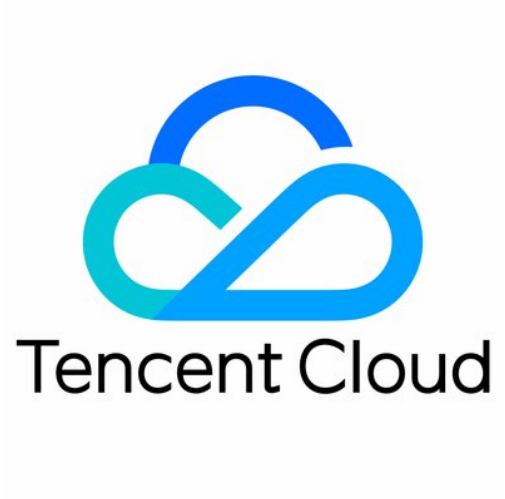
Serverless Cloud Function

Practical Operation of Code

Product Introduction



Contents

Practical Operation of Code Best Practice

Last updated：2018-08-28 18:03:01

Based on the features of Serverless Cloud Function (SCF), we recommend that you:

- Write function codes in a stateless style, to ensure that your codes are not maintained in any status. Use COS, Redis/Memcached and other services to cache intermediate information and implement the final computing results because the results of local storage and internal storage may be lost.
- Instantiate any objects that may be reused (such as database connections) other than execution methods.
- Configure +rx (read and execute) permission to your file in the uploaded ZIP to ensure successful execution of codes.
- Minimize the use of startup code that is not directly related to the processing of current event to reduce cost and improve performance if users want to minimize the startup latency. In addition, since the underlying computing resources may be reused to some extent, users can execute the function regularly to use "warm start" for the subsequent calling operations if they want to minimize the startup latency.
- Maximize the use of log/print statements in codes to provide sufficient information for debugging.
- You can use external code management services (such as Git) for the purpose of version and audit management of core codes, to ensure the completeness of codes (version management feature will be available soon).

Using API Gateway to provide API service Example

Last updated：2018-08-28 16:04:29

In this tutorial, assuming that:

- You want to use SCFs to implement Web backend services, such as listing the articles in a blog and providing article contents.
- You want to use APIs to provide services for webpages and Apps.

Implementation Overview

The implementation process of the service is as follows:

- Create a function, configure API rules in the API gateway and direct the backend service to the function.
- A user sends a request that contains the article ID to the API.
- SCF queries the content corresponding to the ID according to the request parameters, and responds to the request in JSON format.
- The user performs subsequent processing after receiving the response in JSON format.

Note: By the time you finish this tutorial, your account will contain the following resources:

- A SCF triggered by the API gateway.
- An API service in the API gateway and related API rules.

This tutorial is divided into three parts:

- Complete the coding, creation, and testing of a function.
- Complete the design, creation and configuration of an API service and API rules.
- Test and verify the correctness and operation of the APIs through a browser or http request tool.

API Design

The design of APIs for current applications usually follows the Restful specification. Therefore in this example, we design the APIs for obtaining blog articles as follows:

- /article GET
  Return the article list

- /article/{articleId} GET
  Return the content of the article with the specified ID

Step 1. Create and Test blogArticle Function

Last updated：2018-09-03 15:26:36

In this section, you will create a function to achieve API response regarding blog articles, and test the function by calling it through the console.

Creating a blogArticle SCF

1. Log in to the Serverless Cloud Function Console. Select **Guangzhou** from the region list and click **Create**.

2. In the **Function configuration** section, enter `blogArticle` as the function name, leave all the other configuration options unchanged, and then click **Next**.

3. In the **Function code** section, enter `index.main_handler` as the execution method and paste the following codes into the code window, and then click **Next**.

```
# -*- coding: utf8 -*-
import json

testArticleInfo=[
{"id":1,"category":"blog","title":"hello world","content":"first blog! hello world!","time":"2017-12-05 13:45"},
{"id":2,"category":"blog","title":"record info","content":"record work and study!","time":"2017-12-06 08:22"},
{"id":3,"category":"python","title":"python study","content":"python study for 2.7","time":"2017-12-06 18:32"},
]

def main_handler(event,content):
if "requestContext" not in event.keys():
return json.dumps({"errorCode":410,"errorMsg":"event is not come from api gateway"})
if event["requestContext"]["path"] != "/article/{articleId}" and event["requestContext"]["path"] != "/article":
return json.dumps({"errorCode":411,"errorMsg":"request is not from setting api path"})
if event["requestContext"]["path"] == "/article" and event["requestContext"]["httpMethod"] == "GET":
#Obtain the article list
```

```
retList = []
for article in testArticleInfo:
retItem = {}
retItem["id"] = article["id"]
retItem["category"] = article["category"]
retItem["title"] = article["title"]
retItem["time"] = article["time"]
retList.append(retItem)
return json.dumps(retList)
if event["requestContext"]["path"] == "/article/{articleId}" and event["requestContext"]["httpMethod"
] == "GET": #Obtain the article content
articleId = int(event["pathParameters"]["articleId"])
for article in testArticleInfo:
if article["id"] == articleId:
return json.dumps(article)
return json.dumps({"errorCode":412,"errorMsg":"article is not found"})
return json.dumps({"errorCode":413,"errorMsg":"request is not correctly execute"})
```

1. In the **Triggering method** section, you do not need to add any trigger method because the API gateway trigger is configured in the API gateway. Click **Complete**.

Note

The data structure of articles can be saved and simulated using the variable testArticleInfo. Usually, data structures are read from databases or files.

Testing the blogArticle SCF

When a function is created, it is generally tested through the console or API, to ensure the function output meets the expectation, and then you can bind it to a trigger for practical application.

1. In the details page of the function you just created, click **Test**.

2. Select **API Gateway Test Template** from the test templates. Modify it as follows to test the API for obtaining the article list.

```
{
"requestContext": {
"serviceName": "testsvc",
"path": "/article",
"httpMethod": "GET",
```

```
"requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
"identity": {
"secretId": "abdcdxxxxxxxsdfs"
},
"sourceIp": "10.0.2.14",
"stage": "prod"
},
"headers": {
"Accept-Language": "en-US,en,cn",
"Accept": "text/html,application/xml,application/json",
"Host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqloud.com",
"User-Agent": "User Agent String"
},
"pathParameters": {
},
"queryStringParameters": {
},
"headerParameters":{
"Refer": "10.0.2.14"
},
"path": "/article",
"httpMethod": "GET"
}
```

In the code above, the `path` and `httpMethod` fields in `requestContext`, as well as the peripheral `path` and `httpMethod` fields are modified as `/article` (the API path we design) and `GET`.

1. Click **Run** to view the results. The running result should be successful, and the content returned should be the basic information of articles as shown below.

```
[{"category": "blog", "time": "2017-12-05 13:45", "id": 1, "title": "hello world"}, {"category": "blog", "tim
e": "2017-12-06 08:22", "id": 2, "title": "record info"}, {"category": "python", "time": "2017-12-06 18:32",
"id": 3, "title": "python study"}]
```

1. Modify the test template as follows to test the API for obtaining article contents.

```
{
"requestContext": {
"serviceName": "testsvc",
"path": "/article/{articleId}",
"httpMethod": "GET",
"requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
"identity": {
"secretId": "abdcdxxxxxxxsdfs"
```

```
},
"sourceIp": "10.0.2.14",
"stage": "prod"
},
"headers": {
"Accept-Language": "en-US,en,cn",
"Accept": "text/html,application/xml,application/json",
"Host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqloud.com",
"User-Agent": "User Agent String"
},
"pathParameters": {
"articleId":"1"
},
"queryStringParameters": {
},
"headerParameters":{
"Refer": "10.0.2.14"
},
"path": "/article/1",
"httpMethod": "GET"
}
```

The `path` and `httpMethod` fields in `requestContext` are modified as `/article/{articleId}` (the API path we design) and `GET`. The peripheral `path` and `httpMethod` fields are modified as `/article/1` (the actual request path) and `GET`. The `pathParameters` field should be `"articleId":"1"`, the parameter and the actual value extracted from the API gateway.

1. Click **Run** to view the results. The running result should be successful, and the content returned should be the detailed content of articles as shown below.

```
{"category": "blog", "content": "first blog! hello world!", "time": "2017-12-05 13:45", "id": 1, "title": "hell
o world"}
```

Step 2. Create and Test API Service

Last updated：2018-09-03 15:27:06

In this section, you will create a service in the API gateway and related API rules, connect the SCF created in Step 1, and test the correctness of the APIs through the console.

> Note:
> The API service and the function must be in the same region. In this tutorial, the API service is created in the region Guangzhou.

Creating an API Service and API Rules

1. Log in to the Tencent Cloud Console, and select **Internet Middleware** -> **API Gateway** from **Cloud Products**.

2. Click the **Services** tab and change the region to **Guangzhou**.

3. Click **New** to create an API service. Enter the service name `blogAPI` in the pop-up window and click **Submit** to complete the creation.

4. Enter the created service `blogAPI`, and select the **API Management** tab.

5. Click **New** to create an API with the path of `/article` and the request method of GET. To facilitate the test later, select **No Authentication** here. No parameter configuration needs to be made. Click **Next**.

6. Select **Cloud Function** for the backend type, and select `blogArticle` created in Step 1 as the function, and click **Complete**.

7. Click **New** in the **API Management** tab to create another API with the path of `/article/{articleId}` and the request method of GET. Select **No Authentication**, and enter the parameter `articleId` in the parameter configuration with Path as the parameter location, int as the parameter type, and 1 as the default value, and then click **Next**.

8. Select **Cloud Function** for the backend type, and select `blogArticle` created in Step 1 as the function, and click **Complete**.

