upload callback to see which objects were uploaded. This sample cannot implement multipart and breakpoint.

Related documents

- *Basic concepts*
- *Set Cross-Origin Resource Sharing (CORS)*
- *Overview of direct transfer on Web client*
- *Javascript client signature pass-through*
- *Advanced article: application server PHP returns signatures and uses upload callbacks*
- *Mobile Application-side direct transmission practice*

2.3 Directly add a signature on the server, transfer the file, and set upload callback

Background

See *Overview of direct transfer on Web client* for the background information.

The usage of *Direct transfer after adding a signature on the server* solution experiences a few issues. Once the user uploads data, the application server has to be updated with the files user uploads, the file names, image size (if any images are uploaded), and so on. Hence, the upload callback function is developed for OSS.

User request logic

1. The user obtains the upload policy and callback settings from the application server.
2. The application server returns the upload policy and callback settings.
3. The user sends a file upload request directly to OSS.
4. Once the file data is uploaded and before a response is sent by OSS to the user, OSS sends a request to the user's server based on the user's callback settings.
5. If the server returns `success`, OSS returns `success` to the user. If the server returns `failed `, OSS returns `failed` to the user. This makes sure the application server is be notified of all images that the user has successfully uploaded.
6. The application server returns information to OSS.
7. OSS returns the information returned by the application server to the user.

In brief, the user needs to upload a file to the OSS server. And, it is assumed that the user's application server is notified once the upload is completed. In this case, a callback function is required to be set to update user's application server. Due to this, OSS starts the upload once it receives user's upload request. It does not return the result to the user directly after uploading, but notifies the user's application server first with a system-generated message such as "I completed uploading"; then, the application server notifies OSS by sending "I got it. Please pass on the information to my owner" message. After sending these notifications, OSS transfers the result to the user.

Example

Sample user's Computer Browser test: *Click here to experience the upload callback example*

Use your phone to test if the upload is valid. You can use your phone (WeChat, QQ, mobile browser, etc) scan the two-dimensional code to try it (this is not an advertisement, but a two-dimensional code on the above-mentioned web site, in order to let everyone see this implementation can run perfectly on the mobile phone).

Download code

Click *here* to download the code.

The example adopts a backend signature and uses PHP language.

- Click *here* for the example of a backend signature using Java language.
- Click *here* for the example of a backend signature using Go language.
- Click *here* for the example of a backend signature using Python language.

Usage of other languages:

1. Download the corresponding language example.
2. Modify the example code, for example, set the listening port, and then start running.
3. At upload.js in `oss-h5-upload-js-php-callback.zip`, change the variable severUrl to the address configured at step 2. For example, severUrl = `http://1.2.3.4:8080` or serverUrl= `http://abc.com/post/`.

Quick start guide

Follow the steps to upload a file to OSS through the Webpage, and OSS sends a callback notification to the application server set by the user.

1. Set your own id, key, and bucket.

   Setting method: Modify `php/get.php`, and set the variable $id to AccessKeyId, $key to AccessKeySecret, and $host to bucket+endpoint.

   ```
   $id= 'xxxxxx';
    $key= 'xxxxx';
    $host = 'http://post-test.oss-cn-hangzhou.aliyuncs.com
   ```

2. To guarantee browsing security, CORS must be set for bucket. See the following content.
3. Set your own callback URL. It is also known as your own callback server address. For example, `http://abc.com/test.html` (can be accessed through public network). OSS sends the file uploading information to the application server through the callback URL (`http://abc.com/test.html`) set by you after the file is uploaded. Setting method: Modify php/get.php (for this callback server code instance, see the following content).

   ```
   $ Callbackurl = "maid ";
   ```

For more information such as uploading signature and setting a random file name, *click here for uploading details*.

The core logic is analyzed in the following content.

Core code analysis

The following content is to be added to the code:

```
new_multipart_params = {
     'key' : key + '${filename}',
     'policy': policyBase64,
     'OSSAccessKeyId': accessid,
     'success_action_status' : '200', //Instructs the server to return
 200. Otherwise, the server returns 204 by default.
     'callback':   callbackbody,
     'signature': signature,
```

The preceding callbackbody is returned by the PHP server. In this example, the following content is obtained by running the PHP scripts on the backend:

```
{"accessid":"6MKOqxGiGU4AUk44",
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDo1MjoyOVoiLCJjdb25kaXR
pb25zIjpbWyJjdb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4NTc2MDAwXSxbInN0YXJ0
cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8iXV19",
"signature":"VsxOcOudxDbtNSvz93CLaXPz+4s=",
"expire":1446727949,
"callback":"eyJjYWxsYmFja1VybCI6Imh0dHA6Ly9vc3MtZGVtby5hbGl5dW
5jcy5jdb206MjM0NTAiLCJjYWxsYmFja0hvc3QiOiJvc3MtZGVtby5hbGl5dW5jcy5jdb2
0iLCJjYWxsYmFja0JvZHkiOiJmaWxlbmFtZT0ke29iamVjdH0mc2l6ZT0ke3NpemV9Jm1p
bWVUeXBlPSR7bWltZVR5cGV9JmhlaWdodD0ke2ltYWdlSW5mby5oZWlnaHR9JndpZHRoPS
R7aW1hZ2VJdbmZvLndpZHRofSIsImNhbGxiYWNrQm9keVR5cGUiOiJhcHBsaWNhdGlvbi9
4LXd3dy1mb3JtLXVybGVuY29kZWQifQ==","dir":"user-dirs/"}
```

The preceding callbackbody is the Base64 encoded callback content in the returned results.

The decoded content is as follows:

```
{"callbackUrl":"http://oss-demo.aliyuncs.com:23450",
"callbackHost":"oss-demo.aliyuncs.com",
"callbackBody":"filename=${object}&size=${size}&mimeType=${mimeType}&
height=${imageInfo.height}&width=${imageInfo.width}",
"callbackBodyType":"application/x-www-form-urlencoded"}
```

Content analysis:

- callbackUrl: Specifies the URL request sent by OSS to this host.
- callbackHost: Specifies the Host header to be included in the request header when this request is sent by the OSS.
- callbackBody: Specifies the content sent to the application server upon OSS request. This can include a file name, size of the file, type, and image and its size (if any).
- callbackBodyType: Specifies the Content-Type requested to be sent.

Callback application server

Step 4 and 5 is important in the user's request logic. When OSS interacts with the application server. The following are a few questions explained with answers.

- Question: If I am a developer, how can I confirm that the request was sent from OSS?

  Answer: When OSS sends a request, it constructs a signature with the application server. Both use signatures to guarantee security.
- Question: How is this signature constructed? Is there any sample code?

Answer: Yes. The preceding example shows the sample code of the application server callback: `http://oss-demo.aliyuncs.com:23450` (only supports Linux now).

The preceding code runs as follows: *callback_app_server.py.zip*

Running solution: Directly run the file `python callback_app_server.py` under the Linux system. The program automatically implements a simple http server. To run this program, you may need to install the system environment on which the RSA depends.

- Question: Why the callback request received by my application server does not have an Authorization header?

  Answer: Some Web servers resolve the Authorization header automatically, for example, apache2. Therefore, it is set not to resolve this header. Using apache2 as an example, the specific setting method is as follows:

  1. Start the rewrite module, and run the command: `a2enmod rewrite`.
  2. Modify the configuration file `/etc/apache2/apache2.conf` (it varies with the installation path of apache2). Set Allow Override to All, and then add the following content:
     - RewriteEngine on
     - RewriteRule .* - [env=HTTP_AUTHORIZATION:%{HTTP:Authorization},last]

The sample program demonstrates how to check the signature received by the application server . You must add the code for parsing the format of the callback content received by the application server.

Callback application server versions in other languages

- Java version:
  - Download address: *click here*.
  - Running method: Extract the archive and run `java -jar oss-callback-server-demo.jar 9000` (9000 is the port number and can be changed as required).

    Note:
    Note: This jar runs on java 1.7. If any issue occurs, you may make changes based on the provided code. This is a maven project.
- PHP version:
  - Download address: *click here*.

  - Running method: Deploy a program to an Apache environment. Due to the characteristics of PHP language, retrieving headers depends on the environment. You can make changes to the example based on your own environment.
- Python version:
  - Download address: *click here*.
  - Running method: Extract the archive and directly run python callback_app_server.py. The program implements a simple HTTP server. To run this program, you may be required to install the system environment on which the RSA depends.
- Ruby version:
  - Download address: *click here*.
  - Running method: ruby aliyun_oss_callback_server.rb.

Summary

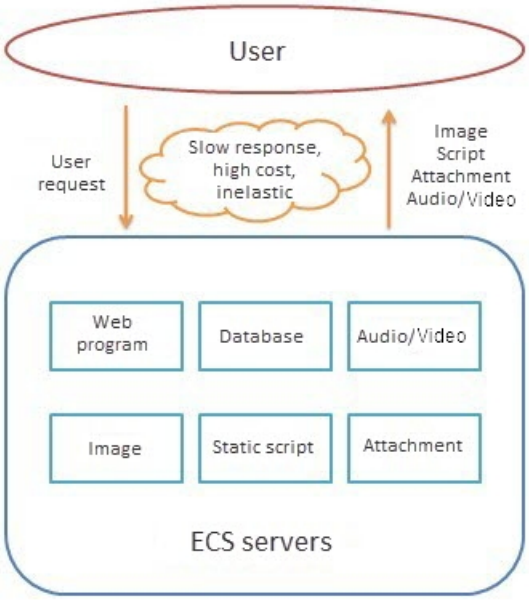
- Example 1: Describes how to add a signature directly on the JavaScript client and upload a file in the form to OSS directly. *oss-h5-upload-js-direct.tar.gz*
- Example 2: Describes how to obtain a signature from the backend using the PHP script and then upload the file in a form to OSS directly. *oss-h5-upload-js-php.tar.gz*
- Example 3: Describes how to obtain a signature from the backend using the PHP script, and perform callback after uploading, and then, upload the form directly to OSS. Consequently, OSS calls back the application server and returns the result to the user. *oss-h5-upload-js-php-callback.tar.gz*

3 Bucket management

3.1 CDN-based OSS acceleration

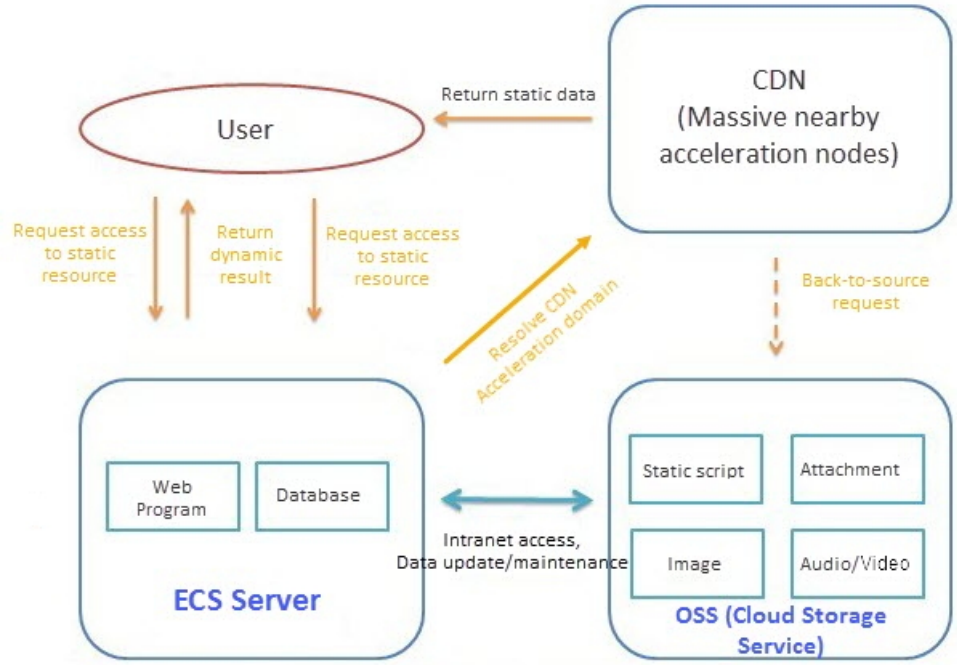
Background

Structure of traditional products without static-dynamic separation (however, performance encounters a bottleneck as traffic increases).



Traditional Web Architecture

Product structure implementing static-dynamic separation (a flexible structure supports massive user traffic).



Web Architecture Implementing Static/Dynamic Separation

Scenarios

- Massive access to static files, high server loads, and I/O problems, causing slow user access.
- Large volumes of static files and insufficient storage space.
- Massive access to static files distributed across various regions.
- Fast and concurrent download of mobile update packages in large volumes within a certain time period.

Structural description

As the storage source for massive file volumes, OSS stores static images, video files, downloaded packages, app update packages, and other resources. As OSS is the origin site for CDN, OSS files can be obtained through CDN accelerated delivery by accessing nearby CDN nodes.

Structural advantages:

- Reduces the load on Web servers, and directs the access to all static files to CDN.
- Provides the lowest storage fees. OSS storage fee is only half that of ECS disks.
- Provides massive storage capacity, without the need to consider structural upgrade.

- Minimizes traffic fees. Apart from a small amount of additional origin retrieval traffic, the majority of the traffic is CDN traffic. And it's cost is lesser than the Internet traffic for direct access from OSS.

Case study

A common website is used as an example. A recently established website www.acar.com, is an automotive news and discussion website which is built on PHP. The main site stores 10 GB of image resources and some JS files. An ECS instance is purchased to store all program codes and MySQL database is installed on the ECS instance.

As access traffic continues to grow, many users report that the website access speed experience s a slow down such as loading of pictures and website response consumes time. The website's technical staff notices that users are uploading an increasing number of images and the total size will soon exceed 1 TB.

The technical staff can use OSS and CDN to optimize the website structure to achieve static-dynamic separation shown in the preceding figure. It enhances user experience, and keep their costs at a manageable level.

The specific solution and procedures are as follows:

1. Sort out the website program code on the ECS instance by storing dynamic programs and static resources in different directories for better management.
   - Create a directory named *`Images`* for storing the website's high-definition images.
   - Create a directory named *`Javascript`* for storing all JS scripts.
   - Create a directory named *`Attachment`* for storing all images and attachments uploaded by users.
2. Create a bucket.

   Select the bucket region based on your ECS region and select the permission option **Public Read**. You have to make sure that the bucket name corresponds to one of the directories created on the ECS instance, for example, acar-image-bucket. For more information, see *Create a bucket*.
3. Enter `image.acar.com` as the domain for the HD videos and images on your website. For more information, see *Manage a domain name*.
4. Upload files to verify the CDN effect.

a. Upload all image files in the Images directory created in Step 1 on the ECS instance to acar-image-bucket. For more information, see *Upload objects*. You can use an OSS client to complete the upload process conveniently.
b. Get the CDN address for this file. The address format is your CDN domain+'/'+'file name'. For more information, see *Get object URL*.
c. Upload image files one by one.

5. Repeat the preceding steps to upload the files in the other two directories, and create the CDN-based OSS buckets **acar-js-bucket** and **acar-csimages-bucket** .
6. In the ECS system, find the access code for the static files and replace the access URL with the CDN domain. Users access static files on your website in OSS+CDN mode without occupying your ECS resources.

> **Note:**
> If you want to automatically synchronize user-uploaded files to **acar-csimages-bucket**, see the OSS SDKs and the PutObject section of the API documentation. This provides information on how to perform automatic upload at the code level.

CDN automatic refresh

If you use Alibaba Cloud CDN with a bound CDN domain that points back to an OSS source, you can use OSS's CDN cache automatic refresh function. OSS automatically refreshes CDN when the data is overwritten (for example, when an existing file is overwritten or deleted). A origin retrieval operation is performed to obtain the overwritten file from OSS, so you do not need to explicitly call the CDN refresh interface. The URL refresh rules are as follow:

```
CDN domain + / + Object
```

For example, if the uploaded file `test.jpg` is overwritten in the bucket bound to the CDN domain `image.acar.com`, OSS refreshes the `image.acar.com/test.jpg` URL. The time required by the refreshed URL to take effect is determined by CDN's guaranteed refresh time, which is typically less than 10 minutes.

To activate CDN-based OSS acceleration, enable the **Refresh CDN cachefunction**on the bucket **Domain Management** page.

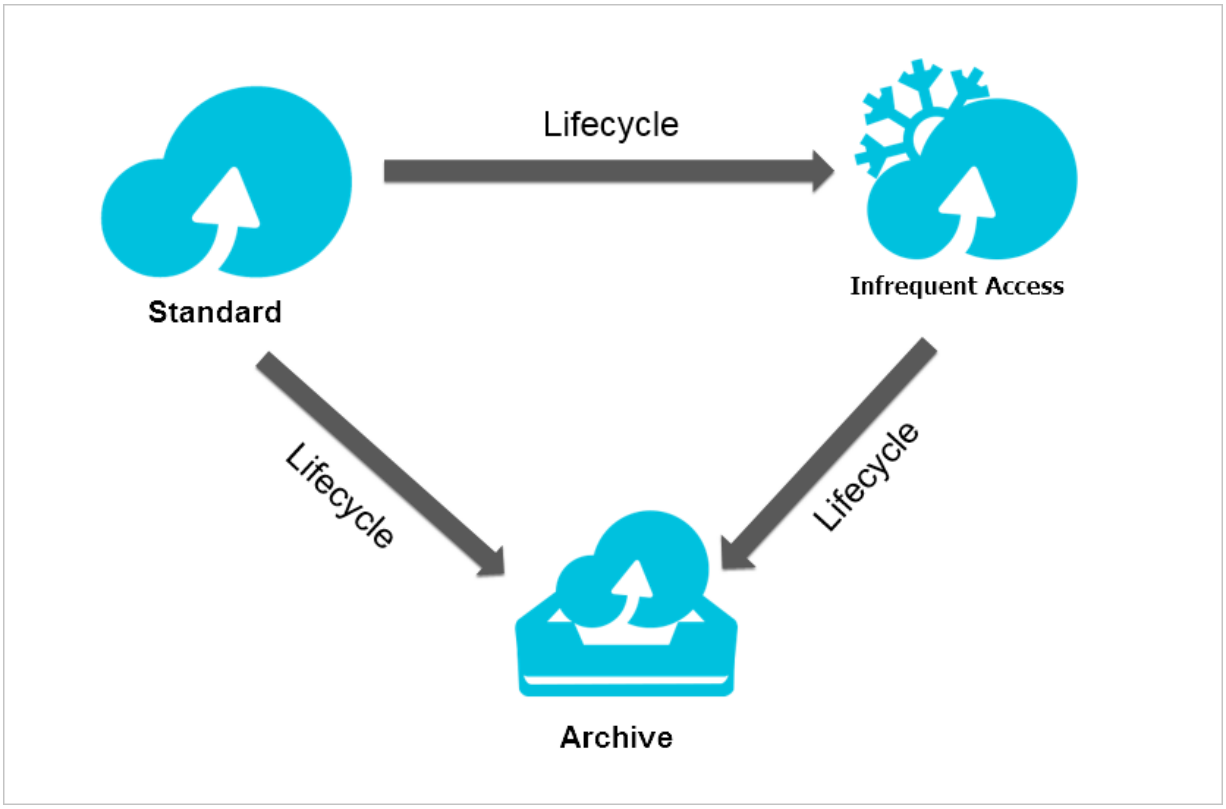
3.2 Storage class conversion

Lifecycle Object Transition

OSS supports three *storage classes*: Standard, Infrequent Access, and Archive.

The Object Transition mechanism is now available in OSS Lifecycle Management function in all regions across China. The following storage classes are supported for automatic conversion:

- Standard -> Infrequent Access
- Standard -> Archive
- Infrequent Access -> Archive

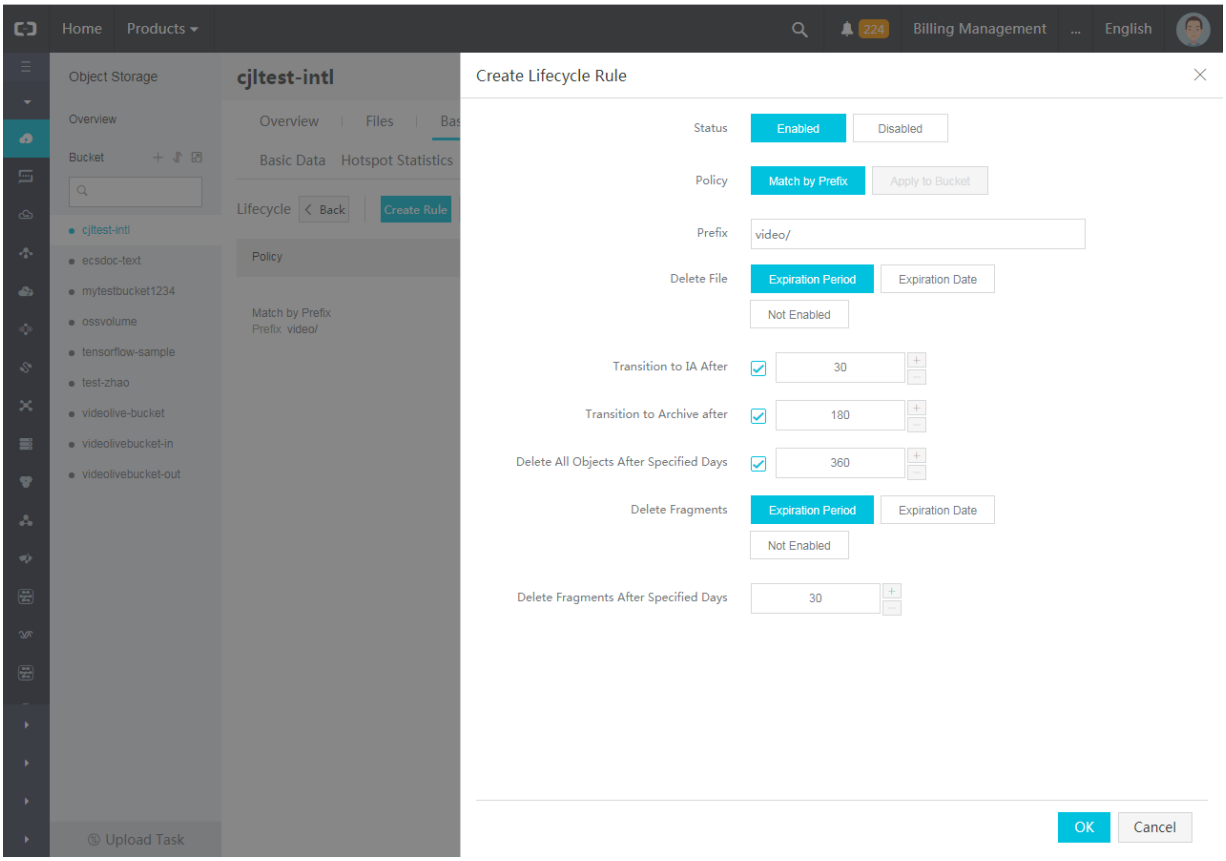


Examples

You can configure lifecycle policies for objects with a given prefix in one bucket as follows:

- They are converted to Infrequent Access class after being stored for 30 days.
- They are converted to Archive class after being stored for 180 days.
- They are deleted automatically after being stored for 360 days.

You can complete the configuration of the preceding lifecycle policies in the console. For more information, see *Set lifecycle*.



Note:

If the following three parameters are configured:

`Transition to IA After`, **`Transition to Archive After`**, and **`Delete All Objects After Specified Days`**, then the number of days set for each parameter must meet the following criteria:

Days for converting to Infrequent Access < Days for converting to Archive < Specified days for deleting

Notes

After the Object type conversion, the storage cost is calculated based on the unit price of converted storage class.

Notes for Infrequent Access and Archive storage types:

- Minimum billable size:

  Objects smaller than 128 KB are charged as 128 KB.
- Minimum storage period:

The stored data is required to be saved for at least 30 days. Charges will be incurred if you delete files that are stored for less than 30 days.

- Restore time of Archive type:

  It takes one minute for Archive type Object to restore the data to a readable state. If real-time read is required in the business scenario, we recommend that you convert the file to the Infrequent Access storage class instead of Archive class. Otherwise, after converting the file to the Archive class, the data cannot be read in real time.

- Data access charges:

  Both Infrequent Access and Archive classes are required to pay data access charges as a separate charge item to outbound traffic. If the average access frequency per Object is higher than once per month, you are not advised to convert the data to Infrequent Access or Archive class.

Storage classes conversion in other ways

For conversions from Archive type to Standard class or Infrequent Access class, or from Infrequent Access class to Standard class, you can read the Object and rewrite it to the Bucket of corresponding storage class. The default storage class of Object is determined by the Bucket.

For example, for the conversion of Infrequent Access Object in the Bucket of Standard type to Standard Object, you can read and rewrite the Object. Based on the type of the Bucket, the newly-written Object is of Standard storage class.

For the Object that has been converted to Archive class, you can only read it after performing Restore operation and restore it to a readable state.

For mor einformation, see *Create and use the Archive bucket*.

3.3 Cross-origin resource sharing (CORS)

Same-origin policy

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

However, cross-origin access is a commonly used on a day-to-day basis. For example, OSS is used at the backend for the website www.a.com. The JavaScript-based upload function

is provided on the webpage. However, requests on the webpage are only sent to www.a.com , whereas all requests sent to other websites are rejected by the browser. As a result, user-uploaded data must be relayed to other sites through www.a.com . If cross-origin access is configured, data can be uploaded directly to OSS instead of relaying it through www.a.com.

CORS overview

CORS is a standard cross-origin solution provided by HTML5. For the specific CORS rules, see *W3C CORS Norms*.

CORS is a set of control policies followed by the browsers, which use HTTP headers for interaction. When identifying a request initiated as a cross-origin request, a browser adds the Origin header to the HTTP request and sends the request to the server. In the preceding example, the Origin header is www.a.com. After receiving the request, the server determines based on certain rules whether to permit the request. If the request is permitted, the server attaches the Access-Control-Allow-Origin header to the response. The header contains www.a.com, indicating that cross-origin access is allowed. In case, server permits all cross-origin requests, set the Access-Control-Allow-Origin header to *. The browser determines whether the cross-origin request is successful based on whether the corresponding header is returned. If the corresponding header is not attached, the browser blocks the request.

The preceding content is a simple scenario. CORS norms classify requests into two types: simple requests and precheck requests. Precheck is a protection mechanism that prevents unauthorized requests from modifying resources. Before sending the actual request, the browser sends an OPTIONS HTTP request to determine whether the server permits the cross-origin request. If the request is not permitted, the browser rejects the actual request.

No precheck request is required only if both of the following conditions are met:

- The request method is one of the following:
  - GET
  - HEAD
  - POST
- All headers are in the following lists:
  - Cache-Control
  - Content-Language
  - Content-Type
  - Expires

- Last-Modified
- Pragma

Precheck requests provide information about the subsequent request to the server, that includes:

- Origin: Request origin information.
- Access-Control-Request-Method: Type of the subsequent request, for example, POST or GET.
- Access-Control-Request-Headers: List of headers explicitly set and included in the subsequent request.

After receiving the precheck request, the server determines whether to permit the cross-origin request based on the attached information. The return information is also sent using the following headers:

- Access-Control-Allow-Origin: list of permitted origins for cross-origin requests.
- Access-Control-Allow-Methods: List of permitted cross-origin request methods.
- Access-Control-Allow-Headers: List of permitted cross-origin request headers.
- Access-Control-Expose-Headers: List of headers permitted to be exposed to JavaScript code.
- Access-Control-Max-Age: Maximum browser cache time in seconds.

Based on the returned information, the browser determines whether to send the actual request. If none of these headers is received, the browser rejects the subsequent request.

Note:

The preceding actions are performed automatically by the browser, and you can ignore the details. If the server is correctly configured, the process is the same for non-cross-origin requests.

Scenarios

Access permission control applies to browsers rather than servers, CORS is only applicable in scenarios where a browser is used. Hence, you do not need to worry about cross-origin issues when using other clients.

Applications that use CORS primarily, use Ajax in a browser to directly access OSS, instead of requiring traffic to be redirected through application servers. This applies to the upload and download processes. For websites powered by both OSS and Ajax technology, CORS is recommended for direct communication with OSS.

OSS support for CORS

OSS supports CORS rule configuration for permitting or rejecting corresponding cross-origin requests as required. CORS rules are configured at the bucket level. For more information, see *PutBucketCORS*.

Whether a CORS request is permitted is independent of OSS identity verification. That is, the OSS CORS rules are only used to determine whether to attach relevant CORS headers. Whether the request is blocked is only determined by the browser.

When using cross-origin requests, make sure the browser's cache function is enabled. For example, the same cross-origin resource is requested respectively by two webpages in the same browser (originated from www.a.com and www.b.com) at the same time. If the request of www.a.com is received by the server in the first place, the server returns the resource with the Access-Control-Allow-Origin header "www.a.com". When www.b.com initiates its request, the browser returns its previous cached request. As the header content does not match the CORS request, the subsequent request fails.

Note:
Currently, all OSS object-related interfaces provide CORS verification. In addition, multipart interfaces fully support CORS verification.

Cross-origin GET request example

In this example, Ajax is used to retrieve data from OSS. For simplified description, all used buckets are public. The CORS configuration for accessing a private bucket is the same and only requires a signature to be attached to the request.
Getting started

Create a bucket. For example, create the bucket oss-cors-test with the access right set to public-read. Then create the text file named test.txt, and upload it to the bucket.

The test.txt access address is *http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/test.txt*.

Note:
Replace the following address with your test address.

Use curl to directly access the file:

```
curl http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/test.txt
```

```
just for test
```

The file can be accessed properly.

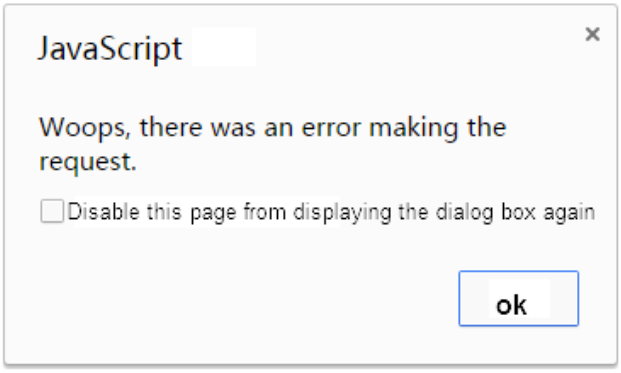
The following code describes how to directly access this website using Ajax. It is the simplest HTML code for access. You can copy the following code, save it as a local HTML file, and open it through your browser. Because no custom headers and hence are included, this request does not require a precheck.

```
<! DOCTYPE html>
<html>
<head>
<script type="text/javascript" src="./functions.js"></script>
</head>
<body>
<script type="text/javascript">
// Create the XHR object.
function createCORSRequest(method, url) {
  var xhr = new XMLHttpRequest();
  if ("withCredentials" in xhr) {
    // XHR for Chrome/Firefox/Opera/Safari.
    xhr.open(method, url, true);
  } else if (typeof XDomainRequest ! = "undefined") {
    // XDomainRequest for IE.
    xhr = new XDomainRequest();
    xhr.open(method, url);
  } else {
    // CORS not supported.
    xhr = null;

  return xhr;

// Make the actual CORS request.
function makeCorsRequest() {
  // All HTML5 Rocks properties support CORS.
  var url = 'http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/test.
txt';
  var xhr = createCORSRequest('GET', url);
  if (! xhr) {
    alert('CORS not supported');
    return;

  // Response handlers.
  xhr.onload = function() {
    var text = xhr.responseText;
    var title = text;
    alert('Response from CORS request to ' + url + ': ' + title);

  xhr.onerror = function() {
    alert('Woops, there was an error making the request.');

  xhr.send();

</script>
<p align="center" style="font-size: 20px;">
<A href = "#" onclick = "Fig (); Return false; "> run sample </a>
</p>
</body>
```

```
</html>
```

After opening the file, click the link (Chrome is used in this example). Check that the link cannot be accessed.

Run Sample



Use Chrome developer tools to identify the cause of the error.

The error is due to the fact that no Access-Control-Allow-Origin header is found. This is because the server is not configured with CORS.

Return to the header interface to check that the browser sends a request with an Origin header. Hence, the request is a cross-origin request. On Chrome, the origin is null because the file is a local file.

Configure Bucket CORS settings

Once the problem is located, you can configure CORS settings for the bucket to make sure successful execution of the preceding operation attempt. To facilitate understanding, the following describes how to configure CORS settings on the console. We recommend that CORS be configured on the console if CORS settings are not complex. CORS settings are composed of individual rules.

When the system looks for matches, each rule is checked as a match starting with the first rule. The first matched rule applies. The following shows how to add a rule with the loosest configurat ion:

This indicates that access is permitted to all origins, all request types, and all headers, and the maximum cache time is 1s.

Once the configuration is completed, perform the test again. The result is as follows:

Access requests can be sent properly.

If you are required to troubleshoot cross-origin access problems, you can configure CORS as shown in the preceding figure. This configuration permits all cross-origin requests. If an error occurs under this configuration, the error is not related to CORS.

Besides the loosest configuration, a more refined control mechanism can be configured for targeted control. The following shows the strictest configuration for a successful match:

In most cases, we recommend that you use the strictest configuration applicable in their use scenarios to guarantee maximum security at minimal configuration.

Use cross-origin requests for POST upload

The following provides a more complex example where a POST request with a signature is used, and the browser must send a precheck request.

PostObjectSample

Note:
After downloading the preceding code, modify all the following sections to meet your requirements. Then run it on your server.