Debugging API Rules

1. To debug the API `/article` created in the step 5 above, click **API Debugging**, send a request in the debugging page, and check whether the response body in the returned result is shown as follows:

[{"category": "blog", "time": "2017-12-05 13:45", "id": 1, "title": "hello world"}, {"category": "blog", "time": "2017-12-06 08:22", "id": 2, "title": "record info"}, {"category": "python", "time": "2017-12-06 18:32", "id": 3, "title": "python study"}]

2. To debug the API `/article/{articleId}` created in the step 7 above, click **API Debugging**, modify the request parameter value to 1 and send a request in the debugging page, and check whether the response body in the returned result is shown as follows:

{"category": "blog", "content": "first blog! hello world!", "time": "2017-12-05 13:45", "id": 1, "title": "hello world"}

3. You can also modify the value of the request parameter articleId in step 2 to other number and check the response content.

Step 3. Publish API Service and Test Online

Last updated：2018-09-03 15:27:20

If you complete Step 2: Create and Test the API Service, and the test results meet the expectation, you can publish this service and initiate requests from a browser to verify whether the APIs run normally.

API Service Publishing

1. In the list in the **Services** page in the API Gateway console, find the blogAPI service created in Step 2, and click **Publish** in **Operation**.

2. In the pop-up window for service publishing, select `Publish` for the publishing environment and enter `Publish API` in the comments, and click **Submit**.

API Online Verification

After published, the APIs can be accessed externally. Next, you can initiate requests from a browser to verify whether the APIs can respond correctly.

1. In the blogAPI service, click the **Environment Management** tab, and copy the access path of the `Publish` environment, such as, `service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release` .
   Note: Since the domain names of services are not the same, the domain name assigned to your service will be different from that in this document. Therefore, do not copy the address directly in this document.

2. Add the path of the created API rules after this path as follows:

```
service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release/article
service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release/article/1
service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release/article/2
```

3. Copy the new path in step 2, paste it to the browser and access it, and check whether the output is the same with that when the API is tested.

4. You can modify the article ID in the request and check the output to see whether the code can handle the wrong article ID correctly.

Now, you have learned how to implement services using SCF and provide services using APIs. You can add new features and API rules subsequently by modifying the code to enrich the application module.

Acquire Image on COS and Create a Thumbnail
Example

Last updated：2018-08-28 15:55:13

In this tutorial, assuming that:

- A user is going to upload a photo to a specific COS Bucket
- You need to create a thumbnail for every image uploaded by the user
- Save the created thumbnails in another COS Bucket

Notes:
1. Two COS Buckets are required. If you **use** the same bucket **as both source and** target buckets, **each** thumbnail uploaded **to** the **source** bucket may **trigger** another **object creation event** that will again **call** the **function**, which may cause an infinite loop.
2. The **function** must be **in** the same region **with COS** Bucket.

Implementation Overview

The implementation process of the function is as follows:

- Create the function and the event source mapping of COS Bucket
- A user uploads the object to the source bucket (object creation event) of COS.
- COS Bucket detects the object creation event.
- COS calls the function and passes the event data to the function in parameters, thus publishing the cos:ObjectCreated:* event to the function.
- The SCF platform receives the call request and runs the function.
- The function acquires the Bucket name and file name from the received event data, obtains the file from the source Bucket, and uses the graph database to create a thumbnail, and then saves it in the target Bucket.

Note: By the time you finish this tutorial, your account will contain the following resources:

- A SCF used to create thumbnails.
- Two COS Buckets: and (the COS Bucket name you specify. For example, if you use the Bucket named "example" as the source bucket, you will create "exampleresized" as the target Bucket)

- Notification configuration on the source Bucket: Bind SCF and COS Bucket to the notification configuration of the Bucket, and add a new option to identify the type of the event to be triggered by COS (file creation/deletion) and the name of the function to be called. For more information about COS notification features, please see API PutBucketNotification.

This tutorial is divided into two parts:

- Complete the steps required to create a function, and call the function manually using the sample COS event data. This is designed to verify whether the function works normally.
- Add notification configurations to the source Bucket to allow COS to call the function when it detects a file creation event.

Step 1. Prepare COS Bucket

Last updated：2018-09-03 15:27:41

Notes:
1. The source Bucket, target Bucket and function must reside in the same region. In this tutorial, South China (Guangzhou) region is used.
2. Two COS Buckets are required. If you **use** the same Bucket **as both source and** target buckets, **each** thumbnail uploaded **to source** bucket may **trigger function** again, therefore unnecessary recursion **is** generated.

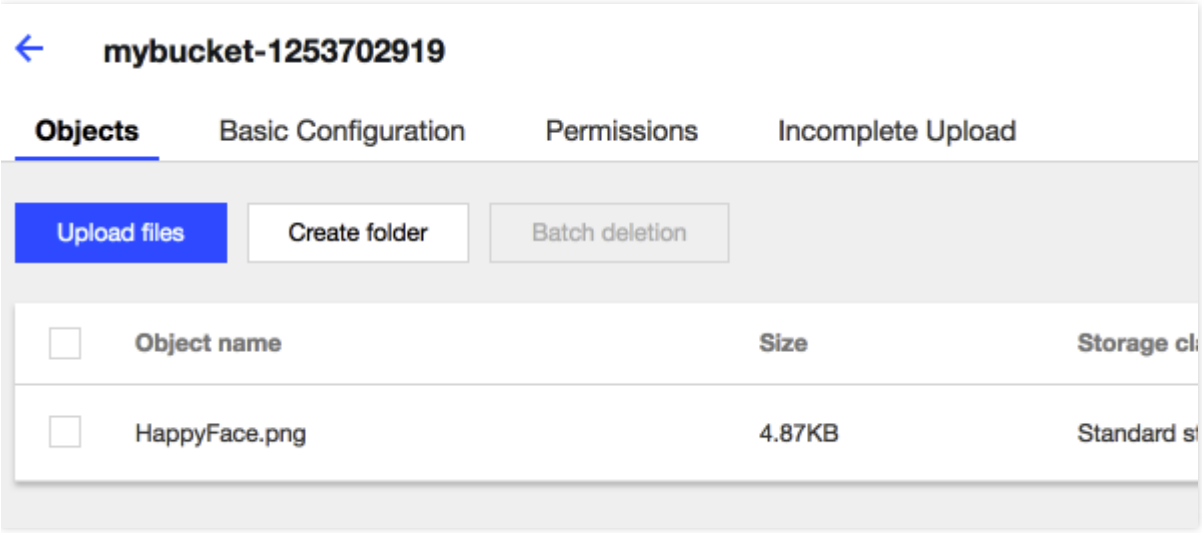
1) Log in to the Tencent Cloud console, and select **Cloud Object Storage**.

2) Click the **Create Bucket** button in the **Bucket List** tab to create a new source COS Bucket.

3) Configure the name of COS Bucket, such as `mybucket`, and set the region to `South China`, access permission to default `Public read & Private write`, CDN acceleration to default `Disabled`, and click **Save** to create a new COS Bucket.

4) Create the target Bucket `mybucketresized` in the same way.

5) Upload any image file to the source Bucket (i.e. mybucket). In this example, we use an image HappyFace.png for demonstration. (Before COS is associated, when manually calling the function to perform test and verification, you need to pass the sample data that contains this file to SCF, so that SCF can locate corresponding file according to this data. Therefore, you first need to create this sample object.)



Step 2. Create Deployment Package

Last updated：2018-09-03 15:27:54

> The version **of** image used **in** SCF underlying container environment **is** CentOS 7.2. Therefore, **in this** example, the Linux example **is** implemented under the CentOS 7.2 environment. Please make proper adjustments **if** your local environment **is** one **of** other released Linux versions.

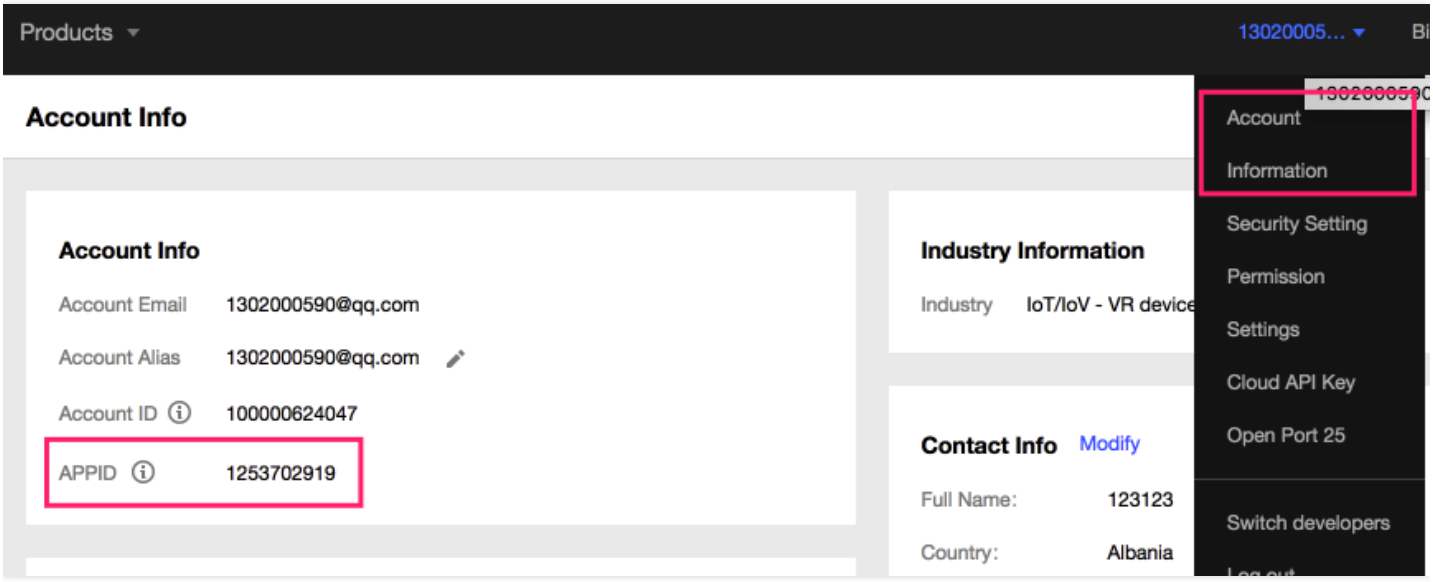
Creating Picture Processing Code

1) Create a directory
If the local environment is Windows, you can create a new folder named CreateThumbnail in any location.
If the local environment is Linux, you can create a new folder named CreateThumbnail in any location, as shown below:

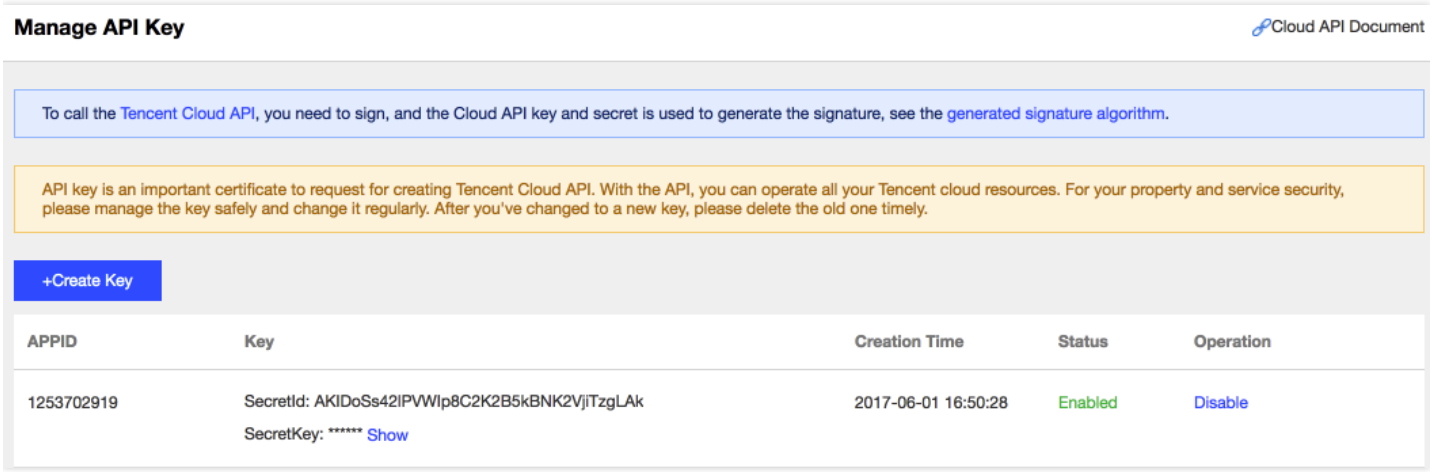
```
[root@[redacted]_centos /]# mkdir /CreateThumbnail
```

2) Open text editor and enter the following code.
Note: Replace `appid, secret_id, secret_key, and region` with your actual data, where:

- appid can be found in **Account Information** in the console.



- secret_id and secret_key can be obtained from **Cloud API Key** in the console.



- region is the region in which the function and COS Bucket reside. `sh, gz, and bj` are supported. Note: The region must be the same with that of COS Bucket created in the previous step. The bucket created in "Step 1: Prepare COS Bucket" resides in South China (Guangzhou), so the region value in the code must be `gz`.

```
import uuid
import json
import os
import logging
from PIL import Image
import PIL.Image
import commands
import datetime
import urllib
from qcloud_cos import CosClient
from qcloud_cos import DownloadFileRequest
from qcloud_cos import UploadFileRequest


print('Loading function')
appid = 1251762222 #please change to your appid. Find it in Account Info
secret_id = u'AKIDYDh085xQp48161uOn2CKKVbeebvDu6j2' #please change to your API secret id. Find it in API secret key pair
secret_key = u'lLkxx40kIfuyqW0lOl0WqyueCYjlgZQ2' #please change to your API secret key. Find it in API secret key pair
region = u'gz'

cos_client = CosClient(appid, secret_id, secret_key, region)
logger = logging.getLogger()
```

```
def resize_image(image_path, resized_path):
with Image.open(image_path) as image:
image.thumbnail(tuple(x / 2 for x in image.size))
image.save(resized_path)

def delete_local_file(src):
logger.info("delete files and folders")
if os.path.isfile(src):
try:
os.remove(src)
except:
pass
elif os.path.isdir(src):
for item in os.listdir(src):
itemsrc=os.path.join(src,item)
delete_file_folder(itemsrc)
try:
os.rmdir(src)
except:
pass

def main_handler(event, context):
logger.info("start main handler")
for record in event['Records']:
try:
bucket = record['cos']['cosBucket']['name']
cosobj = record['cos']['cosObject']['key']
cosobj = cosobj.replace("/"+str(appid)+"/"+bucket,"")
key = urllib.unquote_plus(cosobj.encode('utf8'))
download_path = '/tmp/{}{}'.format(uuid.uuid4(), key.strip('/'))
upload_path = '/tmp/resized-{}'.format(key.strip('/'))
print("Get from [%s] to download file [%s]" %(bucket,key))

# download image from cos
request = DownloadFileRequest(bucket, key, download_path)
download_file_ret = cos_client.download_file(request)
if download_file_ret['code'] == 0:
logger.info("Download file [%s] Success" % key)
logger.info("Image compress function start")
starttime = datetime.datetime.now()

#compress image here
resize_image(download_path, upload_path)
endtime = datetime.datetime.now()
logger.info("compress image take " + str((endtime-starttime).microseconds/1000) + "ms")
```

```
#upload the compressed image to resized bucket
request = UploadFileRequest(u'%sresized' % bucket, key.decode('utf-8'), upload_path.decode('utf-8'
))
upload_file_ret = cos_client.upload_file(request)
logger.info("upload image, return message: " + str(upload_file_ret))

#delete local file
delete_local_file(str(download_path))
delete_local_file(str(upload_path))
else:
logger.error("Download file [%s] Failed, err: %s" % (key, download_file_ret['message']))
return -1
except Exception as e:
print(e)
print('Error getting object {} from bucket {}. Make sure the object exists and your bucket is in the sam
e region as this function.'.format(key, bucket))
raise e
```

3) Save the file as `CreateThumbnail.py` in the directory you just created:

```
[root@VM_67_49_centos /]# cd /CreateThumbnail
[root@VM_67_49_centos CreateThumbnail]# vi CreateThumbnail.py
```

```
[root@VM_67_49_centos CreateThumbnail]# ls
CreateThumbnail.py
```

Creating a Deployment Package

For Windows environment

Because the sample program is dependent on Pillow dependent library of Python, to avoid the dependent library installed under your local Windows environment conflicts with that in the platform, we recommend that you:

Click the link to download Pillow Library directly, and decompress the zip file to the folder CreateThumbnail you just created.

Compress all the files in this folder into a zip file named CreateThumbnailDemo.zip (do not compress the folder): select all files, right-click on them, select a compress software (such as winrar), click **Add to**

Archive..., then select the archive format as zip, and click **OK** to generate a zip file named `CreateThumbnailDemo.zip` .

For Linux environment

> Note: Assuming that the following steps are performed under CentOS 7.2 environment, **if** your enviro nment **is** one **of** other released Linux versions, modify the instructions according to relevant method **of** the version, **and** ensure the Python version **is** 2.7.

1) Install Python environment.

```
sudo yum install python
```

2) Make sure that you have installed a necessary dependent library in the current Linux environment.

```
sudo yum install python-devel python-pip gcc libjpeg-devel zlib-devel python-virtualenv
```

3) Create and activate virtual environment.

```
virtualenv ~/shrink_venv
source ~/shrink_venv/bin/activate
```

4) Install Pillow library under the virtual environment.

```
pip install Pillow
```

5) Add the content related to lib and lib64 into a .zip file (the path is assumed to be `/CreateThumbnailDemo.zip`).

```
cd $VIRTUAL_ENV/lib/python2.7/site-packages
zip -r /CreateThumbnailDemo.zip *
cd $VIRTUAL_ENV/lib64/python2.7/site-packages
zip -r /CreateThumbnailDemo.zip *
```

6) Add the PY file created in the first step to this .zip file.

```
cd /CreateThumbnail
zip -g /CreateThumbnailDemo.zip CreateThumbnail.py
```

Step 3. Create CreateThumbnailDemo Function and Test

Last updated：2018-09-03 15:28:00

In this section, you will create a function to implement thumbnail program, and test the function through the console or by calling APIs.

Creating a CreateThumbnailDemo SCF

1) Log in to the Serverless Cloud Function Console. Select **Guangzhou** from the region list and click **Create**.

2) In **Function configuration** section, enter `CreateThumbnailDemo` as the function name, leave all other configuration options unchanged, and then click **Next**.

3) Go to the **Function code** section, and click **Local upload zip file**. Enter `CreateThumbnail.main_handler` as the execution method, select `CreateThumbnailDemo.zip` created in Step 2: Create Deployment Package, and click **Next**.

4) In the **Triggering method** section, you need to test the function manually, so no trigger method is added. Click **Done**.

Testing the CreateThumbnailDemo SCF

When a function is created, it is generally tested through the console or API, to ensure the function output meets the expectation, and then you can bind it to a trigger for practical application.