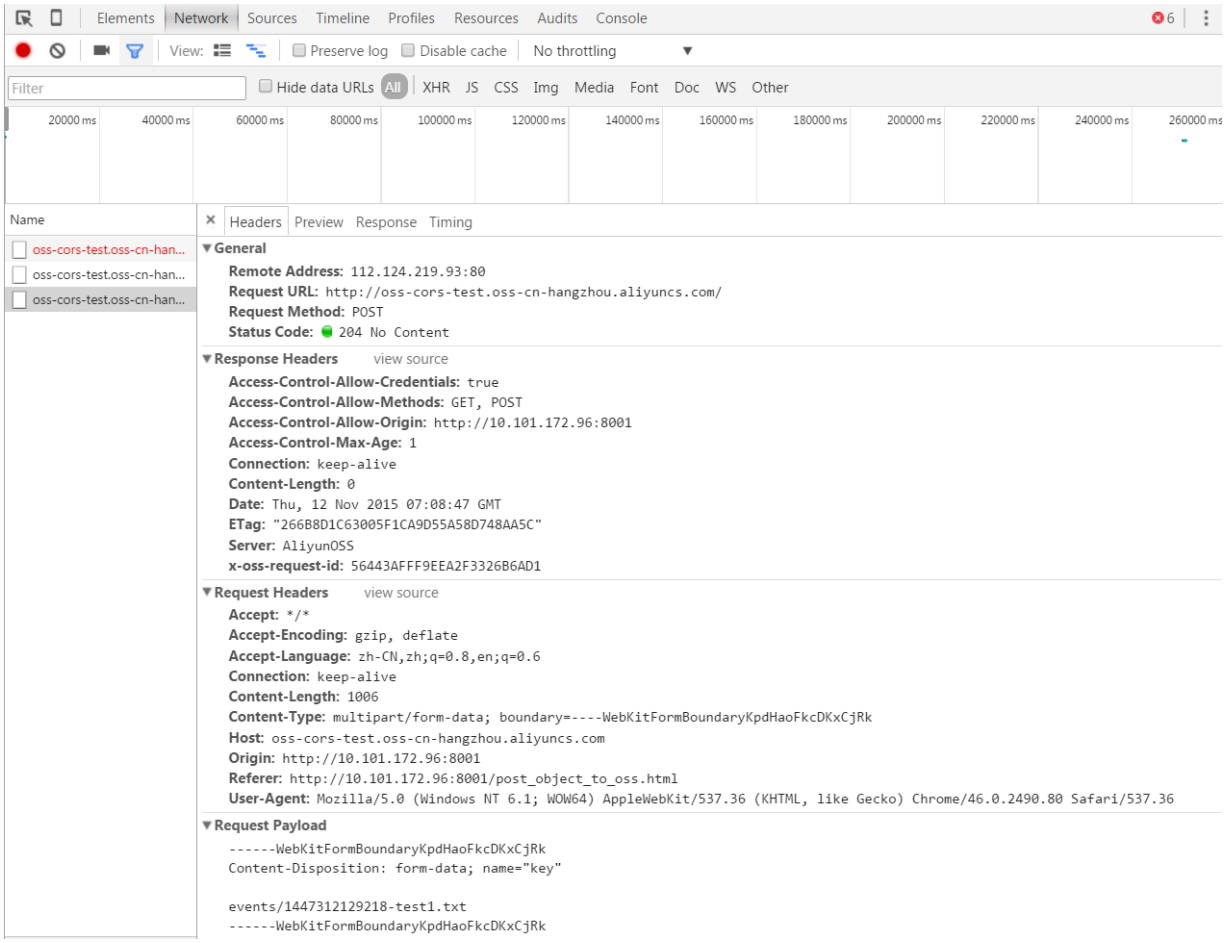
The following describes how to use the bucket oss-cors-test for testing. Before testing, delete all CORS rules to restore the configuration to its initial state.

Access this webpage and select a file to upload.

Start the developer tools, and you can view the following content. Based on the previous GET example, it is easy to find the same cross-origin error. Different from the GET request, the request requires a precheck. As shown in the following figure, the operation fails because the OPTIONS response does not have CORS headers.

Modify the CORS configuration accordingly.

You can perform the operation again to get a successful result. The console displays the newly uploaded file.



Test content:

```
$curl http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/events/
1447312129218-test1.txt
post object test
```

CORS configuration caveats

CORS configuration items include:

- Source: Provide the complete domain information during configuration, for example, `http://10.101.172.96:8001` as shown in the preceding figure.

  Do not omit the protocol name, for example, http. Include the port number if the default one has been changed. If you are not sure, use the browser's debugging function to view the Origin header. This field supports the wildcard *, but only one such symbol can be used. You can perform configuration based on your needs.
- Method: Select the allowed methods based on your requirements.
- Allow Header: Indicates the list of allowed headers. To avoid header omission, we recommend that you set this field to * unless otherwise specified. The header is not case-sensitive.
- Expose Header: Indicates the list of headers exposed to the browser. Wildcards cannot be used. The specific configuration must be selected according to your application. Expose only required headers, for example, ETag headers. If you do not need to expose this informatio n, you can leave this field blank. You can specify headers individually. This field is not case-sensitive.
- Cache Time: In normal cases, you can set a relatively large value, for example, 60s.

The CORS configuration method sets individual rules for each origin that may access the service . If possible, do not include multiple origins in a single rule, and avoid overlap or conflict among multiple rules. For other permissions, you only need to grant the required permissions.

Troubleshooting advice

It is easy to mix up other errors with CORS errors when similar programs are debugged.

For example, when an access request is rejected because of any incorrect signature, the return result may not contain CORS header information because permission verification precedes

CORS verification. In this case, some browsers directly report a CORS error, but the actual CORS configuration on the server is correct. The following two methods can be used to address the preceding problem:

- View the HTTP request's return value. Because CORS verification is an independent process that does not affect core processes, a return value such as 403 is not produced by CORS. You must first rule out the program-related causes. If a precheck request is sent previously, you can view the precheck request results. If the correct CORS headers are returned, the actual request is permitted by the server. Therefore, the error can only be caused by another component.
- Set the server's CORS configuration to the loosest setup shown in the preceding example. Use wildcards to permit all origins and request types. Then re-verify the configuration. If the verification still fails, it is possible that other type of errors have occured.

3.4 Anti-leech

Background

For example, A is the webmaster of a website. Webpages on the website contain links to images and audio/video files. These static resources are stored on *Alibaba Cloud OSS*. For example, A may save an image file on OSS with the URL `http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`.

For external URLs of OSS resources, see *OSS access*. These URLs (unsigned) require the public-read bucket permission.

B is the webmaster of another website. Without A's permission, B places image resources of A's website on B's webpage, so as to steal the storage space and traffic of A. Users who view content on B's website do not know or care where the images on the site are sourced. However, OSS bills fee on the basis of usage. A has to pay the resource usage fee without gaining much benefits.

This document is applicable in the scenario where OSS resources are used as external links on webpages.

Implementation method

Currently, OSS provides two anti-leech protection methods:

- Configuring Referer: This method can be used on the console or through SDKs, and is suitable for the users who prefer not to write code and users used to programming.

- Using signed URLs: This method is suitable for users used to programming.

This document presents two examples:

- Configuring Referer anti-leech protection on the console
- and using the PHP SDK to dynamically generate signed URLs for anti-leech protection.

Configuring Referer

The following content describes what a Referer is and how OSS uses Referers to provide anti-leech protection.

- What is a Referer?

  A Referer is part of an HTTP header. When a browser sends a request to a web server, the request includes a Referer to notify the server of the webpage link where the request is initiated from. Based on the previous example, assume that user B's website is called userdomain-steal, and B wants to leech user A's image link `http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`. The domain name of A's website is `userdomain`.

  Assume that the webpage of the leeching website userdomain-steal is as follows:

```
<html>
    <p>This is a test</p>
    <img src="http://referer-test.oss-cn-hangzhou.aliyuncs.com/
aliyun-logo.png" />
</html>
```

  Assume that the webpage of the origin site userdomain is as follows:

```
<html>
    <p>This is my test link from OSS URL</p>
    <img src="http://referer-test.oss-cn-hangzhou.aliyuncs.com/
aliyun-logo.png" />
</html>
```

  - When an Internet user uses a browser to access B's webpage `http://userdomain-steal/index.html`, the webpage links to an image from A's website. The request is redirected from one domain name (userdomain-steal) to another domain name (oss-cn-hangzhou.aliyuncs.com). Therefore, the browser adds a Referer to the header of the HTTP request.

    The Referer in the HTTP request in the browser is set to `http://userdomain-steal/index.html`. In this document, webpage requests are displayed in Chrome's developer mode.

- When `http://userdomain/error.html` is accessed in a browser, the browser shows that the Referer is set to `http://userdomain/error.html`.
- If a file address is entered in the browser, the Referer in the request is blank.

If user A does not configure any Referer settings on OSS, the preceding three methods can be used to access user A's image links.

- OSS Referer anti-leech protection principles

As mentioned in the preceding example, when a browser requests OSS resources, the browser includes a Referer in the request in the case of a page jump. The Referer value is the URL of the previous page, but may be blank in some cases.

OSS's Referer function provides two options to deal with the preceding two situations.

- Configure whether to allow access with a blank Referer. This option cannot be configured separately, but must be used with a Referer whitelist.
- Configure a Referer whitelist.

Analysis:

- Anti-leech verification is performed only when objects are accessed using a signed URL or anonymously. When the request header contains the "Authorization" field, anti-leech verification is not performed.
- One bucket supports multiple Referer fields.
- The Referer field supports the wildcards * and ?.
- You can configure whether to allow access requests with blank Referer fields.
- If the whitelist is blank, the system does not check whether the Referer field is blank (otherwise, all requests get rejected, including requests with blank Referer fields and requests with non-blank Referer fields which cannot be found in the Referer whitelist).
- When the whitelist is not blank and the rule "Do Not Allow Blank Referer Fields" is configured, only requests with whitelisted Referers are allowed, whereas other requests (including requests with blank Referer fields) are rejected.
- When the whitelist is not blank and the rule "Allow Blank Referer Fields" is configured, requests with blank Referer fields and requests with whitelisted Referers are allowed, whereas other requests are rejected.

- Referer fields are checked under all three bucket permissions (private, public-read, and public-read-write).

Wildcard details:

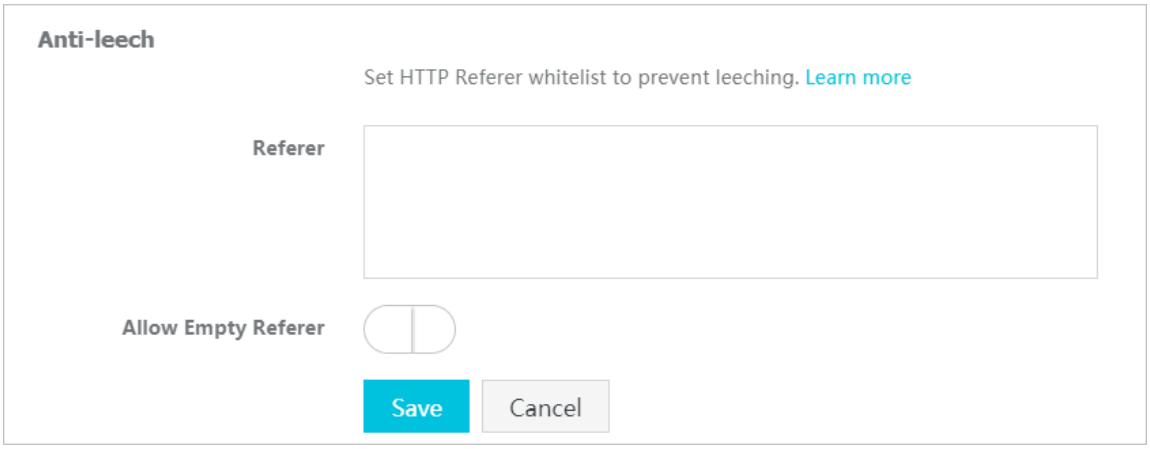
- Asterisk (*): can be used to represent 0 or multiple characters. If you are looking for a file name that starts with “AEW”, you can enter “AEW“ to search for all types of files with the names starting with “AEW”, for example, AEWT.txt, AEWU.EXE, and AEWI.dll. If you want to narrow down the search scope, you can enter AEW.txt to search for all .txt files with names starting with AEW, such as AEWIP.txt and AEWDF.txt.
- Question mark (?): represents one character. If you enter love?, all types of files with names starting with “love” and ending with a character are displayed, such as lovey and lovei. If you want to narrow the search scope, you can enter love?.doc to search for all .doc files with names starting with “love” and ending with a character, such as lovey.doc and lovei.doc.

- Anti-leech effects of different Referer settings

The following describes the effects of Referer settings:

- Disable Allow Empty Referer,

as shown in the following figure:



Direct access: The resources are accessible even when anti-leech protection takes effect. The reason is, if the whitelist is blank, the system does not check whether the Referer field is blank. The Referer setting does not take effect when the whitelist is blank. Therefore, the Referer whitelist must be configured.

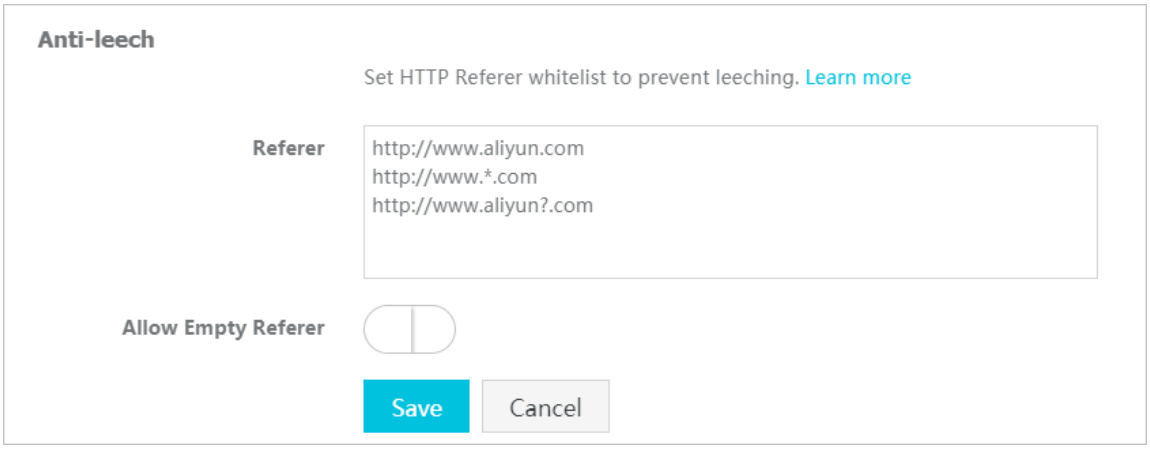
- Disable Allow Empty Referer and configure a Referer whitelist.

As shown in the preceding example, the Referer in the browser request is the URL of the current webpage. Therefore, it is necessary to know from which URL the request jumps and then specify the URL.

Referer whitelist setting rules:

- In the example, the Referer is `http://userdomain/error.html`. Therefore, the Referer whitelist can be set to `http://userdomain/error.html`. As the Referer check performed by OSS is based on prefix matching, access to other webpages such as `http://userdomain/index.html` fails. To avoid this problem, you can set the Referer whitelist set to `http://userdomain/`.
- To allow access to other domain names such as `http://img.userdomain/index.html`, add `http://*.userdomain/` to the Referer whitelist.

Both entries are configured as shown in the following figure:



After testing, the following results are obtained:

Browser input	Expectation	Result
http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png	Expectation for direct access with a blank Referer: Blank Referers are not allowed and OSS returns 403.	As expected
http://userdomain/error.html	Expectation for a request from the origin site: successful access.	As expected
http://userdomain-steal/index.html	Expectation for a request from a leeching site: OSS returns 403. Anti-leech protection is successful.	As expected

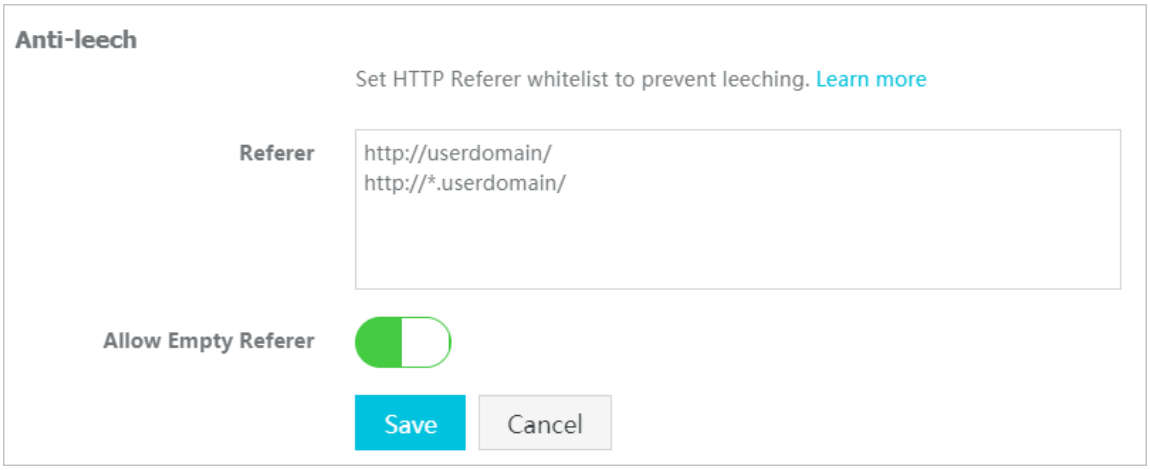
Browser input	Expectation	Result
http://img.userdomain/error.html	Expectation for a request from a third-level domain of the origin site: successful access.	As expected

Note:

- In this test, the domain names only serve as examples, and are not the same as the actual domain names you use. Be sure to differentiate them.
- If the Referer whitelist only contains `http://userdomain/`, and the browser attempts to access the resources through the simulated third-level domain name `http://img.userdomain/error.html`, the third-level domain name fails to match any of the entries in the Referer whitelist, and OSS returns 403.

- Enable Allow Empty Referer and configure a Referer whitelist

  The Referer whitelist contains `http://*.userdomain/` and `http://userdomain`, as shown in the following figure:



After testing, the following results are obtained:

Browser input	Expectation	Result
http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png	Expectation for direct access with a blank Referer: successful access	As expected
http://userdomain/error.html	Expectation for a request from the origin site: successful access	As expected

Browser input	Expectation	Result
http://userdomain-steal/index.html	Expectation for a request from a leeching site: OSS returns 403. Anti-leech protection is successful.	As expected
http://img.userdomain/error.html	Expectation for a request from a third-level domain of the origin site: successful access	As expected

- How to configure Referer on OSS

  Reference:

  — API：*Put Bucket Referer*

  — Console: *Anti-leech settings*

- Pros and cons of Referer anti-leech protection

  Referer anti-leech protection can be easily configured on the console. The main drawback of the Referer anti-leech protection is that it cannot prevent access attempts by the malicious spoofing Referers. If a leecher uses an application to simulate HTTP requests with a spoofing Referer, the Referer can bypass anti-leech protection settings. If you have higher anti-leech protection requirements, consider using signed URL anti-leech protection.