1) In the details page of the CreateThumbnailDemo function you just created, click **Test**.

2) Choose **Upload File to COS/Delete File from COS Test Code** from the drop-down list of test templates.

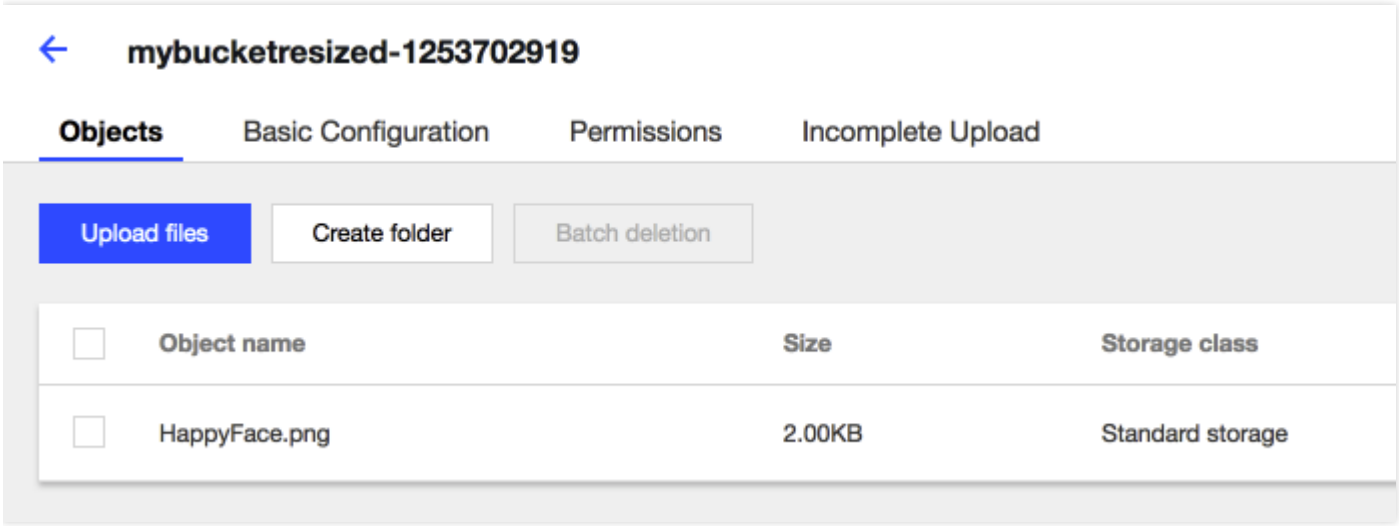
3) In the test code, set `name` to the name of bucket `mybucket` created in "Step 1: Prepare COS Bucket", and set `key` to the key value of `/HappyFace.jpg` uploaded in "Step 1: Prepare COS Bucket", as shown in the example below:

```
{
"Records":[
{
"event": {
```

```
"eventVersion":"1.0",
"eventSource":"qcs::cos",
"eventName":"event-type",
"eventTime":"Unix timestamp",
"eventQueue":"qcs:0:cos:gz:1251111111:cos",
"requestParameters":{
"requestSourceIP": "111.111.111.111",
"requestHeaders":{
"Authorization": "Uploaded authentication information"
}
}
},
"cos":{
"cosSchemaVersion":"1.0",
"cosNotificationId":"Configured or returned ID",
"cosBucket":{
"name":"mybucket", #set to demo bucket here
"appid":"appId",
"region":"gz"
},
"cosObject":{
"key":"/HappyFace.png", #set to demo file here
"size":"1024",
"meta":{
"Content-Type": "text/plain",
"x-cos-meta-test": "Custom meta",
"x-image-test": "Custom meta"
},
"url": "Origin server URL for accessing files"
}
}
}
]
}
```

4) Click **Run** to view the results. This program is running normally if both upload and download are successful in the result。

5) Go to the COS Console, and click `mybucketresized` created in "Step 1: Prepare COS Bucket", to check whether a thumbnail named `HappyFace.png` is generated.



6) Download the picture and compare it with the size of original picture.

Step 4. Add Trigger

Last updated：2018-09-03 15:28:05

If you complete "Step 3: Create CreateThumbnailDemo Function and Test", and the test result meets the expectation, you can add COS configurations so that COS can publish events to SCF and call the function.

1) In the details page of the CreateThumbnailDemo function you just created, select the **Triggering Method** tab, and click **Add trigger mode**.

2) Select COS trigger for the trigger method, `mybucket` created in Step 1: Prepare COS Bucket for COS Bucket, `File Upload` for the event type, and then click **Save**.

In this way, you have completed all steps. You can test the configuration by following the steps below:

1. Go to the COS Console, select `mybucket`, upload a .jpg or .png picture, and check whether a file with the same name exists in `mybucketresized` after a period of time.
2. You can monitor the function activities in the Serverless Cloud Function Console, and select **Logs** to check the logs in which the calling of the function is recorded.

MapReduce Method that Uses WordCount as an Example Example

Last updated：2018-08-28 15:49:18

In this tutorial, assuming that:

- You will upload some text files (such as logs) to a specific COS Bucket from time to time.
- You need to calculate the number of words in these text files.

Notes:
1. Two COS Buckets are required. If you **use** the same bucket **as both source and** target buckets, **each** thumbnail uploaded **to** the **source** bucket may **trigger** another **object creation event** that will again **call** the **function**, which may cause an infinite loop.
2. The **function** must be **in** the same region **with COS** Bucket.

Implementation Overview

The implementation process of the function is as follows:

- Create the function and the event source mapping of COS Bucket
- A user uploads the object to the source bucket (object creation event) of COS.
- COS Bucket detects the object creation event.
- COS calls the function and passes the event data to the function in parameters, thus publishing the cos:ObjectCreated:* event to the function.
- The SCF platform receives the call request and runs the function.
- The function acquires the Bucket name and the file name from the event data it received, obtains the file from the source Bucket, calculates the number of words using the wordcount implemented in the code, and saves it in the target Bucket.

Note: By the time you finish this tutorial, your account will contain the following resources:

- Two SCFs: Mapper and Reducer
- Three COS Buckets: srcmr, middlestagebucket and destmr
- Notification configuration on the source Bucket: Bind SCF and COS Bucket to the notification configuration of the Bucket, and add a new option to identify the type of the event to be triggered by

COS (file creation/deletion) and the name of the function to be called. For more information about COS notification features, please see the API PutBucketNotification.

This tutorial is divided into two parts:

- Complete the steps required to create a function, and call the function manually using the sample COS event data. This is designed to verify whether the function works normally.
- Add notification configurations to the source Bucket to allow COS to call the function when it detects a file creation event.

Step 1: Prepare a COS Bucket

Last updated：2018-09-03 15:28:43

Make sure you have obtained the permission to use SCF before implementing this example.

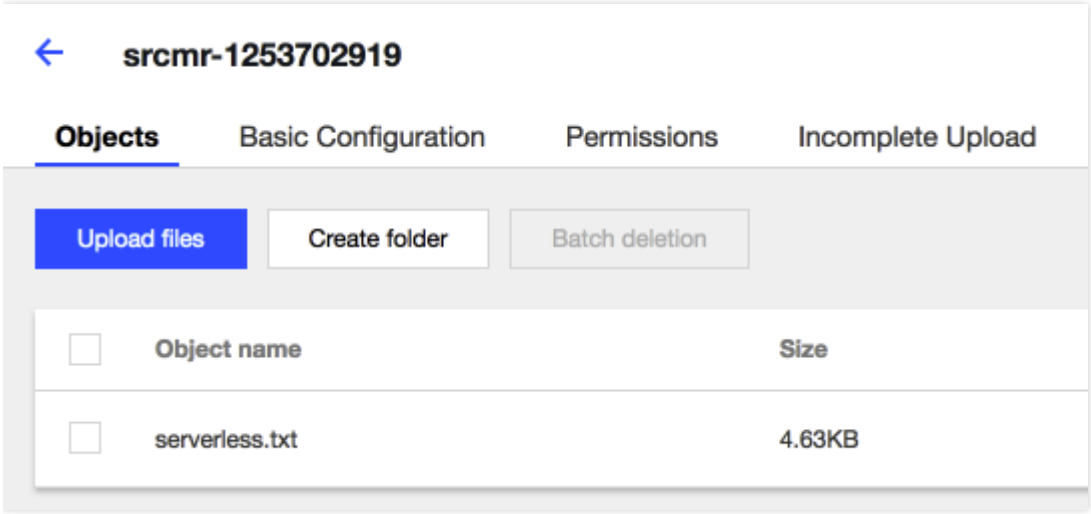
1) Log in to the Tencent Cloud console, and select **Cloud Object Storage**.

2) Click the **Create Bucket** button in the **Bucket List** tab to create a source COS Bucket.

3) Configure the name of COS Bucket, such as `srcmr`, and set the region to `South China`, access permission to default `Public read and private write`, and CDN acceleration to default `Disabled`, and click the **Save** button to create a COS Bucket.

4) Create the intermediate Bucket `middlestagebucket` and target Bucket `destmr` in the same way.

5) Upload a text file to the source Bucket (i.e. srcmr). In this example, we use a text file Serverless.txt for demonstration. (Before COS is associated, when manually calling the function to perform test and verification, you need to pass the sample data that contains this file to SCF, so that SCF can locate corresponding file according to this data. Therefore, you first need to create this sample file.)



Step 2. Create Deployment Package

Last updated：2018-09-03 15:28:48

Creating a Mapper Deployment Package

1) Create a new folder named WordCount in any location.