Signed URLs

For the principles and implementation methods for signed URLs, see *Authorizing third-Party download*. A signed URL is implemented as follows:

1. Set the bucket permission to private-read.
2. Generate a signature based on the expected expiration time (the time when the signed URL expires).

Specific implementation

1. Install the latest PHP code by referring to the *PHP SDK documentation*.
2. Generate a signed URL and add it to the webpage as an external link, for example:

```
<? php
 require 'vendor/autoload.php';
  #Indicates the automatic loading function provided by the latest
PHP.
 use OSS\OssClient;
 #Indicates the namespace used.
 $accessKeyId="a5etodit71tlznjt3pdx7lch";
```

```
  #Indicates the AccessKeyId, which must be replaced by the one you
use.
 $accessKeySecret="secret_key";
  #Indicates the AccessKeySecret, which must be replaced by the one
you use.
 $endpoint="oss-cn-hangzhou.aliyuncs.com";
  #Indicates the Endpoint, selected based on the region created by
the bucket. In the example, the endpoint is Hangzhou.
 $bucket = 'referer-test';
 #Indicates the bucket, which must be replaced by the one you use.
 $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);
 $object = "aliyun-logo.png";
  #Indicates the object to be signed.
 $timeout = 300;
  #Indicates the expected link expiration time. The value indicates
 that the link is valid for 300 seconds from when this line of code
starts running.
  $signedUrl = $ossClient->signUrl($bucket, $object, $timeout); #
Indicates the function used to implement the signed URL.
 $img= $signedUrl;
  #Indicates dynamically placing the signed URL in image resources
and printing it out.
 $my_html = "<html>";
 $my_html .= "<img src=\"".$img. "\" />";
 $my_html .= "<p>".$img."</p>";
 $my_html .= "</html>";
 echo $my_html;
```

3. Access the signed URL using a browser. If the browser requests the resource multiple times, different signed URLs may be displayed. This is a normal phenomenon because the signed URL changes once it expires. After expiration time the link is no longer valid. It is displayed in Unix time format, for example, Expires=1448991693. The time can be converted to the local time. In Linux, the command for converting the time is `date -d@1448991693`, you can also find a conversion tool on the Internet.

Note:

Signed URLs can be used with the Referer whitelist function.

If the expiration time of signed URLs is limited to minutes, even when a leecher spoofs a Referer, the leecher needs to obtain the signed URL and complete leeching before the signed URL expires . Compared with the Referer method, this makes leeching more difficult. Using signed URLs with the Referer whitelist function provides enhanced anti-leech protection results.

Conclusion

Best practices of OSS-based anti-leech protection:

- Use third-level domain name URLs, such as `referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`, as they are more secure than bound second-level domain names.

The third-level domain name access method provides bucket-level cleaning and isolation, enabling you to respond to a burst in leeching traffic while preventing different buckets from affecting each other, thereby increasing service availability.

- If you use custom domain names as links, bind the CNAME to a third-level domain name, with the rule bucket + endpoint. For example, your bucket is named “test” and the third-level domain name is `test.oss-cn-hangzhou.aliyuncs.com`.
- Set the strictest possible permission for the bucket. For example, set a bucket that provides Internet services to public-read or private. Do not set it to public-read-write. For bucket permission information, see *Access control*.
- Verify access sources and set a Referer whitelist based on your requirement.
- If you need a more rigorous anti-leeching solution, consider using signed URLs.
- Record access logs of the bucket, so that you can promptly discover leeching and verify the effectiveness of your anti-leeching solution. For access log information, see *Access logging configuration*.

FAQ

- I have configured anti-leech protection on the OSS Console, but the configuration does not take effect. Access to webpages is blocked, whereas access to players is not. Why? How can this problem be fixed?

  Currently, anti-leech protection fails to take effect for audio and video files. When a media player, such as Windows Media Player or Flash Player, is used to request OSS resources, a blank Referer request is sent. This causes anti-leech protection ineffective. To resolve this issue, you can see the preceding signed URL anti-leech protection method.
- What is a Referer? How is it sent? How to deal with HTTPS websites? Does anything else need to be added, like commas?

  A Referer is a request header in the HTTP protocol. It is attached to a request that involves a page jump. You must check whether the Referer in the request sent by your browser is `http://` or `https://`. In normal cases, the Referer is `http://`.
- How are signed URLs generated? Is storing the AccessKeySecret on the client secure?

  It is not recommended that the AccessKeySecret be directly stored on the client. RAM provides the *STS service* to solve this problem. Also, see*RAM and STS Guide*.
- How do I use wildcard characters (*, ?) to write `a.baidu.com` and `b.baidu.com` ?

You can use `http://*.baidu.com`. If the wildcard character represents a single character only, you can also use `http://?.baidu.com`.

- *.domain.com can match a second-level domain name, but does not match domain.com. Only adding a second entry of domain.com does not work either. What settings must be configured?

  Note that a Referer generally includes a parameter such as http. You can view the request Referer in Chrome's developer mode and then specify the corresponding Referer. As in this case, you may have forgotten to include `http://`, which is required to be `http://domain.com`.

- What must I do if anti-leech protection does not take effect?

  We recommend that you use Chrome to solve the problem. Enable the developer mode and click on the webpage to view the `Referer` value in the HTTP request. Check whether the Referer value matches the `Referer` value configured on OSS. If they do not match, set the Referer value configured on OSS to the Referer value in the HTTP request. If the problem persists, open a ticket.

3.5 Static website hosting

This document describes the process and procedure about how to build a simple static website based on OSS right from the beginning and also includes FAQs as well. The following are the key steps:

1. Apply for a domain name.
2. Activate OSS and create a bucket.
3. Activate Static Website Hosting on OSS.
4. Access OSS with custom domain names.

Static website hosting overview

You can build a simple static website page based on OSS. Once you activate this function, OSS provides a default homepage and a default 404 page. For more information, see *Static Website Hosting* in the developer guide.

Procedure

1. Apply for a domain name
2. Activate OSS and create a bucket

a. Log on to the OSS console and create a bucket named “imgleo23” in Shanghai with the endpoint `oss-cn-shanghai.aliyuncs.com`. For detailed operation, see *Create a bucket*.

b. Set the bucket permission to public-read. For detailed operation, see *Set bucket ACL*.

c. Upload the content of index.htm and error.htm. For detailed operation, see *Upload objects*.

- Body of index.html:

```
<html>
  <head>
      <title>Hello OSS! </title>
      <meta charset="utf-8">
  </head>
  <body>
      <p>Welcome to OSS Static Website Hosting.</p>
      <p>This is the homepage.</p>
  </body>
</html>
```

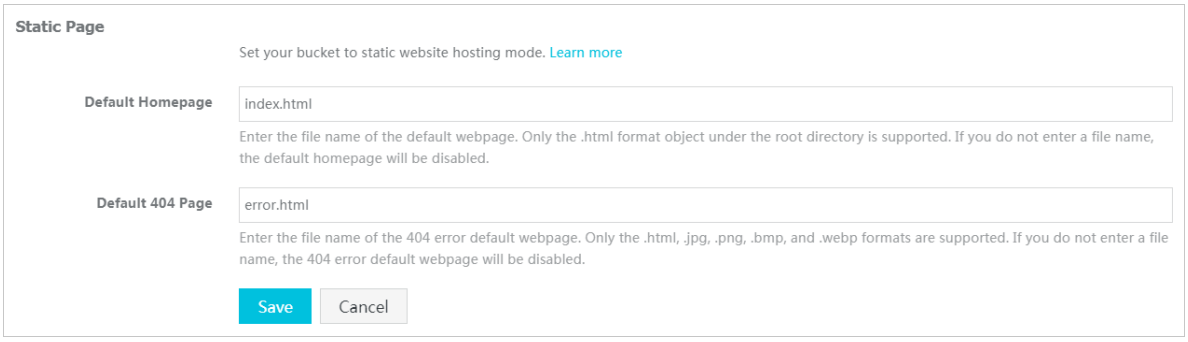
- Body of error.html:

```
<html>
  <head>
      <title>Hello OSS! </title>
      <meta charset="utf-8">
  </head>
  <body>
      <p>This is an error homepage for OSS Static Website
Hosting.</p>
  </body>
</html>
```

- `aliyun-logo.png` is a picture.

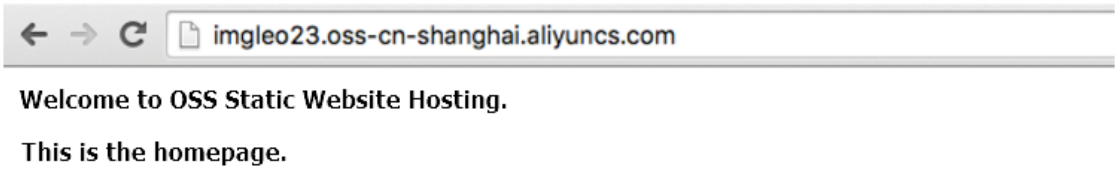
3. Activate static website hosting on OSS

As shown in the following figure, once you log on to the OSS console, set **`Default Homepage`** to `index.html` and **`Default 404 Page`** to `error.html`. For more information, see *Set static website hosting*.



To test the Static Website Hosting function, enter the URL as shown in the following figure:

- Display the default homepage:



When a similar URL is entered, the body of index.html specified upon activating the function is displayed.

- Display normal files

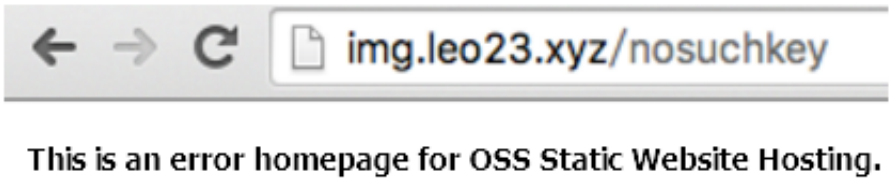
When a matched file for the entered URL is found, data is read successfully.

4. Access OSS with custom domain names

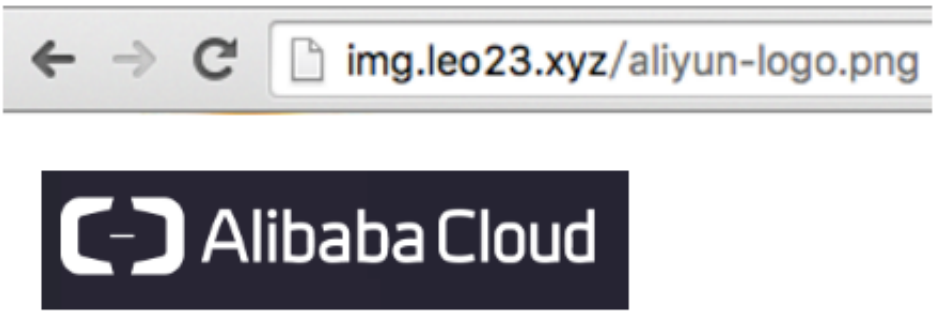
For more information about how to access OSS with custom domain names, see *Access OSS with custom domain names*.

- Display the default homepage

- Display the default 404 page



- Display normal files



FAQ

- What are the benefits of OSS Static Website Hosting?

  An ECS instance is saved in case any user needs a relatively small amount of traffic. In the case of larger traffic volumes, CDN can be used.

- How is OSS priced? How does OSS work with CDN?

  For pricing, see the OSS and CDN prices on Alibaba Cloud website. For cases on combination of OSS and CDN, see*CDN-based OSS acceleration practices*.

- Do the default homepage and default 404 page both need to be set?

  The default homepage needs to be set, whereas the default 404 page does not need to be set.

- Why does the browser return a 403 error after a URL is entered?

  The reason may be that the bucket permission is not public-read, or your Static Website Hosting function is suspended due to overdue payment.

4 Access control

4.1 Overview

Alibaba Cloud's permission management mechanism includes Resource Access Management (RAM) and Security Token Service (STS). This enables users to access OSS through subaccount s with different permissions and grants users temporary access authorization. Usage of RAM and STS can greatly improve management flexibility and security.

The following content is introduced in permission management:

- *What is RAM and STS*
- *Access a bucket without using the primary account*
- *Read/Write permission separation*
- *Bucket permission separation*
- *Access control*
- *STS temporary access authorization*
- *The problem of OSS authority and Its Troubleshooting*
- *STS frequently asked questions and troubleshooting*
- *OSS sub-account setup Frequently Asked Questions*

Click *RAM Policy Editor* Online Editing allows you to generate authorization policies.

4.2 What is RAM and STS

RAM and STS are permission management systems provided by Alibaba Cloud.

RAM is primarily used to control account system permissions. RAM enables users to create subaccounts within the range of primary account permissions. Different subaccounts can be allocated different permissions for authorization management.

STS is a security credential (token) management system that grants temporary access permission s. STS allows users to grant access rights to the temporary accounts.

Why RAM and STS?

RAM and STS are designed to resolve the core issue such as how to securely grant access permissions to other users without disclosing the primary account's AccessKey. Disclosure of AccessKey poses a serious security threat because unauthorized users may operate account resources and the risk of data leakage or stealing of important information is high.

RAM provides a long-term permission control mechanism. Various subaccounts assign different permissions to the different users. This way, even the disclosure of subaccount information would not cause a global information leakage. However, subaccounts have long-term validity.

> **Note:**
> Therefore, AccessKey of subaccounts must not be disclosed.

On the contrary, STS provides temporary access authorization by returning a temporary AccessKey and the token. This information can be provided directly to the temporary accounts, allowing them access to OSS. Generally, the permissions obtained from STS are more restrictive and only valid for a limited period of time. Thus, the disclosure of this information has little effect on the system.

These functions are further illustrated with the help of examples.

Basic concepts

The following are some explanations of the basic concepts:

- Subaccount: A subaccount is created from the Alibaba Cloud primary accounts. Once created , it is assigned an independent password and permissions. Each subaccount has its own AccessKey and can perform authorized operations similar to the primary account. Generally, subaccounts can be understood as users with certain permissions or operators with permission s to perform specific operations.
- Role: Role is a virtual concept for certain operation permissions. However, it does not have independent logon passwords or AccessKeys.

  > **Note:**
  > Subaccounts can assume roles. When a role is assumed, the permissions granted for a subaccount are the permissions of the role.
- Policy: Policies are rules used to define permissions; for example, they permit users to read or write certain resources.
- Resource: Resources are the cloud resources that users can access like all OSS buckets, a certain OSS bucket, or a certain object in a specific OSS bucket.

A subaccount and roles have the same relationship to each other as you and your identities. At work, you may be an employee, while at home you may be a father. In different scenarios, you may assume different roles. Different roles are assigned corresponding permissions. The concept of “employee” or “father” is not an actual entity that can be the subject of actions. These concepts

are only complete when an individual assumes them. This illustrates an important concept: a role may be assumed by multiple people at the same time.

Note:

Once the role is assumed, this individual automatically obtains all the permissions of the role.

The following example provides better understanding of the concept:

- Assume that Alice is the the Alibaba Cloud user and she has two private OSS buckets, alice_a and alice_b. Alice has full permission for both buckets.
- To avoid leaking her Alibaba Cloud account AccessKey, which would pose a major security risk , Alice uses RAM to create two subaccounts, Bob and Carol. Bob has read/write permission for alice_a and Carol has read/write permission for alice_b. Bob and Carol both have their own AccessKeys. This way, if one is leaked, only the corresponding bucket is affected and Alice can easily cancel the leaked user permissions on the console.
- Now, for some reason, Alice must authorize another person to read the objects in alice_a. In this situation, she must not only disclose Bob's AccessKey. Rather, she can create a new role like AliceAReader, and grant this role the read permission for alice_a. However, note that, at this time, AliceAReader cannot be used because no AccessKey corresponds to this role. AliceAReader is currently only a virtual entity with the permission to access alice_a.
- To obtain temporary authorization, Alice can call the STS's AssumeRole interface to notify STS that Bob wants to assume the AliceAReader role. If successful, STS returns a temporary AccessKeyId, AccessKeySecret, and SecurityToken, which serve as the access credentials. When these credentials are given to a temporary account, the user obtains temporary permission to access alice_a. The credentials' expiration time is specified when the AssumeRole interface is called.

Why are RAM and STS so complex?

Initially, RAM and STS concepts seem to be complex. This is because flexibility is given to permission control at the cost of simplicity.

Subaccounts and roles are separated to separate the entity that executes operations from the virtual entity that represents a permissions set. If a user requires many permissions including the read and write permissions but each operation only requires part of the total permission set, you can create two roles, one with the read permission and the other with the write permission. Then create a user who does not have any permission but can assume these two roles. When the user needs to read or write data, the user can temporarily assume the role with the read permission

or the role with the write permission. This reduces the risk of permission leaks for each operation . Additionally, roles can be used to grant permissions to other Alibaba Cloud users, making the collaboration easier.

Here, flexibility does not mean you have to use all these functions. You only need to use the subset of the functions as required. For example, if you do not need to use temporary access credentials that have an expiration time, you can only use the RAM subaccount function, without STS.

In what follows, we use examples to create a RAM and STS user guide and provide instructio ns. For the operations in these examples, we do our best to use console and command line operations to reduce the actual amount of codes that must be used. If you must use code to perform these operations, we recommend that you see the RAM and STS API Manual.

Test tool

During testing, we use osscmd, a tool in the OSS PythonSDK that allows you to directly work on OSS through the command line. osscmd can be obtained from *PythonSDK*.