2) Create a new `.py` file named `map_function`, enter the following code in it and save the file. Note: Replace `appid, secret_id, secret_key, and region` with your actual data, where:

- appid can be found in **Account Information** in the console.
- secret_id and secret_key can be obtained from **Cloud API Key** in the console.
- region is the region in which the function and COS Bucket reside. `sh, gz, and bj` are supported. Note: The region must be the same with that of COS Bucket created in the previous step. The bucket created in "Step 1: Prepare COS Bucket" resides in South China (Guangzhou), so the region value in the code must be `gz`:

```
import mapper_triggered as Mapper
import datetime
from qcloud_cos import CosClient

def map_caller(event, context):
appid = 1251762222 # change to user's appid
secret_id = u'AKIDYDh085xQp48161uOn2CKKVbeebvDu6EE' # change to user's secret_id
secret_key = u'lLkxx40kIfuyqW0lOl0WqyueCYjlgZEE' # change to user's secret_key
region = u'gz' # change to user's region
cos_client = CosClient(appid, secret_id, secret_key, region)

bucket = event['Records'][0]['cos']['cosBucket']['name']
key = event['Records'][0]['cos']['cosObject']['key']
middle_stage_bucket = u'middlestagebucket'
middle_file_key = '/' + 'middle_' + key.split('/')[-1]

return Mapper.do_mapping(cos_client, bucket, key, middle_stage_bucket, middle_file_key)

def main_handler(event, context):
start_time = datetime.datetime.now()
res = map_caller(event, context)
end_time = datetime.datetime.now()
print("data mapping duration: " + str((end_time-start_time).microseconds/1000) + "ms")
if res == 0:
```

```
return "Data mapping SUCCESS"
else:
return "Data mapping FAILED"
```

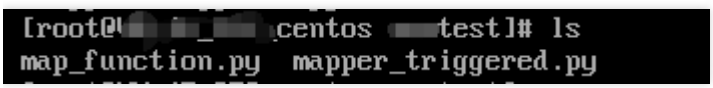
Upon creation, create a `.py` file named `mapper_triggered` under *the same path*, enter the following code in it and save the file:

```
from qcloud_cos import UploadFileRequest
from qcloud_cos import DownloadFileRequest
import re
import os
import logging


logger = logging.getLogger()

#delete folders and files
def delete_file_folder(src):
if os.path.isfile(src):
try:
os.remove(src)
except:
pass
elif os.path.isdir(src):
for item in os.listdir(src):
itemsrc=os.path.join(src,item)
delete_file_folder(itemsrc)
try:
os.rmdir(src)
except:
pass

# Download files
def download_file(cos_client, bucket, key, local_file_path):
request = DownloadFileRequest(bucket, key, local_file_path)
download_file_ret = cos_client.download_file(request)
if download_file_ret['code'] == 0:
logger.info("Download file [%s] Success" % key)
return 0
else:
logger.error("Download file [%s] Failed, err: %s" % (key, download_file_ret['message']))
return -1

# Upload file to bucket
def upload_file(cos_client, bucket, key, local_file_path):
```

```
request = UploadFileRequest(bucket.decode('utf-8'), key.decode('utf-8'), local_file_path.decode('utf-8'))
upload_file_ret = cos_client.upload_file(request)
if upload_file_ret['code'] == 0:
logger.info("Upload data map file [%s] Success" % key)
return 0
else:
logger.error("Upload data map file [%s] Failed, err: %s" % (key, upload_file_ret['message']))
return -1

# domapping
def do_mapping(cos_client, bucket, key, middle_stage_bucket, middle_file_key):
src_file_path = u'/tmp/' + key.split('/')[-1]
middle_file_path = u'/tmp/' + u'mapped_' + key.split('/')[-1]
download_ret = download_file(cos_client, bucket, key, src_file_path) #download src file
if download_ret == 0:
inputfile = open(src_file_path, 'r') #open local /tmp file
mapfile = open(middle_file_path, 'w') #open a new file write stream

for line in inputfile:
line = re.sub('[^a-zA-Z0-9]', ' ', line) #replace non-alphabetic/number characters
words = line.split()
for word in words:
mapfile.write('%s\t%s' % (word, 1)) #count for 1
mapfile.write('\n')

inputfile.close()
mapfile.close()

upload_ret = upload_file(cos_client, middle_stage_bucket, middle_file_key, middle_file_path) #upload the file's each word

delete_file_folder(src_file_path)
delete_file_folder(middle_file_path)
return upload_ret
else:
return -1
```

3) If the local environment is Windows, you can find two py files under this path.

If the local environment is Linux, you can find two py files under this path, as shown below:



Compress these two files into a zip file named mapper (note: you need to compress the files instead of the folder in which these files reside).

In Windows environment:
Select these two files, right-click on them, select a compress software (such as winrar), click **Add to Archive...**, then select the archive format as zip, and click **OK** to generate a zip file.

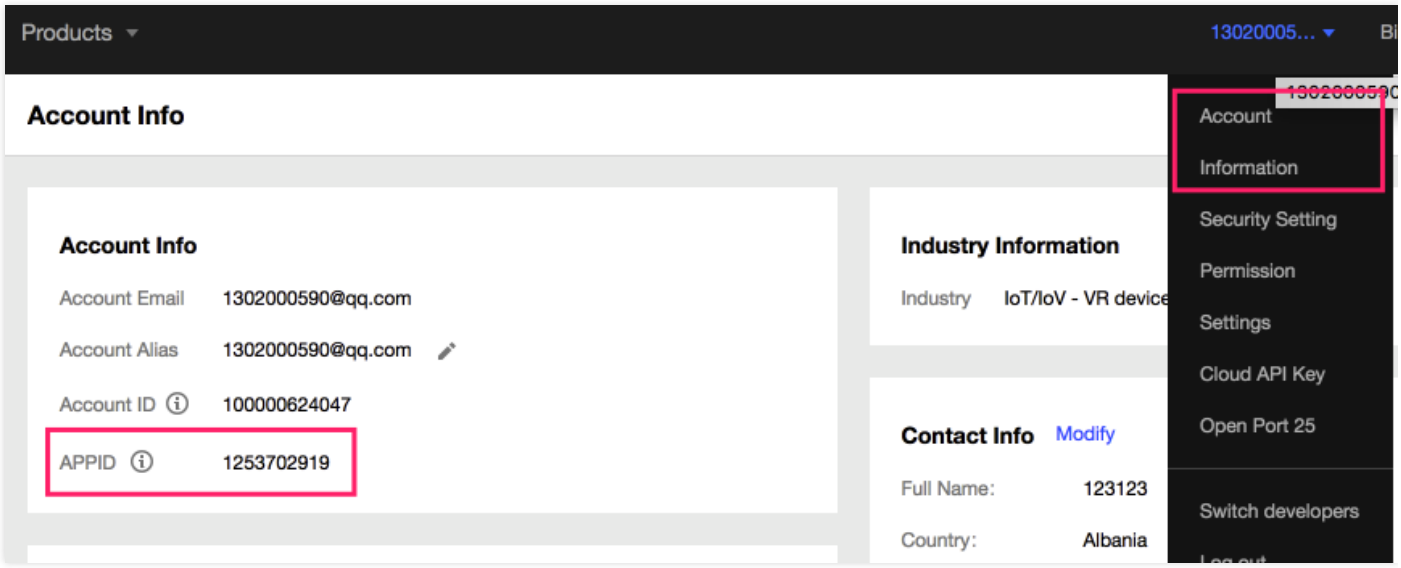
In Linux environment:
Enter the directory directly to run the command.

```
cd /WordCount
zip mapper.zip map_function.py mapper_triggered.py
```

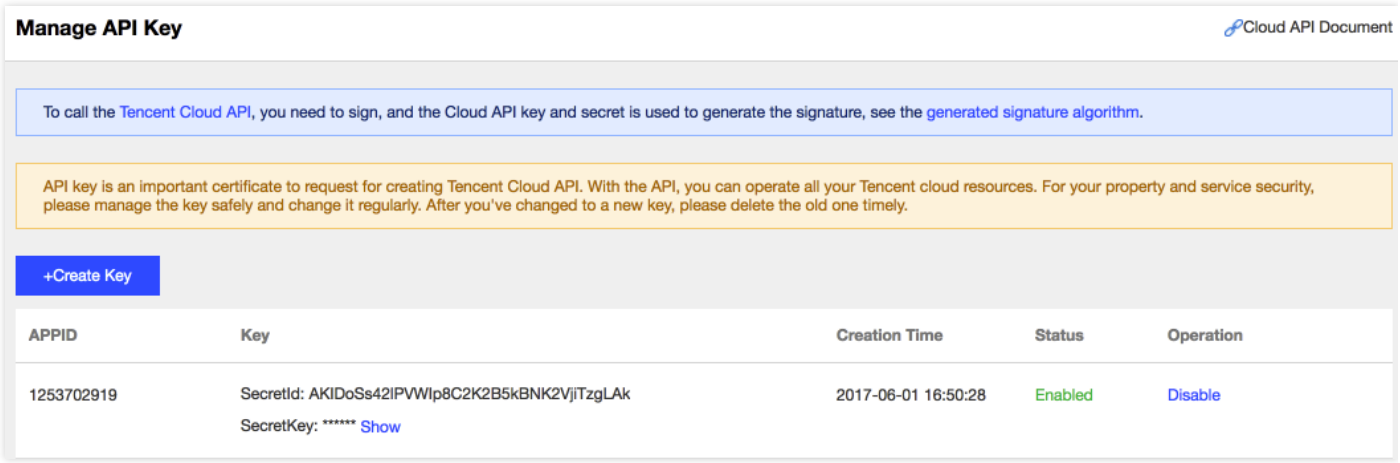
Creating a Reducer Deployment Package

1) Similarly, create a `.py` file named `reduce_function` under WordCount directory, enter the following code in it and save the file. Note: Replace `appid, secret_id, secret_key, and region` with your actual data, where:

- appid can be found in **Account Information** in the console.