Typical osscmd usage:

```
Download files
./osscmd get oss://BUCKET/OBJECT LOCALFILE --host=Endpoint -i
AccessKeyId -k AccessKeySecret
Here, replace BUCKET and OBJECT with your own bucket and object, and
the endpoint format must be similar to oss-cn-hangzhou.aliyuncs.com.
For AccessKeyId and AccessKeySecret, use the information corresponding
 to your own account
Upload files
./osscmd put LOCALFILE oss://BUCKET/OBJECT --host=Endpoint -i
AccessKeyId -k AccessKeySecret
The meaning of each field is the same as for the download example
```

4.3 Access a bucket without using the primary account

Assume that the user is a mobile developer and currently only has one bucket, ram-test-dev, for development, testing, and other functions. The user must stop using the primary account to access this bucket. This can avoid problems caused by AccessKey and password leaks. In the following example, replace AccessKey with your own AccessKey. The procedure is as follows:

1. On the console, select **Products and Services** > **Resource Access Management**.

   Note:

   The service must be activated first if you have never used it before.

2. Click **Users** to go to the **User Management** page.

3. The page shows that no user is created. Click **New User** on the upper right corner to create a subaccount with the same OSS access permissions as the primary account. Remember to select the **Auto generate AccessKey for this user**.
4. The AccessKey for this account is generated and must be saved for later use.
5. Return to **User Management**interface, which shows the newly created account named ram_test. When created, this subaccount does not have any permissions yet. Click the **Authorize** link on the right side and grant this subaccount full access permissions for OSS.

After authorization, click the **Management** link on the right side if you want to give the subaccount console logon or other permissions.

Now we can test the uploading and downloading operations. In the example, the AccessKey is ram_test's AccessKey. During the test, replace this with your own AccessKey.

```
$./osscmd get
oss://ram-test-dev/test.txt test.txt --host=oss-cn-hangzhou.aliyuncs.
com -i oOhue******Frogv -k OmVwFJO3qcT0******FhOYpg3p0KnA
100% The object test.txt is downloaded to test.txt, please check.
0.069(s) elapsed
```

```
$./osscmd put test.txt oss://ram-test-dev/test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100%
Object URL is: http://ram-test-dev.oss-cn-hangzhou.aliyuncs.com/test.
txt
Object abstract path is: oss://ram-test-dev/test.txt
ETag is "E27172376D49FC609E7F46995E1F808F"
0.108(s) elapsed
```

As you can see, this subaccount can basically be used for all operations, so you can avoid leaking the primary account's AccessKey.

4.4 Read/Write permission separation

When the users want to use an application server to provide external service, OSS can store back-end static resources. In this case, we recommend that the application server be granted the OSS read-only permission to reduce the risk of attacks. The read and write permission separation can be configured to grant the application server a user with the read-only permission.

1. Create an account ram_test_pub. As shown in the following figure, select ReadOnly in the authorization management area:

2. You can now use the AccessKey of the subaccount to test the upload and download permissions. The AccessKey here is a ram_test_pub AccessKey and is to be replaced with your own AccessKey during the test.

```
$./osscmd get oss://ram-test-dev/test.txt test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100% The object test.txt is downloaded to test.txt, please check.
0.070(s) elapsed
```

```
$. /Osscmd put test.txt OSS: // Ram-test-dev/test.txt -- Host =
porteroohue ****** frogv-K OmVwFJO3qcT0 * FhOYpg3p0KnA?
100% Error Headers:
[('content-length', '229'), ('server', 'AliyunOSS'), ('connection', '
keep-alive'), ('x-oss-request-id', '5646E49C1790CF0F531BAE0D'), ('date
', 'Sat, 14 Nov 2015 07:37:00 GMT'), ('content-type', 'application/xml
')]
Error Body:
<? xml version="1.0" encoding="UTF-8"? >
<Error>
  <Code>AccessDenied</Code>
  <Message>AccessDenied</Message>
  <RequestId>5646E49C1790CF0F531BAE0D</RequestId>
  <HostId>ram-test-dev.oss-cn-hangzhou.aliyuncs.com</HostId>
</Error>
Error Status:
403
put Failed!
```

With reference to the preceding example, we can conclude that the ram_test_pub account cannot be used to upload files.

4.5 Bucket permission separation

Another scenario is introduced in this section. If another user is using the developed app, you can use an individual bucket to store your app data. Assume that the bucket is the ram-test-app. In consideration of permission separation, the application server must not be allowed to access the ram-test-app; that is, the account ram_test_pub is permitted only to read ram-test-dev. This can also be realized through the RAM permission system. The procedure is as follows:

1. Because the system has no default bucket-level policy, we must create a custom policy.

   The bucket access policy is shown as follows. For more information, see *RAM Policy Description* and *OSS Authorization FAQs*.

```
   "Version": "1",
   "Statement": [

       "Effect": "Allow",
```

```
      "Action": [
        "oss:ListObjects",
        "oss:GetObject"

      "Resource": [
        "acs:oss:*:*:ram-test-dev",
        "acs:oss:*:*:ram-test-dev/*"
```

After setting, we can see the policy in the custom authorization policy list.

2. In user authorization management, add this policy to the selected authorization policy list. Also in**Users** > **Management** > **Authorization policy,**all previously granted OSS read permissions can be revoked.

3. Test the validity of permission configured.

   - The object in ram-test-dev can be accessed:

     ```
     $./osscmd get oss://ram-test-dev/test.txt test.txt --host=oss-cn
     -hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
     FhOYpg3p0KnA
     100% The object test.txt is downloaded to test.txt, please check.
     0.047(s) elapsed
     ```

   - The object in ram-test-app cannot be accessed:

     ```
     $./osscmd get oss://ram-test-app/test.txt test.txt --host=oss-cn
     -hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
     FhOYpg3p0KnA
      Error Headers:
      [('content-length', '229'), ('server', 'AliyunOSS'), ('connection
     ', 'keep-alive'), ('x-oss-request-id', '5646ED53F9EEA2F3324191A2
     '), ('date', 'Sat, 14 Nov 2015 08:14:11 GMT'), ('content-type', '
     application/xml')]
      Error Body:
      <? xml version="1.0" encoding="UTF-8"? >
      <Error>
        <Code>AccessDenied</Code>
        <Message>AccessDenied</Message>
        <RequestId>5646ED53F9EEA2F3324191A2</RequestId>
        <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
      </Error>
      Error Status:
      403
     ```

```
 get Failed!
```

- Files cannot be uploaded to oss-test-app:

```
$./osscmd put test.txt oss://ram-test-app/test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
 100% Error Headers:
 [('content-length', '229'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '5646ED7BB8DE437A912DC7A8
'), ('date', 'Sat, 14 Nov 2015 08:14:51 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <? xml version="1.0" encoding="UTF-8"? >
 <Error>
   <Code>AccessDenied</Code>
   <Message>AccessDenied</Message>
   <RequestId>5646ED7BB8DE437A912DC7A8</RequestId>
   <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
 </Error>
 Error Status:
 403
 put Failed!
```

Using the preceding configuration, we have successfully separated the permissions for ram-test-dev and ram-test-app.

The preceding section explains how to use the subaccount permission control function to separate permissions and minimize the potential risk of information leakage.

If you want to implement more complex access control, see *RAM User Guide*.

4.6 STS temporary access authorization

In the previous documents, we used only the RAM user functions. These user accounts are for long-term normal use. This poses as a serious risk if the RAM user permissions cannot be promptly revoked in case of information leakage.

In the previous example, assume that our developer's app allows users to upload data to the OSS bucket am-test-app and currently, the number of app users is large. In this case, how can the app securely grant data upload permissions to many users and how can it be certain of storage isolation among multiple users?

In such scenarios, we need to grant users temporary access using STS. STS can be used to specify a complex policy that restricts specified users by only granting them the minimum necessary permissions.

Create a role

Based on the example in the previous document, the app user has a bucket, ram-test-app, to store personal data. A role can be created as follows:

1. Create a RAM user account named ram_test_app using the process illustrated in the previous documents. Do not grant this account any permissions, because it inherits the permissions of a role which it assumes.
2. Create roles. Here you must create two roles for users to perform read operations and to upload files respectively.
   - Log on to the RAM console and select **Roles** > **Create Role**.
   - Select a role type. Here you must select `User role`.
   - Enter the role type information. Because this role has been used by its own Alibaba Cloud account. Use the default setting.
   - Configure basic role information.
3. When the role was created, it did not have any permissions. Therefore, we must create a custom authorization policy using the process described earlier. The following is the authorization policy:

```
"Version": "1",
"Statement": [

   "Effect": "Allow",
   "Action": [
     "oss:ListObjects",
     "oss:GetObject"

   "Resource": [
     "acs:oss:*:*:ram-test-app",
     "acs:oss:*:*:ram-test-app/*"
```

   This indicates read-only permission for ram-test-app.

4. After the policy is established, give the role RamTestAppReadOnly the ram-test-app read-only permission on the role management page.

5. Perform the same procedure to create the role RamTestAppWrite and use a custom authorization policy to grant ram-test-app write permission. The authorization policy is as follows:

```
"Version": "1",
"Statement": [

   "Effect": "Allow",
   "Action": [
     "oss:DeleteObject",
     "oss:ListParts",
     "oss:AbortMultipartUpload",
     "oss:PutObject"

   "Resource": [
     "acs:oss:*:*:ram-test-app",
     "acs:oss:*:*:ram-test-app/*"
```

Now we have created two roles, RamTestAppReadOnly and RamTestAppWrite, with read-only and write permissions for ram-test-app, respectively.

Temporary access authorization

After creating roles, we can use them to grant temporary access to OSS.

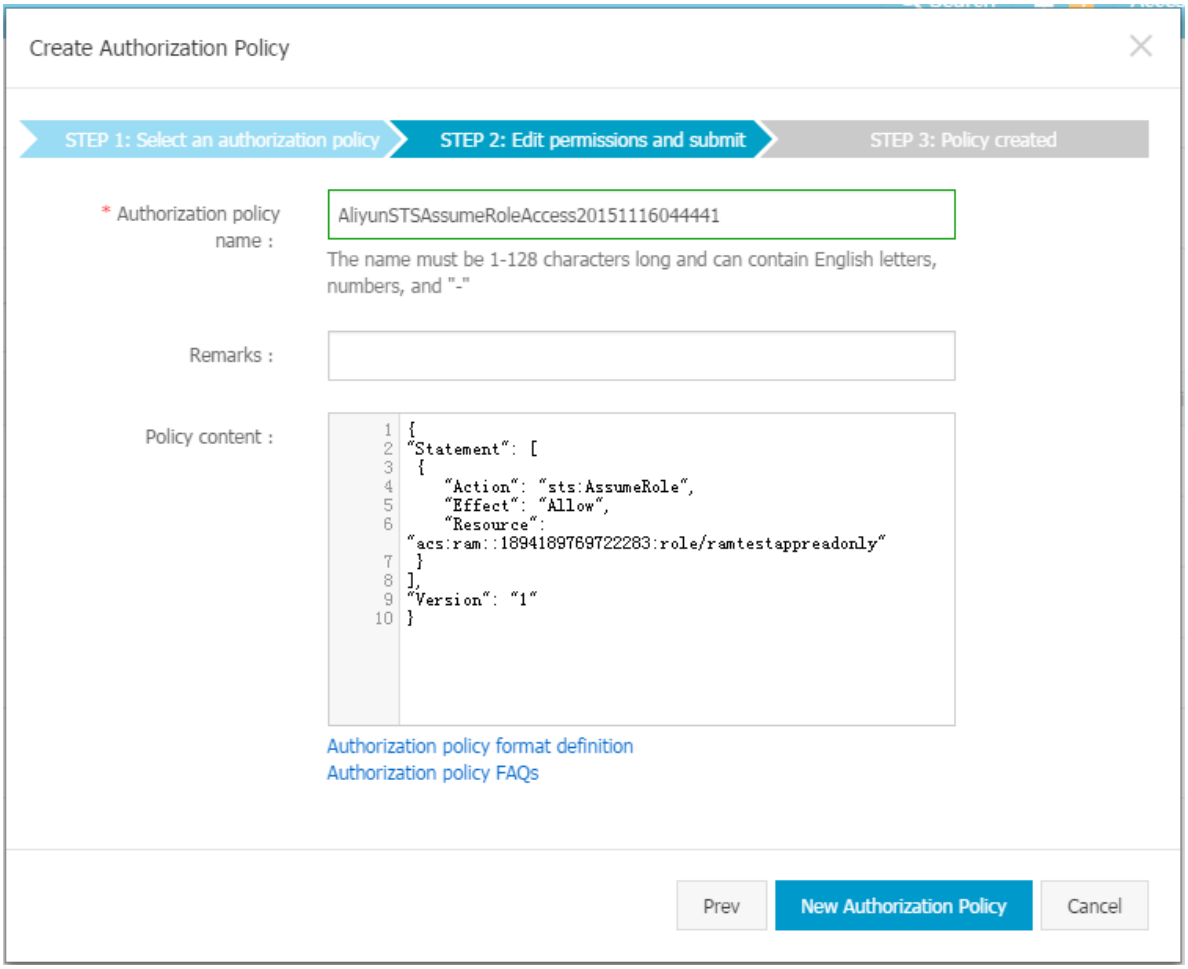
Preparation

Authorization is required for assuming roles. Otherwise, any RAM user could assume these roles , which can lead to unpredictable risks. Therefore, to assume corresponding roles, a RAM user needs to have explicitly configured permissions.

1. Create two custom authorization policies in authorization policy management.

```
"Statement": [

   "Action": "sts:AssumeRole",
   "Effect": "Allow",
   "Resource": "acs:ram::1894189769722283:role/ramtestappreadonly"
```

```
"Version": "1"
```




```
"Statement": [

   "Action": "sts:AssumeRole",
   "Effect": "Allow",
   "Resource": "acs:ram::1894189769722283:role/ramtestappwrite"


"Version": "1"
```

Here, the content entered after Resource is a role's ID. Role IDs can be found in**Roles** > **Role Details** .

2. Grant the two authorization policies to the account ram_test_app.

Use STS to grant access permissions

Now, we are ready with the platform to officially use STS to grant access permissions.

Here we use a simple STS Python command line tool *sts.py*. The calling method is as follows:

```
$python ./sts.py AssumeRole RoleArn=acs:ram::1894189769722283:role
/ramtestappreadonly RoleSessionName=usr001 Policy='{"Version":"1
","Statement":[{"Effect":"Allow","Action":["oss:ListObjects","oss:
GetObject"],"Resource":["acs:oss:*:*:ram-test-app","acs:oss:*:*:ram-
test-app/*"]}]}' DurationSeconds=1000 --id=id --secret=secret
```

- RoleArn: indicates the ID of a role to be assumed. Role IDs can be found in**Roles** > **Role Details** .
- RoleSessionName: indicates the name of the temporary credentials. Generally, we recommend that you separate this using different application users.
- Policy: indicates a permission restriction, which is added when the role is assumed.
- DurationSeconds: indicate the validity time of the temporary credentials in seconds. The minimum value is 900, and the maximum value is 3600.
- id and secret: indicate the AccessKey of the RAM user to assume a role.

Here, we need to explain what is meant by “Policy”. The policy mentioned here is used to restrict the temporary credential permissions after a role is assumed. Ultimately, the permissions obtained by means of temporary credentials are overlapping permissions of the role and the policy passed in.

When a role is assumed, a policy can be entered to increase the flexibility. For example, when uploading the files, we can add different upload path restrictions for different users. This is shown in the following example.

Now, let’s test the STS function. To test the bucket, first use the console to put the file test.txt in ram-test-app, with the content ststest.

Firstly, use the RAM user account ram_test_app to directly access the file. Next, replace AccessKey with your own AccessKey used in the test.

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
$./osscmd get oss://ram-test-app/test.txt test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
Error Headers:
[('content-length', '229'), ('server', 'AliyunOSS'), ('connection', '
keep-alive'), ('x-oss-request-id', '564A94D444F4D8B2225E4AFE'), ('date
', 'Tue, 17 Nov 2015 02:45:40 GMT'), ('content-type', 'application/xml
')]
Error Body:
<? xml version="1.0" encoding="UTF-8"? >
<Error>
  <Code>AccessDenied</Code>
  <Message>AccessDenied</Message>
  <RequestId>564A94D444F4D8B2225E4AFE</RequestId>
```

```
  <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
</Error>
Error Status:
403
get Failed!
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
$./osscmd put test.txt oss://ram-test-app/test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100% Error Headers:
[('content-length', '229'), ('server', 'AliyunOSS'), ('connection', '
keep-alive'), ('x-oss-request-id', '564A94E5B1119B445B9F8C3A'), ('date
', 'Tue, 17 Nov 2015 02:45:57 GMT'), ('content-type', 'application/xml
')]
Error Body:
<? xml version="1.0" encoding="UTF-8"? >
<Error>
  <Code>AccessDenied</Code>
  <Message>AccessDenied</Message>
  <RequestId>564A94E5B1119B445B9F8C3A</RequestId>
  <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
</Error>
Error Status:
403
put Failed!
```

Without access permission, access attempts using the RAM user account ram_test_app are failed

.

Use temporary authorization for downloads

Now, we use STS to download files. To make it simple to understand, the entered policy and the role policy are the same. The expiration time is set to 3600s, and the app user here is usr001. The steps are as follows:

1. Use STS to obtain a temporary credential.

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $python ./sts.py AssumeRole RoleArn=acs:ram::1894189769722283:role
/ramtestappreadonly RoleSessionName=usr001 Policy='{"Version":"1
","Statement":[{"Effect":"Allow","Action":["oss:ListObjects","oss:
GetObject"],"Resource":["acs:oss:*:*:ram-test-app","acs:oss:*:*:ram
-test-app/*"]}]}' --id=oOhue******Frogv --secret=OmVwFJO3qcT0******
FhOYpg3p0KnA
 Https://sts.aliyuncs.com /? Signatureversion = 1.0 & format = JSON
 & timestamp = glas% 3a07% 3a25z & rolearn = ACS % 3-Aram % 3A %
3A1894189769722283% 3 Arole % 2 framtestappreadonly & RoleSessio
nName = usr001 & the Access Key ID = oOhuek56i53Frogv & Policy = %
7b % 22 version % 22% 3A % 221% 22% 2C % 22 Statement % 22% 3A %
1B % 7b % 22 effect % 22% 3A % 22 allow % 22% 2C % 22 action % 22
% 3A % 1B % 22oss % 3 alistobjects % 22% 2C % 22oss % 3 agetobject
 % 22% 5D % 2C % 22 Resource % 22% 3A % 4B % 22acs % 3 aoss % 3A %
2a % 3A % 2a % fig % 22% 2C % 22acs % 3 aoss % 3A % 2a % 3A % 2a %
3aram-test-app % Sch % 2a % 22% 5d % 7D % 5d % 7D & seaguremethod =
HMAC-SHA1 & Version = 2015-04-01 & Signature = bshxPZpwRJv5ch3SjaBi
XLodwq0 % 3D & Action = AssumeRole & signaturenonce = 53e1be9c-8cd8-
11e5-9b86-008cfa5e4938
```

```
   "AssumedRoleUser": {
     "Arn": "acs:ram::1894189769722283:role/ramtestappreadonly/
usr001",
     "AssumedRoleId": "317446347657426289:usr001"

   "Credentials": {
     "AccessKeyId": "STS. 3mQEbNf******wa180Le",
     "AccessKeySecret": "B1w7rCbR4dzGwNYJ******3PiPqKZ3gjQhAxb6mB",
     "Expiration": "2015-11-17T04:07:25Z",
     "SecurityToken": "CAESvAMIARKAASQQUUTSE+7683CGlhdGsv2/di8uI+
X1BxG7MDxM5FTd0fp5wpPK/7UctYH2MJ///c4yMN1PUCcEHI1zppCINmpDG2XeNA3
OS16JwS6ESmI50sHyWBmsYkCJW15gXnfhz/OK+mSp1bYxlfB33qfgCFe97Ijeuj8RMgq
Fx0Hny2BzGhhTVFMuM21RRWJOZnR5Yzl1T3dhMTgwTGUiEjMxNzQ0NjM0NzY
1NzQyNjI4OSoGdXNyMDAxMJTrgJ2RKjoGUnNhTUQ1QpsBCgExGpUBCgVBbGx
vdxI4CgxBY3Rpb25FcXVhbHMSBkFjdGlvbhogCg9vc3M6TGlzdE9iamVjdHM
KDW9zczpHZXRPYmplY3QSUgoOUmVzb3VyY2VFcXVhbHMSCFJlc291cmNlGjY
KGGFjczpvc3M6KjoqOnJhbS10ZXN0LWFwcAoaYWNzOm9zczoqOio6cmFtLXR
lc3QtYXBwLypKEDE4OTQxODk3Njk3MjIyODNSBTI2ODQyWg9Bc3N1bWVkUm9
sZVVzZXJgAGoSMzE3NDQ2MzQ3NjU3NDI2Mjg5chJyYW10ZXN0YXBwcmVhZG9ubHk="

   "RequestId": "8C009F64-F19D-4EC1-A3AD-7A718CD0B49B"
```

2. Use the temporary credential to download files. Here, sts_token is the SecurityToken returned by the STS.

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd get oss://ram-test-app/test.txt test.txt --host=oss
-cn-hangzhou.aliyuncs.com -i STS. 3mqebnf *** wa180le-k*** ***
3pipqkz3gjqhaxb6mb *** -- sts_token = caesvamiarkaasqquutse +
7683CGlhdGsv2/di8uI + X1BxG7MDxM5FTd0fp5wpPK/7UctYH2MJ/c4yMN1PUCc
EHI1zppCINmpDG2XeNA3OS16JwS6ESmI50sHyWBmsYkCJW15gXnfhz/OK +
mSp1bYxlfB33qfgCFe97Ijeuj8RMgqFx0Hny2BzGhhTVFMuM21RRWJOZnR5Y
zl1T3dhMTgwTGUiEjMxNzQ0NjM0NzY1NzQyNjI4OSoGdXNyMDAxMJTrgJ2RK
joGUnNhTUQ1QpsBCgExGpUBCgVBbGxvdxI4CgxBY3Rpb25FcXVhbHMSBkFjd
GlvbhogCg9vc3M6TGlzdE9iamVjdHMKDW9zczpHZXRPYmplY3QSUgoOUmVzb
3VyY2VFcXVhbHMSCFJlc291cmNlGjYKGGFjczpvc3M6KjoqOnJhbS10ZXN0L
WFwcAoaYWNzOm9zczoqOio6cmFtLXRlc3QtYXBwLypKEDE4OTQxODk3Njk3M
jIyODNSBTI2ODQyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzE3NDQ2MzQ3NjU3N
DI2Mjg5chJyYW10ZXN0YXBwcmVhZG9ubHk =
 100% The object test.txt is downloaded to test.txt, please check.
 0.061(s) elapsed
```

3. As you can see, we can use the temporary credentials to download the file. Next, we test if we can use them to upload a file.

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd put test.txt oss://ram-test-app/test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i STS. 3mQEbNf******wa180Le -k B1w7rCbR4d
zGwNYJ******3PiPqKZ3gjQhAxb6mB --sts_token=CAESvAMIARKAASQQUUTSE+
7683CGlhdGsv2/di8uI+X1BxG7MDxM5FTd0fp5wpPK/7UctYH2MJ///c4yMN1PUCc
EHI1zppCINmpDG2XeNA3OS16JwS6ESmI50sHyWBmsYkCJW15gXnfhz/OK+mSp1bYxlfB
33qfgCFe97Ijeuj8RMgqFx0Hny2BzGhhTVFMuM21RRWJOZnR5Yzl1T3dhMTg
wTGUiEjMxNzQ0NjM0NzY1NzQyNjI4OSoGdXNyMDAxMJTrgJ2RKjoGUnNhTUQ
1QpsBCgExGpUBCgVBbGxvdxI4CgxBY3Rpb25FcXVhbHMSBkFjdGlvbhogCg9
vc3M6TGlzdE9iamVjdHMKDW9zczpHZXRPYmplY3QSUgoOUmVzb3VyY2VFcXV
hbHMSCFJlc291cmNlGjYKGGFjczpvc3M6KjoqOnJhbS10ZXN0LWFwcAoaYWN
zOm9zczoqOio6cmFtLXRlc3QtYXBwLypKEDE4OTQxODk3Njk3MjIyODNSBTI
2ODQyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzE3NDQ2MzQ3NjU3NDI2Mjg5chJ
yYW10ZXN0YXBwcmVhZG9ubHk=
```

```
 100% Error Headers:
 [('Content-Length', '254'), ('server ', 'aliyunoss '), ('Connection
', 'keep-alive '), ('x-OSS-request-id', '564a9a2a1790cf0f53c15c82
 '), ('date', 'tue, 17 2015 03:08:26 GMT '), ('content-type', '
application/XML ')]
 Error Body:
 <? xml version="1.0" encoding="UTF-8"? >
 <Error>
   <Code>AccessDenied</Code>
   <Message>Access denied by authorizer's policy.</Message>
   <RequestId>564A9A2A1790CF0F53C15C82</RequestId>
   <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
 </Error>
 Error Status:
 403
 put Failed!
```

The file upload is failed. This is because the assumed role only has download permission hence.

Use temporary authorization for uploads

Now, we try to use STS to upload a file. The steps are as follows:

1. Obtain an STS temporary credential. The app user is usr001.

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $python ./sts.py AssumeRole RoleArn=acs:ram::1894189769722283:role
/ramtestappwrite RoleSessionName=usr001 Policy='{"Version":"1","
Statement":[{"Effect":"Allow","Action":["oss:PutObject"],"Resource
":["acs:oss:*:*:ram-test-app/usr001/*"]}]}' --id=oOhue******Frogv --
secret=OmVwFJO3qcT0******FhOYpg3p0KnA
 https://sts.aliyuncs.com/?SignatureVersion=1.0&Format=JSON&
Timestamp=2015-11-17T03%3A16%3A10Z&RoleArn=acs%3Aram%3A%3A18941897
69722283%3Arole%2Framtestappwrite&RoleSessionName=usr001&AccessKeyI
d=oOhuek56i53Frogv&Policy=%7B%22Version%22%3A%221%22%2C%22Statemen
t%22%3A%5B%7B%22Effect%22%3A%22Allow%22%2C%22Action%22%3A%5B%22oss
%3APutObject%22%5D%2C%22Resource%22%3A%5B%22acs%3Aoss%3A%2A%3A%2A%
3Aram-test-app%2Fusr001%2F%2A%22%5D%7D%5D%7D&SignatureMethod=HMAC-
SHA1&Version=2015-04-01&Signature=Y0OPUoL1PrCqX4X6A3%2FJvgXuS6c%3D&
Action=AssumeRole&SignatureNonce=8d0798a8-8cd9-11e5-9f49-008cfa5e49
38

   "AssumedRoleUser": {
     "Arn": "acs:ram::1894189769722283:role/ramtestappwrite/usr001
",
     "AssumedRoleId": "355407847660029428:usr001"

   "Credentials": {
     "AccessKeyId": "STS.rtfx13******NlIJlS4U",
     "AccessKeySecret": "2fsaM8E2maB2dn******wpsKTyK4ajo7TxFr0zIM",
     "Expiration": "2015-11-17T04:16:10Z",
     "SecurityToken": "CAESkwMIARKAAUh3/Uzcg13YLRBWxy0IZjGew
Mpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6DLcvPa8azKJKtzV0oKSy+
mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1pgxJCBzNZ2YV/6ycTaZySSE
1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzRFlNVWJjTmxJSmxTNFUiE
jM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzoJ2RKjoGUnNhTUQ1QnYKA
TEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY3Rpb24aDwoNb3NzOlB1d
E9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3VyY2UaIwohYWNzOm9zcz
oqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxODk0MTg5NzY5NzIyMjgzUg
```

```
UyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NTQwNzg0NzY2MDAyOTQyOH
IPcmFtdGVzdGFwcHdyaXRl"

   "RequestId": "19407707-54B2-41AD-AAF0-FE87E8870B0D"
```

2. Let us test if we can use the credentials to upload and download.

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd get oss://ram-test-app/test.txt test.txt --host=oss-cn-
hangzhou.aliyuncs.com -i STS.rtfx13******NlIJlS4U -k 2fsaM8E2maB2dn
******wpsKTyK4ajo7TxFr0zIM --sts_token=CAESkwMIARKAAUh3/Uzcg13YLRB
Wxy0IZjGewMpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6DLcvPa8azK
JKtzV0oKSy+mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1pgxJCBzNZ2
YV/6ycTaZySSE1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzRFlNVWJjTmx
JSmxTNFUiEjM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzoJ2RKjoGUnN
hTUQ1QnYKATEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY3Rpb24aDwo
Nb3NzOlB1dE9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3VyY2UaIwoh
YWNzOm9zczoqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxODk0MTg5NzY5
NzIyMjgzUgUyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NTQwNzg0NzY2
MDAyOTQyOHIPcmFtdGVzdGFwcHdyaXRl
 Error Headers:
 [('content-length', '254'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '564A9C31FFFC811F24B6E7E3
'), ('date', 'Tue, 17 Nov 2015 03:17:05 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <? xml version="1.0" encoding="UTF-8"? >
 <Error>
   <Code>AccessDenied</Code>
   <Message>Access denied by authorizer's policy.</Message>
   <RequestId>564A9C31FFFC811F24B6E7E3</RequestId>
   <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
 </Error>
 Error Status:
 403
 get Failed!
 [admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd put test.txt oss://ram-test-app/test.txt --host=oss-cn-
hangzhou.aliyuncs.com -i STS.rtfx13******NlIJlS4U -k 2fsaM8E2maB2dn
******wpsKTyK4ajo7TxFr0zIM --sts_token=CAESkwMIARKAAUh3/Uzcg13YLRB
Wxy0IZjGewMpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6DLcvPa8azK
JKtzV0oKSy+mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1pgxJCBzNZ2
YV/6ycTaZySSE1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzRFlNVWJjTmx
JSmxTNFUiEjM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzoJ2RKjoGUnN
hTUQ1QnYKATEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY3Rpb24aDwo
Nb3NzOlB1dE9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3VyY2UaIwoh
YWNzOm9zczoqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxODk0MTg5NzY5
NzIyMjgzUgUyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NTQwNzg0NzY2
MDAyOTQyOHIPcmFtdGVzdGFwcHdyaXRl
 100% Error Headers:
 [('content-length', '254'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '564A9C3FB8DE437A91B16772
'), ('date', 'Tue, 17 Nov 2015 03:17:19 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <? xml version="1.0" encoding="UTF-8"? >
 <Error>
   <Code>AccessDenied</Code>
   <Message>Access denied by authorizer's policy.</Message>
   <RequestId>564A9C3FB8DE437A91B16772</RequestId>
   <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
```

```
</Error>
Error Status:
403
put Failed!
```

The test.txt upload fails. We have formatted the entered policy discussed at the beginning of this document, which is as follows:

```
    "Version": "1",
        "Statement": [

            "Effect": "Allow",
            "Action": [
                "oss:PutObject"

            "Resource": [
                "acs:oss:*:*:ram-test-app/usr001/*"
```

This policy indicates that users are only allowed to upload files like usr001/ to the ram-test-app bucket. If the app user is usr002, the policy can be changed to only allow for the uploading of files like usr002/. By setting different policies for different app users, we can isolate the storage space of different app users.

3. Retry the test and specify the upload destination as ram-test-app/usr001/test.txt.

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd put test.txt oss://ram-test-app/usr001/test.txt --
host=oss-cn-hangzhou.aliyuncs.com -i STS.rtfx13******NlIJlS4U -k
 2fsaM8E2maB2dn******wpsKTyK4ajo7TxFr0zIM --sts_token=CAESkwMIAR
KAAUh3/Uzcg13YLRBWxy0IZjGewMpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6
DLcvPa8azKJKtzV0oKSy+mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1
pgxJCBzNZ2YV/6ycTaZySSE1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzR
FlNVWJjTmxJSmxTNFUiEjM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzo
J2RKjoGUnNhTUQ1QnYKATEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY
3Rpb24aDwoNb3NzOlB1dE9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3
VyY2UaIwohYWNzOm9zczoqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxOD
k0MTg5NzY5NzIyMjgzUgUyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NT
QwNzg0NzY2MDAyOTQyOHIPcmFtdGVzdGFwcHdyaXRl
 100%
 Object URL is: http://ram-test-app.oss-cn-hangzhou.aliyuncs.com/
usr001%2Ftest.txt
 Object abstract path is: oss://ram-test-app/usr001/test.txt
 ETag is "946A0A1AC8245696B9C6A6F35942690B"
 0.071(s) elapsed
```

The upload is successful.

Summary

This document describes how to grant users temporary access authorization for OSS using STS . In typical mobile development scenarios, STS can be used to grant temporary authorizations to access OSS when different app users need to access the app. The temporary authorization can be configured with expiration time to greatly reduce the hazards caused by leakages. When obtaining temporary authorization, we can enter different authorization policies for different app users to restrict their access permissions. For example, to restrict the object paths accessible to users. This isolates the storage space of different app users.

4.7 FAQs about subaccount settings

How to create an STS temporary account and how to use it to access resources?

See *STS temporary access authorization*.

Client or console logon error reported for an authorized sub-account

See *Why does a sub-account encounters an error of no operation permission for a bucket on the OSS console after it has been granted the bucket operation permission*.

How to authorize a sub-account with the operation permission for a single bucket

See *How to assign the full operation permission for a specified bucket to a sub-account*.

How to authorize a sub-account with the operation permission for a directory in a bucket

See *OSS directory authorization*

How to authorize a sub-account with the read-only permission for a bucket

See *Authorize a sub-user to list and read resources in a bucket*.

Error upon an OSS SDK call: InvalidAccessKeyId

See *STS errors and troubleshooting*.

Error upon an STS call: Access denied by authorizer's policy

Detailed error information: ErrorCode: AccessDenied ErrorMessage: Access denied by authorizer 's policy.

Cause of the error:

- The temporary account has no access permission.
- The authorization policy specified for assuming the role of this temporary account does not assign the access permission to the account.

For more STS errors and the causes, see *OSS permission errors and troubleshooting.*

5 Data security

5.1 Check data transmission integrity by using 64-bit CRC

Background

An error may occur when data is transmitted between the client and the server. Currently, OSS can return the 64-bit CRC value for an object uploaded in any mode. To check the data integrity, the client can compare the 64-bit CRC value with the locally calculated value.

- OSS calculates 64-bit CRC value for newly uploaded object, stores the result as metadata of the object, and then adds the x-oss-hash-crc64ecma header to the returned response header, indicating its 64-bit CRC value. This 64-bit CRC is calculated according to ECMA-182 Standard .
- For the object that already exists on OSS before the 64-bit CRC goes live, OSS does not calculate its 64-bit CRC value. Therefore, its 64-bit CRC value is not returned when such object is obtained.

Operation instructions

- Put Object / Append Object / Post Object / Multipart upload part returns the corresponding 64-bit CRC value. The client can get the 64-bit CRC value returned by the server after the upload is completed and can check it against the locally calculated value.
- In the case of Multipart Complete, if all the parts have their respective 64-bit CRC values, then the 64-bit CRC value of the entire object is returned. Otherwise, the 64-bit CRC value is not returned (for example, if a part has been uploaded before the 64-bit CRC goes live).
- Get Object / Head Object / Get ObjectMeta returns the corresponding 64-bit CRC value (if any). After Get Object is completed, the client can get the 64-bit CRC value returned by the server and check it against the locally calculated value.

   **Note:**
  The 64-bit CRC value of the entire object is returned for the range get object.
- For copy related operations, for example, Copy Object/Upload Part Copy, the newly generated object/Part may not necessarily have the 64-bit CRC value.

Python example

An example of complete Python code is as follows. It shows how to check data transmission integrity based on the 64-bit CRC value.

1. Calculate the 64-bit CRC value.