- secret_id and secret_key can be obtained from **Cloud API Key** in the console.



- region is the region in which the function and COS Bucket reside. `sh, gz, and bj` are supported. Note: The region must be the same with that of COS Bucket created in the previous step. The bucket created in "Step 1: Prepare COS Bucket" resides in South China (Guangzhou), so the region value in the code must be `gz`:

```
import reducer_triggered as Reducer
import datetime
from qcloud_cos import CosClient

def reduce_caller(event, context):
appid = 1251762222 # change to user's appid
secret_id = u'AKIDYDh085xQp48161uOn2CKKVbeebvDu6EE' # change to user's secret_id
secret_key = u'lLkxx40kIfuyqW0IOI0WqyueCYjlgZEE' # change to user's secret_key
region = u'gz' # change to user's region
cos_client = CosClient(appid, secret_id, secret_key, region)

bucket = event['Records'][0]['cos']['cosBucket']['name']
key = event['Records'][0]['cos']['cosObject']['key']
result_bucket = u'destmr'
result_key = '/' + 'result_' + key.split('/')[-1]

return Reducer.qcloud_reducer(cos_client, bucket, key, result_bucket, result_key)

def main_handler(event, context):
start_time = datetime.datetime.now()
res = reduce_caller(event, context)
end_time = datetime.datetime.now()
print("data reducing duration: " + str((end_time-start_time).microseconds/1000) + "ms")
if res == 0:
```

```
return "Data reducing SUCCESS"
else:
return "Data reducing FAILED"
```

Upon creation, create a `.py` file named `reducer_triggered` under *the same path*, enter the following code in it and save the file:

```
from qcloud_cos import UploadFileRequest
from qcloud_cos import DownloadFileRequest
import os
import logging
from operator import itemgetter


logger = logging.getLogger()

#delete folders and files
def delete_file_folder(src):
if os.path.isfile(src):
try:
os.remove(src)
except:
pass
elif os.path.isdir(src):
for item in os.listdir(src):
itemsrc=os.path.join(src,item)
delete_file_folder(itemsrc)
try:
os.rmdir(src)
except:
pass

# Download files
def download_file(cos_client, bucket, key, local_file_path):
request = DownloadFileRequest(bucket, key, local_file_path)
download_file_ret = cos_client.download_file(request)
if download_file_ret['code'] == 0:
logger.info("Download file [%s] Success" % key)
return 0
else:
logger.error("Download file [%s] Failed, err: %s" % (key, download_file_ret['message']))
return -1

# Upload file to bucket
def upload_file(cos_client, bucket, key, local_file_path):
```

```
request = UploadFileRequest(bucket.decode('utf-8'), key.decode('utf-8'), local_file_path.decode('utf-8'))
upload_file_ret = cos_client.upload_file(request)
if upload_file_ret['code'] == 0:
logger.info("Upload data map file [%s] Success" % key)
return 0
else:
logger.error("Upload data map file [%s] Failed, err: %s" % (key, upload_file_ret['message']))
return -1

# doreducing
def qcloud_reducer(cos_client, bucket, key, result_bucket, result_key):
word2count = {}
src_file_path = u'/tmp/' + key.split('/')[-1]
result_file_path = u'/tmp/' + u'result_' + key.split('/')[-1]
download_ret = download_file(cos_client, bucket, key, src_file_path)
if download_ret == 0:
map_file = open(src_file_path,'r')
result_file = open(result_file_path,'w')

for line in map_file:
line = line.strip()
word, count = line.split('\t', 1)
try:
count = int(count)
word2count[word] = word2count.get(word, 0) + count
except ValueError:
logger.error("error value: %s, current line: %s" % (ValueError, line))
continue
map_file.close()
delete_file_folder(src_file_path)

sorted_word2count = sorted(word2count.items(), key=itemgetter(1))[::-1]
for wordcount in sorted_word2count:
res = '%s\t%s' % (wordcount[0], wordcount[1])
result_file.write(res)
result_file.write('\n')
result_file.close()

upload_ret = upload_file(cos_client, result_bucket, result_key, result_file_path)
delete_file_folder(result_file_path)
return upload_ret
```

3) Compress these two files into a zip file named reducer according to the above method (note: directly compress the files instead of the folder in which these files reside).

Step 3. Create Mapper and Reducer Functions and Test

Last updated：2018-09-03 15:29:27

In this section, you will create two functions to implement simple WordCount, and test the functions through the console or by calling API.

Creating a Mapper Function

Creating a function through the console

1) Log in to the Serverless Cloud Function Console. Select **Guangzhou** from the region list and click **Create**.

2) In the **Function configuration** section, enter `Mapper` as the function name and keep default settings for the other options, and then click **Next**.

3) Go to the **Function code** section, and click **Local upload zip file**. Enter `map_function.main_handler` for the execution method, select the `mapper.zip` created in "Step 2: Create Deployment Package", and click **Next**.

4) In the **Triggering method** section, you need to test the function manually, so no trigger method is added. Click **Done**.

Creating a function using API

For more information, please see API CreateFunction.

Creating a Reducer Function

Creating a function through the console

1) Log in to the Serverless Cloud Function Console. Select **Guangzhou** from the region list and click **Create**.

2) In the **Function configuration** section, enter `Reducer` as the function name, leave all the other configuration options unchanged, and then click **Next**.

3) Go to the **Function code** section, and click **Local upload zip file**. Enter `reduce_function.main_handler` for the execution method, select the `reducer.zip` created in "Step 2: Create Deployment Package", and

click **Next**.

4) In the trigger method section, you need to test the function manually, so no trigger method is added. Click **Done**.

Creating a function using API

For more information, please see API CreateFunction.

Testing Functions

When a function is created, it is generally tested through the console or API, to ensure the function output meets the expectation, and then you can bind it to a trigger for practical application.

Testing the function through the console

1) In the details page of the Mapper function you just created, click **Test**.

2) Choose **Upload File to COS/Delete File from COS Test Code** from the drop-down list of test templates.

3) In the test code, set `name` to the name of bucket `srcmr` created in "Step 1: Prepare COS Bucket", and set `key` to the key value of `/serverless.txt` uploaded in "Step 1: Prepare COS Bucket", as shown in the example below:

```
{
"Records":[
{
"event": {
"eventVersion":"1.0",
"eventSource":"qcs::cos",
"eventName":"event-type",
"eventTime":"Unix timestamp",
"eventQueue":"qcs:0:cos:gz:1251111111:cos",
"requestParameters":{
"requestSourceIP": "111.111.111.111",
"requestHeaders":{
"Authorization": "Uploaded authentication information"
}
}
},
"cos":{
"cosSchemaVersion":"1.0",
"cosNotificationId":"Configured or returned ID",
"cosBucket":{
```

```
"name":"srcmr", #set to demo bucket here
"appid":"appId",
"region":"gz"
},
"cosObject":{
"key":"/serverless.txt", #set to demo file here
"size":"1024",
"meta":{
"Content-Type": "text/plain",
"x-cos-meta-test": "Custom meta",
"x-image-test": "Custom meta"
},
"url": "Origin server URL for accessing files"
}
}
}
]
}
```

4) Click **Run** to view the results.

5) Go to the COS Console, and click `destmr` created in "Step 1: Prepare COS Bucket", to check whether a file named `result_middle_serverless.txt` is generated in COS Bucket. The file collects the number of times each word appears in the uploaded text file:

6) Download this file, and you can see the content similar to the following:

```
the 29
as 25
to 20
of 17
and 16
a 16
code 16
in 15
be 14
or 12
serverless 10
is 10
that 10
computing 9
by 9
for 9
an 8
not 8
```

cloud 7
functions 6
edit 6
can 6
with 5
servers 5
on 5
because 5
at 5
Serverless 5
have 5
platform 5
such 5
time 4
virtual 4
up 4
function 4
provider 4
autoscaling 4
used 4
it 4
js 4
are 4
also 4
than 4
service 4
written 4
2016 4
1 4
runtime 4
example 4
use 4
Node 4
run 3
does 3
Java 3
execution 3
other 3
model 3
In 3
requests 3
means 3
typically 3
which 3
latency 3

any 3
could 3
This 3
may 3
via 3
resources 3
required 2
10 2
developers 2
application 2
underutilisation 2
hosted 2
was 2
start 2
support 2
microservices 2
per 2
back 2
server 2
source 2
generally 2
It 2
well 2
user 2
running 2
Python 2
handling 2
JSON 2
addition 2
limits 2
both 2
available 2
For 2
request 2
The 2
efficient 2
part 2
2 2
system 2
APIs 2
first 2
completely 2
supports 2
container 2
programmer 2