```
import oss2
from oss2.models import PartInfo
import os
import crcmod
import random
import string
do_crc64 = crcmod.mkCrcFun(0x142F0E1EBA9EA3693L, initCrc=0L, xorOut=
0xffffffffffffffffL, rev=True)
def check_crc64(local_crc64, oss_crc64, msg="check crc64"):
if local_crc64 ! = oss_crc64:
print "{0} check crc64 failed. local:{1}, oss:{2}.".format(msg,
local_crc64, oss_crc64)
return False
else:
print "{0} check crc64 ok.".format(msg)
return True
def random_string(length):
return ''.join(random.choice(string.lowercase) for i in range(length
))
bucket = oss2. Bucket(oss2. Auth(access_key_id, access_key_secret),
endpoint, bucket_name)
```

2. Verify Put Object.

```
content = random_string(1024)
 key = 'normal-key'
 result = bucket.put_object(key, content)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(content))
 check_crc64(local_crc64, oss_crc64, "put object")
```

3. Verify Get Object.

```
result = bucket.get_object(key)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(result.resp.read()))
 check_crc64(local_crc64, oss_crc64, "get object")
```

4. Verify Upload Part and Complete.

```
part_info_list = []
 key = "multipart-key"
 result = bucket.init_multipart_upload(key)
 upload_id = result.upload_id
 part_1 = random_string(1024 * 1024)
 result = bucket.upload_part(key, upload_id, 1, part_1)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_1))
 #Check whether the uploaded part 1 data is complete
 check_crc64(local_crc64, oss_crc64, "upload_part object 1")
 part_info_list.append(PartInfo(1, result.etag, len(part_1)))
 part_2 = random_string(1024 * 1024)
 result = bucket.upload_part(key, upload_id, 2, part_2)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_2))
 #Check whether the uploaded part 2 data is complete
 check_crc64(local_crc64, oss_crc64, "upload_part object 2")
 part_info_list.append(PartInfo(2, result.etag, len(part_2)))
```

```
 result = bucket.complete_multipart_upload(key, upload_id,
part_info_list)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_2, do_crc64(part_1)))
 #Check whether the final object on the OSS is consistent with the
local file
 check_crc64(local_crc64, oss_crc64, "complete object")
```

OSS SDK support

Part of the OSS SDK already supports the data validation using crc64 for the upload and download, as shown in the following table:

SDK	Support for CRC?	Example
Java SDK	Yes	*CRCSample.java*
Python SDK	Yes	*object_check.py*
PHP SDK	No	N/A
C# SDK	No	None
C SDK	Yes	*oss_crc_sample.c*
JavaScript SDK	No	None
Go SDK	Yes	*crc_test.go*
Ruby SDK	No	None
iOS SDK	No	*OSSCrc64Tests.m*
Android SDK	No	*OSSCrc64Tests.m*

5.2 Protect data through client encryption

Client encryption means that the encryption is completed before the user data is sent to the remote server, whereas the plaintext of the key used for encryption is kept in the local computer only. Therefore, the security of user data can be ensured because others cannot decrypt the data to obtain the original data even if the data leaks.

This document describes how to protect data through client encryption based on the current Python SDK version of OSS.

Principles

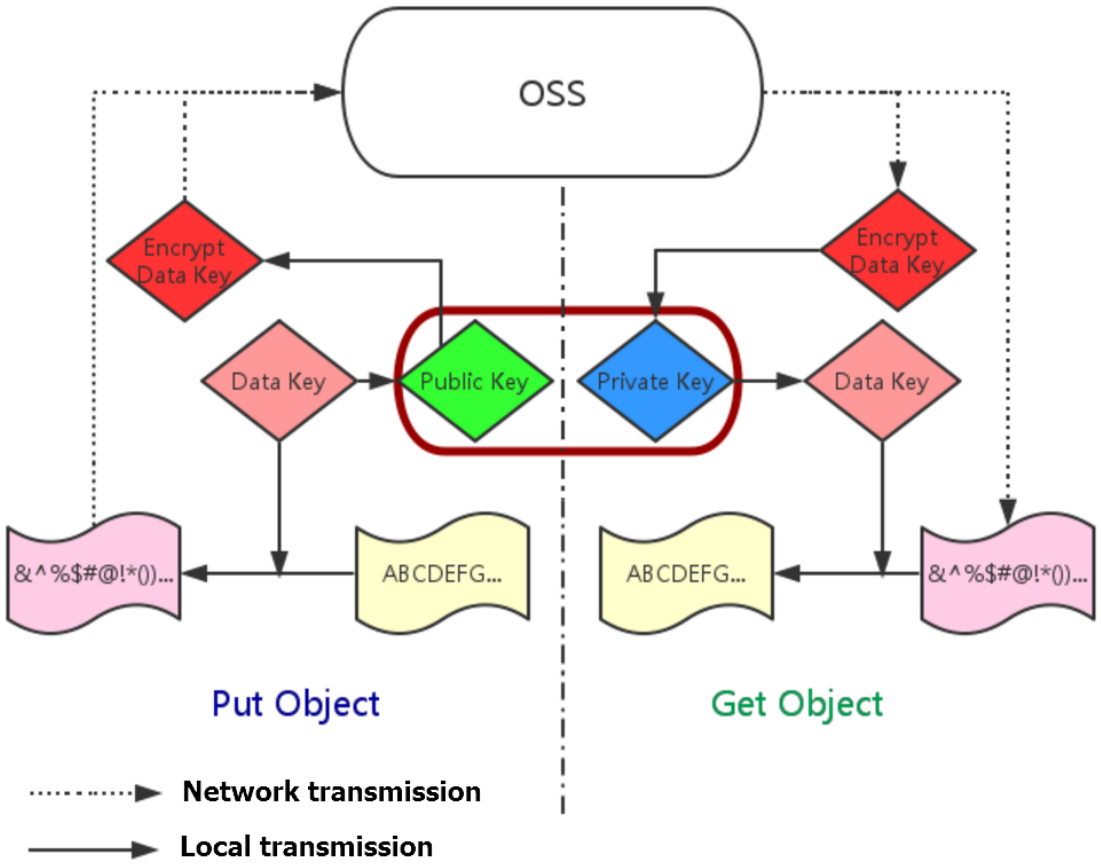
1. The user maintains a pair of RSA keys (`rsa_private_key` and `rsa_public_key`) in the local computer.

2. Each time when any object is uploaded, a symmetric key `data_key` of AES256 type is generated randomly, and then `data_key` is used to encrypt the original content to obtain encrypt_content.
3. Use `rsa_public_key` to encrypt `data_key` to obtain `encrypt_data_key`, place it in the request header as the custom meta of the user, and send it together with encrypt_content to the OSS.
4. When Get Object is performed, encrypt_content and `encrypt_data_key` in the custom meta of the user are obtained first.
5. The user uses `rsa_private_key` to decrypt `encrypt_data_key` to obtain `data_key`, and then uses `data_key` to decrypt encrypt_content to obtain the original content.

Note:

The user's key in this document is an asymmetric RSA key, and the AES256-CTR algorithm is used when object content is encrypted. For more information, see *PyCrypto Document*. This document describes how to implement client encryption through the custom meta of an object. The user can select the encryption key type and encryption algorithm as required.

Structural diagram



Preparation

1. For installation and usage of the Python SDK, see *Quick Installation of Python SDK*.
2. Install the PyCrypto library.

```
pip install pycrypto
```

Example of complete Python code

```
# -*- coding: utf-8 -*-
import os
import shutil
import base64
import random
import oss2
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Util import Counter
# aes 256, key always is 32 bytes
_AES_256_KEY_SIZE = 32
_AES_CTR_COUNTER_BITS_LEN = 8 * 16
class AESCipher:
```

```
    def __init__(self, key=None, start=None):
        self.key = key
        self.start = start
        if not self.key:
            self.key = Random.new().read(_AES_256_KEY_SIZE)
        if not self.start:
            self.start = random.randint(1, 10)
        ctr = Counter.new(_AES_CTR_COUNTER_BITS_LEN, initial_value=
self.start)
        self.cipher = AES.new(self.key, AES.MODE_CTR, counter=ctr)
    def encrypt(self, raw):
        return self.cipher.encrypt(raw)
    def decrypt(self, enc):
        return self.cipher.decrypt(enc)
# First, initialize the information such as AccessKeyId, AccessKeyS
ecret, and Endpoint.
# Obtain the information through environment variables or replace the
 information such as "<Your AccessKeyId>" with the real AccessKeyId,
and so on.

# Use Hangzhou region as an example. Endpoint can be:
# http://oss-cn-hangzhou.aliyuncs.com
# https://oss-cn-hangzhou.aliyuncs.com
# Access using the HTTP and HTTPS protocols respectively.
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '<your AccessKeyId
>')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '<Your
AccessKeySecret>')
bucket_name = os.getenv('OSS_TEST_BUCKET', '<Your Bucket>')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '<Your Access Domain Name>')
# Make sure that all the preceding parameters have been filled in
correctly.
for param in (access_key_id, access_key_secret, bucket_name, endpoint
):
    assert '<' not in param, 'Please set the parameter:' + param
##### 0 prepare ########
# 0.1 Generate the RSA key file and save it to the disk
rsa_private_key_obj = RSA.generate(2048)
rsa_public_key_obj = rsa_private_key_obj.publickey()
encrypt_obj = PKCS1_OAEP.new(rsa_public_key_obj)
decrypt_obj = PKCS1_OAEP.new(rsa_private_key_obj)
# save to local disk
file_out = open("private_key.pem", "w")
file_out.write(rsa_private_key_obj.exportKey())
file_out.close()
file_out = open("public_key.pem", "w")
file_out.write(rsa_public_key_obj.exportKey())
file_out.close()
# 0.2 Create the Bucket object. All the object-related interfaces can
be implemented by using the Bucket object
bucket = oss2. Bucket(oss2. Auth(access_key_id, access_key_secret),
endpoint, bucket_name)
obj_name = 'test-sig-1'
content = "test content"
#### 1 Put Object ####
# 1.1 Generate the one-time symmetric key encrypt_cipher used to
encrypt this object, where key and start are values generated at
random
encrypt_cipher = AESCipher()
# 1.2 Use the public key to encrypt the information for assisting
encryption, and save it in the custom meta of the object. When Get
 Object is performed later, we can use the private key to perform
```

```
decryption and obtain the original content according to the custom
meta
headers = {}
headers['x-oss-meta-x-oss-key'] = base64.b64encode(encrypt_obj.encrypt
(encrypt_cipher.key))
headers['x-oss-meta-x-oss-start'] = base64.b64encode(encrypt_obj.
encrypt(str(encrypt_cipher.start)))
# 1.3. Use encrypt_cipher to encrypt the original content to obtain
encrypt_content
encryt_content = encrypt_cipher.encrypt(content)
# 1.4 Upload the object
result = bucket.put_object(obj_name, encryt_content, headers)
if result.status / 100 ! = 2:
    exit(1)
#### 2 Get Object ####
# 2.1 Download the encrypted object
result = bucket.get_object(obj_name)
if result.status / 100 ! = 2:
    exit(1)
resp = result.resp
download_encrypt_content = resp.read()
# 2.2 Resolve from the custom meta the key and start that are
previously used to encrypt this object
download_encrypt_key = base64.b64decode(resp.headers.get('x-oss-meta-x
-oss-key', ''))
key = decrypt_obj.decrypt(download_encrypt_key)
download_encrypt_start = base64.b64decode(resp.headers.get('x-oss-meta
-x-oss-start', ''))
start = int(decrypt_obj.decrypt(download_encrypt_start))
# 2.3 Generate the cipher used for decryption, and decrypt it to
obtain the original content
decrypt_cipher = AESCipher(key, start)
download_content = decrypt_cipher.decrypt(download_encrypt_content)
if download_content ! = content:
    print "Error!"
else:
    print "Decrypt ok. Content is: %s" % download_content
```

6 OSS resource monitoring and alarm service

The CloudMonitor service can monitor OSS resources. You can use CloudMonitor to view resource usage, performance, and health status on Alibaba Cloud. Using the alarm service, you can react rapidly to keep applications running smoothly. This article introduces how to monitor OSS resources, set OSS alarm rules, and create custom monitoring dashboard.

Prerequisites

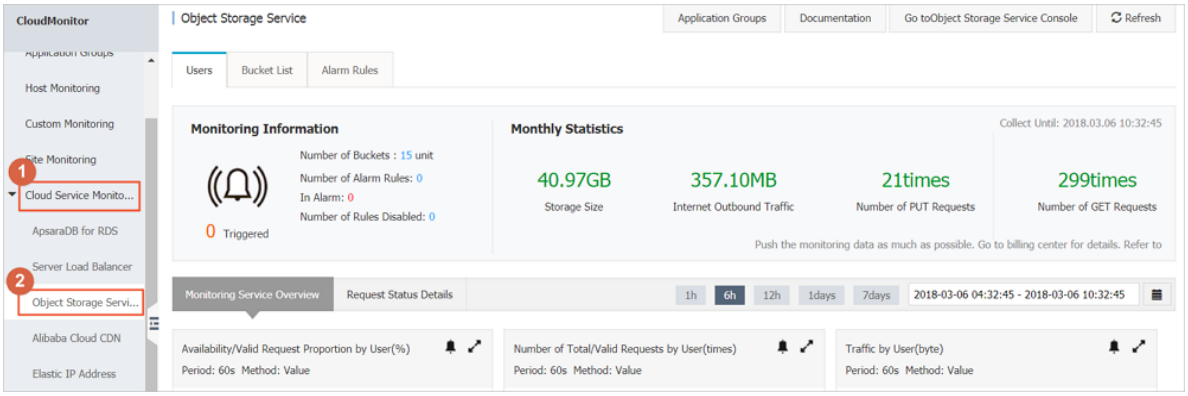
- Activate the *OSS service*.
- Activate the *CloudMonitor service*.

Monitor OSS resources

1. Log on to the *CloudMonitor console*.
2. Select **Cloud Service Monitoring** > **Object Storage Service** from the left-side navigation pane to enter the OSS monitoring page, as shown in the following figure.

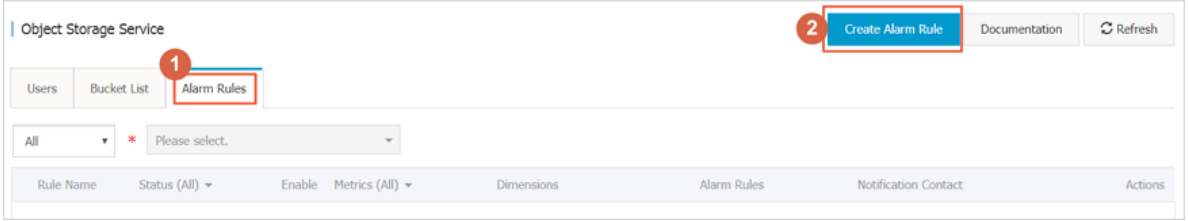
   You can obtain monitoring data on the OSS monitoring page.

> **Note:**
> "by User" refers to user-level data, that is, all bucket data of this user.



Set alarm rules

1. Find the **Alarm Rules** tab on OSS monitoring page, and then click **Create Alarm Rule**.



2. Configure your alarm rules.

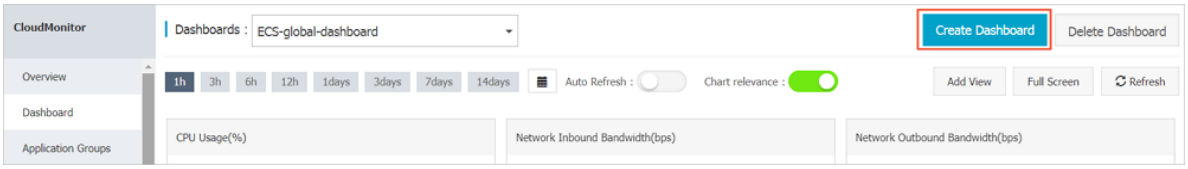
   For configuration details, see *Manage alarm rules*.

3. The alarm rule is generated when the configuration is completed. You can use test data to check whether the rule has taken effect by verifying if the alarm information was received successfully (over email, SMS, Trademanager, or DingTalk).

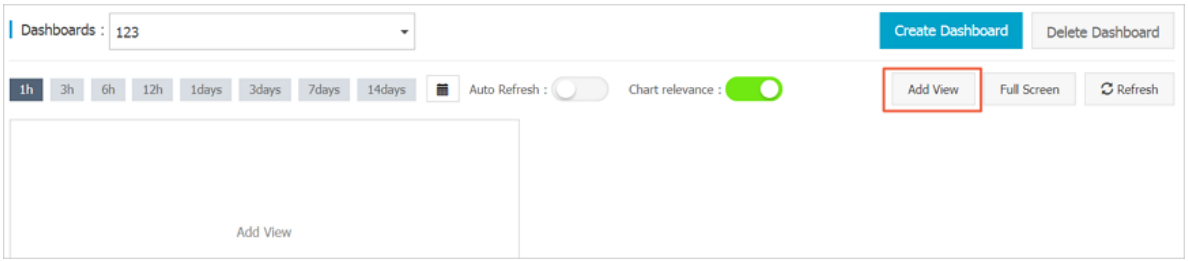
Custom monitoring dashboard

You can customize the OSS resource monitoring map on the CloudMonitor Console. The procedure is as follows.

1. Log on to the *CloudMonitor console*.
2. Click **Dashboard** from the left-side navigation pane.
3. Click **Create Dashboard**.



4. Enter the name of dashboard, and then click **Add View**.



5. Configure tables as required, and then click **Save**.

   For configuration details, see *Monitoring indicators reference*.

7 OSS performance and scalability best practice

Partitions and naming conventions

OSS automatically partitions user data by file names encoded in UTF-8 to process massive data and meet the needs for high request rates. However, if you use sequential prefixes (such as timestamps and sequential numbers) as part of the names when uploading a large number of objects, there may be lots of file indexes stored in a single partition. In this way, when the request rates exceed 2,000 operations per second (downloading, uploading, deleting, copying, and obtaining metadata are each counted as one operation, while deleting or enumerating more than one files in batch is considered as multiple operations), the following results may occur:

- This partition becomes a hotspot partition, leading to the exhausted I/O capacity and low request rate limited automatically by the system.
- With a hotspot partition, the partitioned data is constantly rebalanced, which may increase the processing time.

Therefore, the horizontal scaling capability of OSS is affected, thus resulting in limited request rate .

To address these issues, you must delete the sequential prefixes in the file names. Instead, you can add random prefix in file names. In this way, the file indexes (and I/O loads) are evenly distributed in different partitions.

The following shows the examples of changing sequential prefixes into random prefixes.

- Example 1: Add hex hash prefixes into file names

  As shown in this example, you may use a combination of dates and customer IDs (including sequential timestamp prefixes) in file names:

```
sample-bucket-01/2017-11-11/customer-1/file1
sample-bucket-01/2017-11-11/customer-2/file2
sample-bucket-01/2017-11-11/customer-3/file3
...
sample-bucket-01/2017-11-12/customer-2/file4
sample-bucket-01/2017-11-12/customer-5/file5
sample-bucket-01/2017-11-12/customer-7/file6
```

```
...
```

In this case, you can calculate a hash value for the customer ID, that is, the MD5 (customer-id), and combine a hash prefix of several characters as the prefix to the file name. If you use a 4-character hash prefix, the file names are as follows:

```
sample-bucket-01/2c99/2017-11-11/customer-1/file1
sample-bucket-01/7a01/2017-11-11/customer-2/file2
sample-bucket-01/1dbd/2017-11-11/customer-3/file3
...
sample-bucket-01/7a01/2017-11-12/customer-2/file4
sample-bucket-01/b1fc/2017-11-12/customer-5/file5
sample-bucket-01/2bb7/2017-11-12/customer-7/file6
...
```

In this case, a 4-character hex hash value is used as the prefix, and each character can be any one of the 16 values (0-f), so there are 16^4=65,536 possible character combinations. Technically, the data in the storage system is constantly partitioned into up to 65,536 partitions . Leveraging the performance bottleneck limit (2,000 operations per second) and the request rate of your service, you can determine a proper number of hash buckets.

If you want to list all the files with a specific date in the file name, for example, files with 2017 -11-11 in the name in sample-bucket-01, you must enumerate the files in sample-bucket-01 (acquire all files in sample-bucket-01 in batch by multiple calls of the List Object API) and combine files with this date in the file names.

- Example 2: Reverse the file name

  In this example, you may use a UNIX timestamp with millisecond precision to generate file names, which is also a sequential prefix:

```
sample-bucket-02/1513160001245.log
sample-bucket-02/1513160001722.log
sample-bucket-02/1513160001836.log
sample-bucket-02/1513160001956.log
...
sample-bucket-02/1513160002153.log
sample-bucket-02/1513160002556.log
sample-bucket-02/1513160002859.log
...
```

  As mentioned in the preceding paragraph, if you use the sequential prefix in file names, the performance may be affected when the request rate exceeds a certain limit. To address this issue, you can reverse the timestamp prefix to exclude the sequential prefix. The result is as follows:

```
sample-bucket-02/5421000613151.log
sample-bucket-02/2271000613151.log
```

```
sample-bucket-02/6381000613151.log
sample-bucket-02/6591000613151.log
...
sample-bucket-02/3512000613151.log
sample-bucket-02/6552000613151.log
sample-bucket-02/9582000613151.log
...
```

The first three digits of the file name represent the millisecond, which can be any one of the 1 ,000 values. The forth digit changes every second. Similarly, the fifth digit changes every 10 seconds. In this way, the prefixes are randomly specified and the loads are distributed evenly to multiple partitions, thus avoiding the performance bottleneck.

8 OssDemo for Android

8.1 OssDemo introduction

The OSS Developer Guide has introduced *Mobile Terminal Development and Uploading Scenario*. Taking this scenario as an example, how to use SDK to perform some common operations on Android, that is the OssDemo is introduced in the following document. It includes:

- How to use the application server (STS) which has been set up
- How to use SDK to upload a file
- How to use the image service

Here, we assume you have certain knowledge about OSS mobile development scenarios and STS (Security Token Service).

Preparation

Since the development is based on Android, you are required to be equipped with the following:

1. Activate the OSS. For more information, see *Quick Start*.
2. Set up an application server. For more information, see *Setting up Direct Data Transfer for Mobile Apps*.
3. Prepare the Android development environment. Android Studio is used in this scenario. Here, we do not mention the steps or procedure involved in using the Android Studio. This is because , instructions to use the Andriod Studio are easily available on the Internet.
4. Download the *source code of OssDemo*. You can take a trial by yourself, after installation. For more information on how to implement the preceding common operations, see the source code analysis.
5. Open the *OSS Android SDK Documentation* provided by OSS for reference.

8.2 Use the setup application server

This document elaborates, how to use a mobile app like OssDemo to access the application server for the purpose of uploading data to OSS without the need of storing AccessKeyId and AccessKeySecret to the app.

Logic of calling

1. OssDemo sends a request to the address of the sts_server received by OssDemo.

2. sts_server returns AccessKeyId, AccessKeySecret, SecurityToken, and Expiration to OssDemo
   .
3. After receiving such information, OssDemo calls SDK and creates an OssClient instance.

Code

1. Generate an EditText control.

```
Location:
 res/layout/content_main.xml
 Content:
 <EditText
     android:layout_height="wrap_content"
     android:layout_width="0dp"
     android:layout_weight="4"
     android:id="@+id/sts_server"
     android:text="@string/sts_server"
     />
 Location:
 res/values/strings
 Content:
 <string name="sts_server">http://oss-demo.aliyuncs.com/app-server/
sts.php</string>
```

2. Get the codes of corresponding STS parameters from the application server.

   Function implementation:

```
OSSFederationToken getFederationToken()
```

3. Get the STS returned parameters and initialize the OssClient code.

   Function implementation:

```
//Initialize an OssService used for uploading and downloading.
 public OssService initOSS(String endpoint, String bucket,
ImageDisplayer displayer) {
     //If you want to directly use the accessKey for access purposes
, you can directly use OSSPlainTextAKSKCredentialProvider for
authentication.
     //OSSCredentialProvider credentialProvider = new OSSPlainTe
xtAKSKCredentialProvider(accessKeyId, accessKeySecret);
     //Use your own class to retrieve an STSToken
     OSSCredentialProvider credentialProvider = new STSGetter(
stsServer);
     ClientConfiguration conf = new ClientConfiguration();
     conf.setConnectionTimeout(15 * 1000); // Connection timeout, 15
 seconds by default
     conf.setSocketTimeout(15 * 1000); // Socket timeout, 15 seconds
 by default
     conf.setMaxConcurrentRequest(5); // Maximum concurrent requests
, 5 by default
     conf.setMaxErrorRetry(2); // Maximum error retries, 2 by
default
     OSS oss = new OSSClient(getApplicationContext(), endpoint,
credentialProvider, conf);
     return new OssService(oss, bucket, displayer);
```

```
}
```

8.3 Upload a file

Simple upload

Simple upload means the Put Object interface in the OSS API is called to upload the selected files to OSS on a one-off basis.

- Logic of calling

  1. Once you select the Upload option, you can select the files to be uploaded.
  2. Once the processing parameters are selected, OssDemo sends a request to the address of the sts_server received by OssDemo.
  3. sts_server returns AccessKeyId, AccessKeySecret, SecurityToken, and Expiration to OssDemo.
  4. Once you have received the preceding information, OssDemo calls SDK, creates an OssClient instance, and implements simple upload.

- Code

  1. Generate a button control.

```
Location:
res/layout/content_main.xml
Content:
<Button
    style="?android:attr/buttonStyleSmall"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/multipart_upload"
    android:id="@+id/multipart_upload" />
```

  2. Click “Upload” and select the files to be uploaded.

     Snippet of function implementation:

```
Button upload = (Button) findViewById(R.id.upload);
upload.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent(
                Intent.ACTION_PICK,
                android.provider.MediaStore.Images.Media.
EXTERNAL_CONTENT_URI);
        startActivityForResult(i, RESULT_UPLOAD_IMAGE);
    }
}
```

  3. Call the upload interface of the SDK.

Snippet of function implementation:

```
 @Override
 protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
     super.onActivityResult(requestCode, resultCode, data);
     if ((requestCode == RESULT_UPLOAD_IMAGE || requestCode ==
RESULT_PAUSEABLEUPLOAD_IMAGE) && resultCode == RESULT_OK && null !
= data) {
         Uri selectedImage = data.getData();
         String[] filePathColumn = { MediaStore.Images.Media.DATA
 };
         Cursor cursor = getContentResolver().query(selectedImage,
                 filePathColumn, null, null, null);
         cursor.moveToFirst();
         int columnIndex = cursor.getColumnIndex(filePathColumn[0
]);
         String picturePath = cursor.getString(columnIndex);
         Log.d("PickPicture", picturePath);
         cursor.close();
         try {
             Bitmap bm = ImageDisplayer.autoResizeFromLocalFile(
picturePath);
             displayImage(bm);
             File file = new File(picturePath);
             DisplayInfo("file: " + picturePath + "\nsize: " +
String.valueOf(file.length()));
         }
         catch (IOException e) {
             e.printStackTrace();
             displayInfo(e.toString());
         }
         //Perform simple upload or resumable upload based on the
specified operation.
         if (requestCode == RESULT_UPLOAD_IMAGE) {
             final EditText editText = (EditText) findViewById(R.
id.edit_text);
             String objectName = editText.getText().toString();
             //Call the simple upload interface to upload the
files.
             ossService.asyncPutImage(objectName, picturePath,
getPutCallback(), new ProgressCallbackFactory<PutObjectRequest>().
get());
         }
     }
 }
```

How to deal with the uploading results is not mentioned here. You may see onSuccess and onFailure in source code.

Resumable upload based on multipart upload

Call the Multipart Upload interface of the OSS API to achieve a resumable upload effect.

Logic of calling

1. Once you select the Upload option, you can select the files to be uploaded.

2. Once the processing parameters are selected, OssDemo sends a request to the address of the sts_server received by OssDemo.
3. sts_server returns AccessKeyId, AccessKeySecret, SecurityToken, and Expiration to OssDemo .
4. Once you have received the preceding information, OssDemo calls SDK, creates an OssClient instance, and implements a multipart upload.
5. If you have clicked **Pause** but the multipart upload process is still in progress, you can continue the upload remaining part, by clicking **Continue**. This can achieve the resumable upload effect.

Code

1. Generate a button control.

```
Location:
 res/layout/content_main.xml
 Content:
 <Button
     style="? android:attr/buttonStyleSmall"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/multipart_upload"
     android:id="@+id/multipart_upload" />
```

2. Click “Upload” and select the files to be uploaded.

   Snippet of function implementation:

```
Button multipart_upload = (Button) findViewById(R.id.multipart_
upload);
 multipart_upload.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
         //To make it simple, only one resumable upload task is
running.
         Intent i = new Intent(
                 Intent.ACTION_PICK,
                 android.provider.MediaStore.Images.Media.EXTERNAL_C
ONTENT_URI);
         startActivityForResult(i, RESULT_PAUSEABLEUPLOAD_IMAGE);
     }
 }
 );
```

3. Click “Upload” for resumable upload of the remaining part.

   Snippet of function implementation:

```
 Click "Upload":
 //The multipart upload interface of the SDK is called.
 task = ossService.asyncMultiPartUpload(objectName, picturePath,
getMultiPartCallback().addCallback(new Runnable() {
     @Override
     public void run() {
```

```
            pauseTaskStatus = TASK_NONE;
            multipart_resume.setEnabled(false);
            multipart_pause.setEnabled(false);
            task = null;
        }
    }}, new ProgressCallbackFactory<PauseableUploadRequest>().get());
    From the encapsulating logic for the SDK at the underlying layer,
we can see that resumable upload is implemented by asyncUpload in
the multiPartUploadManager.
    //During resumable upload, the returned task can be used to pause
the task.
    public PauseableUploadTask asyncMultiPartUpload(String object,
                                                    String localFile,
                                                    @NonNull final
OSSCompletedCallback<PauseableUploadRequest, PauseableUploadResult>
userCallback,
                                                    final OSSProgres
sCallback<PauseableUploadRequest> userProgressCallback) {
        if (object.equals("")) {
            Log.w("AsyncMultiPartUpload", "ObjectNull");
            return null;
        }
        File file = new File(localFile);
        if (!file.exists()) {
            Log.w("AsyncMultiPartUpload", "FileNotExist");
            Log.w("LocalFile", localFile);
            return null;
        }
        Log.d("MultiPartUpload", localFile);
        PauseableUploadTask task = multiPartUploadManager.asyncUpload(
object, localFile, userCallback, userProgressCallback);
        return task;
    }
```

8.4 Image processing

Image processing means that once the image is uploaded and displayed on the OssDemo, it is processed. The differences between image processing and downloading an image are that:

- The endpoint for processing an image is used.
- Some processing parameters are added under object.

Watermark an image

- Logic of calling

  1. Upload an image to OSS. By default, the bucket is sdk-demo, the object is test, and the endpoint of the OSS is `oss-cn-hangzhou.aliyuncs.com`.
  2. Based on the image processing method, processing parameters are added under test for the required effect.
  3. Once these processing parameters are selected, OssDemo sends a request to the address of the sts_server received by OssDemo.

4. sts_server returns AccessKeyId, AccessKeySecret, SecurityToken, and Expiration to OssDemo.
5. Once you have received all this information, OssDemo calls SDK, creates an OssClient instance, and downloads the image. The displayed effect is the effect produced after image processing. However, the endpoint for image service is `img-cn-hangzhou.aliyuncs.com`.

- Code

1. Click **More** and the page showing the processed image is displayed.
2. Add a watermark with a size of 100 at the lower-right corner of the previously uploaded image and get such an operating command.

   Snippet of function implementation:

```
In the ImageService class,
 A method is provided to add the parameters necessary for a
function to the object.
 //Add a text watermark to the image. All parameters other than
font size are default values. You can modify the parameter values
when necessary according to the image service documentation.
 public String textWatermark(String object, String text, int size)
 {
     String base64Text = Base64.encodeToString(text.getBytes(),
Base64.URL_SAFE | Base64.NO_WRAP);
     String queryString = "@watermark=2&type=" + font + "&text="
 + base64Text + "&size=" + String.valueOf(size);
     Log.d("TextWatermark", object);
     Log.d("Text", text);
     Log.d("QuerySyring", queryString);
     return (object + queryString);
 }
```

3. Call the SDK download interface to process the image.

   Snippet of function implementation:

```
 getImage(imageService.textWatermark(objectName, "OSS test", 100
), 0, "text watermark at bottom right corner, size: 100");
 public void getImage(final String object, final Integer index,
final String method) {
     GetObjectRequest get = new GetObjectRequest(bucket, object);
     Log.d("Object", object);
     OSSAsyncTask task = oss.asyncGetObejct(get, new UICallback<
GetObjectRequest, GetObjectResult>(uiDispatcher) {
         @Override
         public void onSuccess(GetObjectRequest request,
GetObjectResult result) {
             // Request succeeded
             InputStream inputStream = result.getObjectContent();
             Log.d("GetImage", object);
             Log.d("Index", String.valueOf(index));
             try {
                 //Do not exceed the maximum display number.
```

```
                    adapter.getImgMap().put(index, new ImageDisplayer
(1000, 1000).autoResizeFromStream(inputStream));
                    adapter.getTextMap().put(index, method + "\n" +
object);
                    //Perform auto scaling based on the size of the
corresponding view.
                    addCallback(new Runnable() {
                        @Override
                        public void run() {
                            adapter.notifyDataSetChanged();
                        }
                    }, null);
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
                super.onSuccess(request,result);
            }
```

How to deal with the failure in downloading the results is not mentioned in this document.

Scaling, cropping, and rotating an image

This is similar to the watermarking process. In the ImageService, add a function for getting the processing commands. Add the processing parameters under the object. Finally, call the Get Object interface of the SDK to process the image.

```
//Scaling
getImage(imageService.resize(objectName, 100, 100), 1, "scale to 100*
100");
//Cropping
getImage(imageService.crop(objectName, 100, 100, 9), 2, "crop the
lower-right corner by 100*100");
//Rotating
getImage(imageService.rotate(objectName, 90), 3, "rotate by 90 degree
");
```