API 2
name 2
provision 2
now 2
they 2
3 2
its 2
operations 2
machine 2
end 2
more 2
from 2
public 2
officially 2
even 2
cost 2
Functions 2
all 2
requires 1
At 1
2006 1
starting 1
serialized 1
calls 1
assets 1
InterConnect 1
exposure 1
needed 1
included 1
suited 1
includes 1
you 1
5 1
setting 1
when 1
no 1
periods 1
abstract 1
defined 1
called 1
continuously 1
simplifying 1
Infrequently 1
but 1
triggered 1

significantly 1
registration 1
down 1
functional 1
into 1
own 1
software 1
automatically 1
introduced 1
about 1
although 1
processing 1
business 1
However 1
latencies 1
some 1
creation 1
exposed 1
spins 1
either 1
specific 1
necessary 1
using 1
rules 1
usable 1
has 1
Despite 1
4 1
web 1
world 1
being 1
just 1
spend 1
without 1
person 1
task 1
provisioned 1
uses 1
Resource 1
meets 1
development 1
composition 1
units 1
make 1
pay 1

if 1
worry 1
technology 1
launched 1
stopping 1
owns 1
policies 1
will 1
batch 1
unlike 1
case 1
demand 1
ensuring 1
bulk 1
containers 1
offers 1
been 1
billing 1
tuning 1
GitHub 1
sequences 1
simply 1
simple 1
point 1
consume 1
Monitoring 1
s 1
sensitive 1
triggers 1
essentially 1
packages 1
premise 1
Programming 1
initially 1
terms 1
C 1
invoker 1
behind 1
capacity 1
Performance 1
configured 1
wrote 1
where 1
fixed 1
architecture 1

imposed 1
Swift 1
resource 1
infrequent 1
conjunction 1
cheaper 1
do 1
8 1
black 1
only 1
outside 1
renting 1
quantity 1
Cloud 1
Cost 1
traditional 1
considered 1
alpha 1
significant 1
performance 1
online 1
purchasing 1
responsible 1
offering 1
option 1
need 1
rent 1
debugging 1
jobs 1
PaaS 1
11 1
specialists 1
released 1
serve 1
Zimki 1
structures 1
their 1
management 1
given 1
open 1
directly 1
instances 1
number 1
features 1
major 1

launch 1
language 1
multithreading 1
announced 1
7 1
since 1
measure 1
another 1
events 1
fully 1
9 1
introduce 1
Alternatively 1
greater 1
affected 1
non 1
might 1
2014 1
systems 1
production 1
actually 1
group 1
style 1
suffer 1
including 1
order 1
allow 1
Haskell 1
purchase 1
response 1
data 1
designed 1
produce 1
REST 1
provisioning 1
rather 1
involve 1
providers 1
dedicated 1
believed 1
this 1
hour 1
endpoint 1
high 1
billed 1

known 1
expose 1
sharing 1
6 1
Microsoft 1
would 1
environment 1
private 1
followed 1
FaaS 1
involves 1
amount 1
deserialized 1
box 1
groups 1
supporting 1
satisfy 1
new 1
version 1
Disadvantages 1
machines 1
likely 1
cron 1
HTTP 1
workloads 1
small 1
level 1
announcing 1
include 1
languages 1
Docker 1
go 1
manages 1

Step 4. Add Trigger

Last updated：2018-09-03 15:29:32

If you complete "Step 3: Create Mapper and Reducer Functions and Test", and the test result meets the expectation, you can add COS configuration so that COS can publish event to SCF and call the function.

1) In the details page of Mapper function you just created, select "Trigger Method" tab, and click "Add Trigger Method" button.

2) Select COS trigger for the trigger method, `srcmr` created in "Step 1: Prepare COS Bucket" for COS Bucket, `File Upload` for the event type, and click "Save" button.

Now you have completely implemented this example. You can test the configuration by following the steps below:

1. Go to COS Console, select `src` , upload any .txt text file, and check whether a similar file generated in `destmr` after a certain period of time.
2. You can monitor the function activities in SCF Console, and select "Log" to check the logs in which the calling of the function is recorded.

Send Email According to the Messages in CMQ
Example

Last updated：2018-08-29 17:09:24

In this tutorial, assuming that:

- The system needs to send an email in a specific case.
- You want to use CMQ to collect and send necessary information and specify the recipient.

Implementation Overview

The implementation process of the function is as follows:

- Create the function and the event source mapping of CMQ Topic.
- A user sends the required message contents and the recipient to CMQ in a specific data format.
- CMQ calls the SCF and passes the message to the function in an event.
- The SCF platform receives the call request and runs the function.
- The function acquires the message contents and the recipient from the event data, and calls the sendEmail API to send an email.

Note: By the time you finish this tutorial, your account will contain the following resources:

- An SCF to send emails.
- A CMQ Topic.
- Subscription configuration in the CMQ Topic.

This tutorial is divided into three parts:

- Create a CMQ Topic.
- Complete the steps required to create a function, and call the function manually using the sample CMQ event data. This is designed to verify whether the function works normally.
- Bind the CMQ Topic and the function and test the linkage of the CMQ and the SCF using the sendEmail API, so that the CMQ can call the function when receiving a message.

Data Structure Design

We assume that the data structure used to send an email is shown below. This data structure will be sent to the CMQ after you enter required values as needed and SCF will receive and process it to send the email.

```
{
"fromAddr":"sender@testhost.com",
"toAddr":"test@testhost.com",
"title":"hello from scf & cmq",
"body":"email content to send"
}
```

Step 1. Create CMQ Topic Mode Queue

Last updated：2018-09-03 15:29:39

Note:
CMQ and the function must be in the same region. In this tutorial, South China (Guangzhou) region is used.

1. Log in to the Tencent Cloud Console, select **Messaging Service CMQ** in Cloud Products.

2. Click the **Subscribe Topic** tab and select the **South China (Guangzhou)** region.

3. Click the **+Create** button to create a queue and enter `sendEmailQueue` as the topic name in the pop-up window.

4. Click **Create**.

Step 2. Create and Test sendEmial Function

Last updated：2018-09-03 15:29:44

In this section, you will create a function to implement the sendEmail program, and test the function through the console or by calling APIs.

Creating a sendEmail SCF

1. Log in to the Serverless Cloud Function Console. Select **Guangzhou** from the region list and click **Create**.

2. In the **Function configuration** section, enter `sendEmail` as the function name and keep default settings for the other options, and then click **Next**.

3. In the **Function code** section, enter `index.main_handler` as the execution method and paste the following codes into the code window, and then click **Next**.

```
# -*- coding: utf8 -*-
import json
import smtplib
from email.mime.text import MIMEText
from email.header import Header

#Third-party SMTP service
mail_host="smtp.qq.com" #SMTP server
mail_user="3473058547@qq.com" #User name
mail_pass="xxxxxxx" #Password
mail_port=465 # SMTP service port

def sendEmail(fromAddr,toAddr,subject,content):
sender = fromAddr
receivers = [toAddr] # To receive emails. You can set it as your QQ mailbox or other mailbox.

message = MIMEText(content, 'plain', 'utf-8')
message['From'] = Header(fromAddr, 'utf-8')
message['To'] = Header(toAddr, 'utf-8')
message['Subject'] = Header(subject, 'utf-8')
```

```
try:
smtpObj = smtplib.SMTP_SSL(mail_host, mail_port)
smtpObj.login(mail_user,mail_pass)
smtpObj.sendmail(sender, receivers, message.as_string())
print("send success")
except smtplib.SMTPException as e:
print(e)
print("Error: send fail")

def main_handler(event, context):
cmqMsg = None
if event is not None and "Records" in event.keys():
if len(event["Records"]) >= 1 and "CMQ" in event["Records"][0].keys():
cmqMsgStr = event["Records"][0]["CMQ"]["msgBody"]
cmqMsg = json.loads(cmqMsgStr)
print cmqMsg
sendEmail(cmqMsg['fromAddr'], cmqMsg['toAddr'], cmqMsg['title'], cmqMsg['body'])
return "send email success"
```

1. In the **Triggering method** section, you need to test the function manually, so no trigger method is added currently. Click **Complete**.

Note

You must configure the `mail_host, mail_user, mail_pass, and mail_port` parameters based on the mailbox or mail server you use to receive emails. For example, you can view here to learn about how to enable the SMTP feature of QQ Mail. After it is enabled, the relevant parameters are as follows:

- mail_host: SMTP server "smtp.qq.com"
- mail_user: Your email address as the login user name, such as `3473058547@qq.com`
- mail_pass: The password you specified when you enable the SMTP feature.
- mail_port: The server login port. Since you can only log in to your QQ mailbox using SSL, the port must always be 465. Use `smtplib.SMTP_SSL` in codes to build SSL SMTP connection.

Testing the sendEmail SCF

When a function is created, it is generally tested through the console or API, to ensure the function output meets the expectation, and then you can bind it to a trigger for practical application.

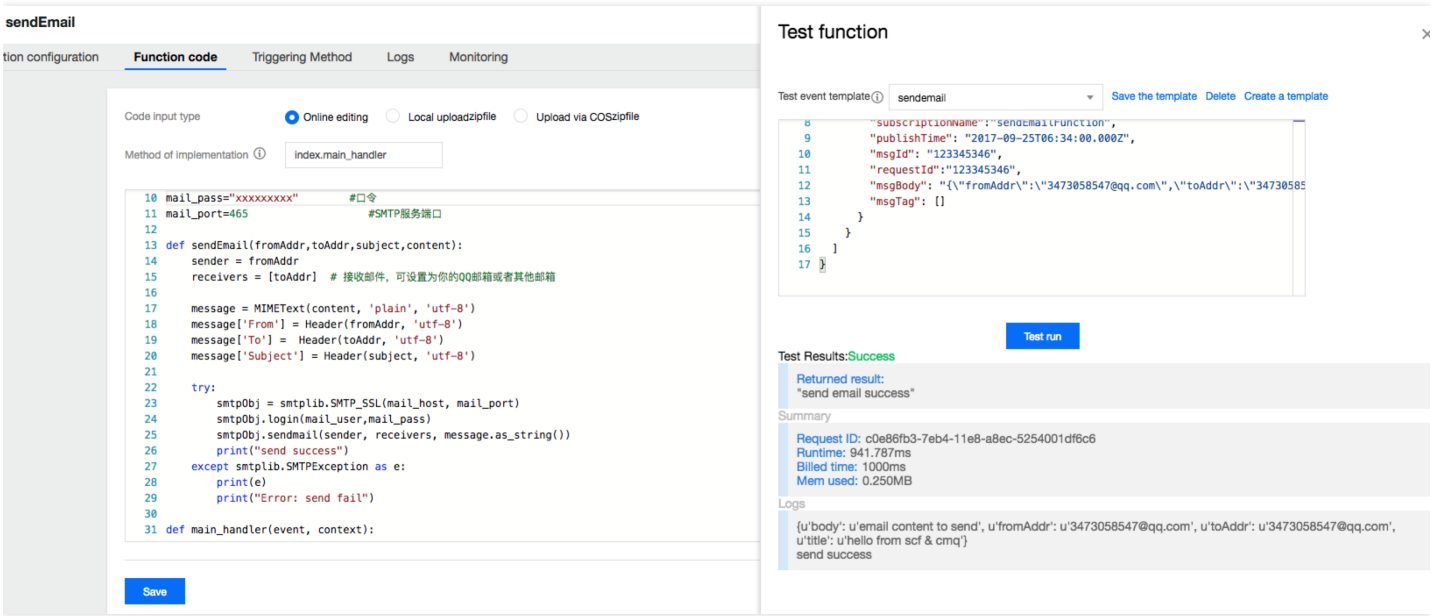
1) In the details page of the sendEmail function you just created, click **Test**.

2) Enter the following in the test template:

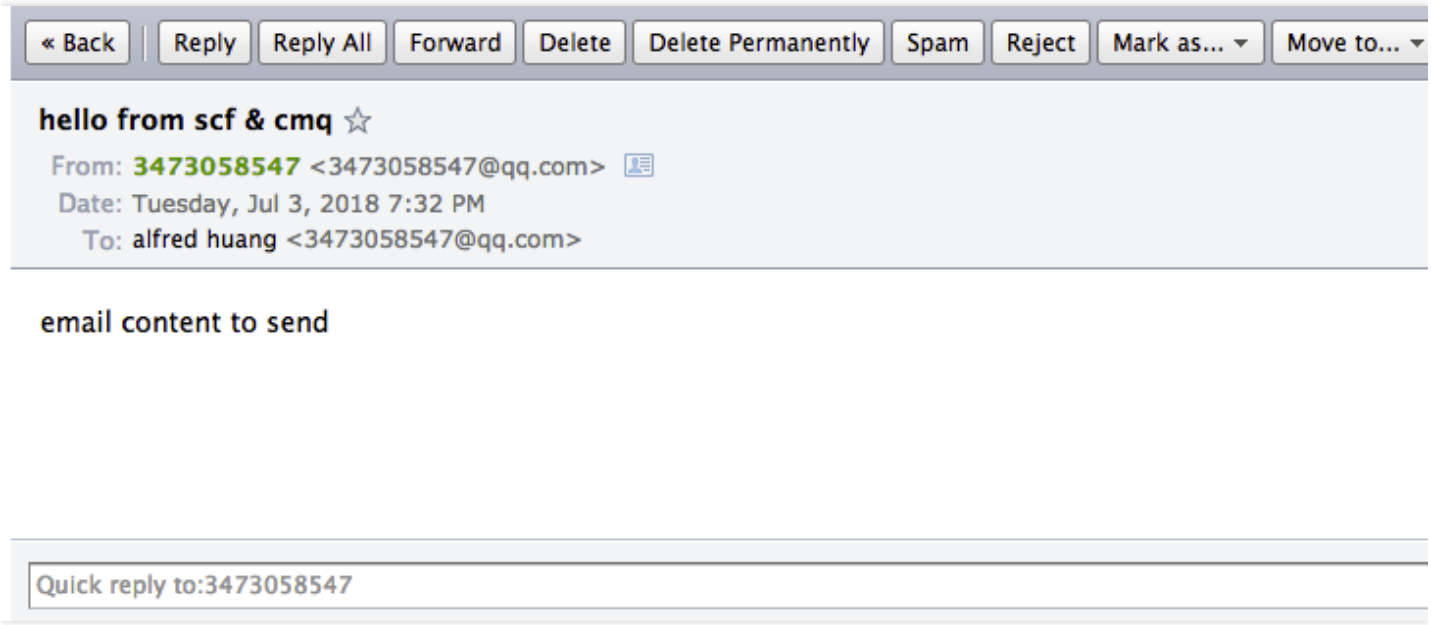
```
{
"Records": [
{
"CMQ": {
"type": "topic",
"topicOwner":1253970226,
"topicName": "sendEmailQueue",
"subscriptionName":"sendEmailFunction",
"publishTime": "2017-09-25T06:34:00.000Z",
"msgId": "123345346",
"requestId":"123345346",
"msgBody": "{\"fromAddr\":\"3473058547@qq.com\",\"toAddr\":\"3473058547@qq.com\",\"title\":\"hello from scf & cmq\",\"body\":\"email content to send\"}",
"msgTag": []
}
}
]
}
```


You can modify both `fromAddr` and `toAddr` in the `msgBody` field to your email address. In this way, you can send an email from and to the same email address to test the validity of email sending. Here, we use `3473058547@qq.com` to test.

3) Click **Run** to view the results. This program is running normally if "send email success" is displayed in both the returned value and the log.



4) Go to the mailbox you specified to check whether the email is received. Open the email to check whether configured information is displayed.



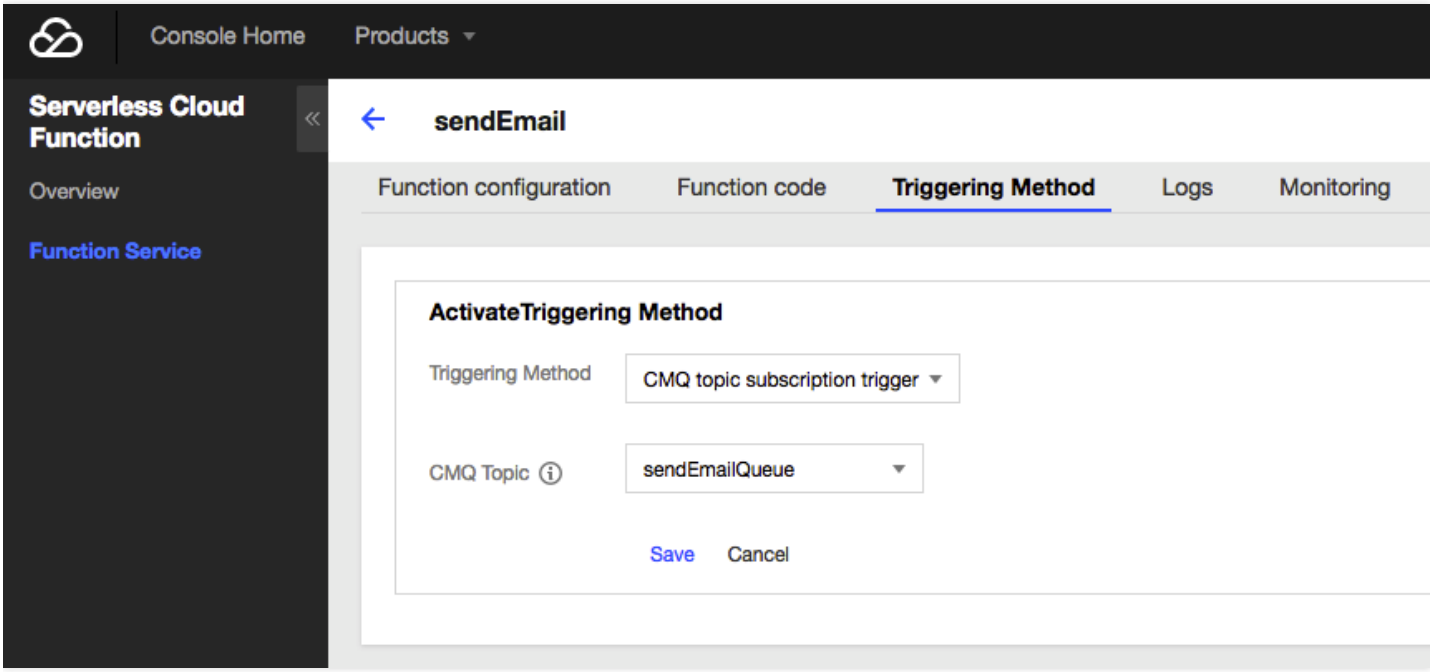
Step 3. Adding Trigger and Test

Last updated：2018-09-03 15:29:49

If you complete "Step 2: Create sendEmail Function and Test", and the test result meets the expectation, you can add a CMQ Topic trigger configuration so that the CMQ Topic can send message events to the SCF and call the function.

1. In the details page of the sendEmail function you just created, click the **Triggering Method** tab and click **Add trigger mode**.

2. Select **CMQ topic subscription trigger**, and `sendEmailQueue` created in Step 1: Create a CMQ Topic for the CMQ Topic, and then click **Save**.



In this way, you have completed all steps. You can test the configuration by following the steps below:

1. Go to Messaging Service CMQ. Select **Subscribe Topic** in the left navigation pane, find the created `sendEmailQueue` queue from the list, and click `Send MSG` in the Operation column of the queue.
   Enter the following message in the popup window:
   You can modify the message content as needed, including "fromAddr", "toAddr", "title", and "body".

```
{
"fromAddr":"xxx@qq.com",
"toAddr":"xxx@qq.com",
"title":"hello from scf & cmq",
"body":"email content to send"
}
```

2. Monitor the activities of the `sendEmail` function you created in the Serverless Cloud Function Console and select **Logs** to check the logs in which the calling of the function is recorded.

3. Log in to your receiving mailbox and check whether the email is received and the email content is correct.

After the test, you can embed the CMQ SDK in your application codes and send the message defined in the example to the `sendEmailQueue` queue to send the email.