FunctionGraph

# Best Practices

| | |
|---|---|
| **Issue** | 01 |
| **Date** | 2021-07-08 |

HUAWEI TECHNOLOGIES CO., LTD.

## Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

## Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

# Contents



# 1 Compressing Images

## 1.1 Introduction

The best practice for FunctionGraph guides you through image compressing based on a function.

### Scenarios

- Upload images to a specified Object Storage Service (OBS) bucket.
- Compress each uploaded image.
- Upload the processed images to another specified OBS bucket.

  NOTE

  1. This tutorial uses two different OBS buckets.
  2. The function you create must be in the same region (default region) as the OBS buckets.

### Procedure

- Create two buckets on the OBS console.
- Create a function with an OBS trigger.
- Upload an image to one of the buckets.
- The function is triggered to compress the image.
- The function uploads the processed image to the other bucket.

  NOTE

  After you complete this tutorial, your account will have the following resources:

  1. Two OBS buckets (respectively used for storing uploaded and processed images)
  2. A thumbnail image creation function (fss_examples_image_thumbnail)
  3. An OBS trigger used for associating the function with the OBS buckets

# 1.2 Preparation

Before creating a function and adding an event source, you need to create two OBS buckets to respectively store uploaded and compressed images.

After creating the OBS buckets, you must create an agency to delegate FunctionGraph to access OBS resources.

## Creating OBS Buckets

### Precautions

- Ensure that the source bucket for storing uploaded images, the target bucket for storing processed images, and the function you will create are in the same region. In this tutorial, all of them are in **CN North-Beijing1**, the default region where FunctionGraph resides.
- Use two different OBS buckets. If only one bucket is used, the function will be executed infinitely. (When an image is uploaded to the bucket, the function is triggered to process the image and store the processed image into the bucket again. In this way, the function executes endlessly.)

### Procedure

**Step 1** Log in to the **OBS console**, and click **Create Bucket**.

**Step 2** On the **Create Bucket** page, set the bucket information, as shown in **Figure 1-1**.

For **Region**, select a region.

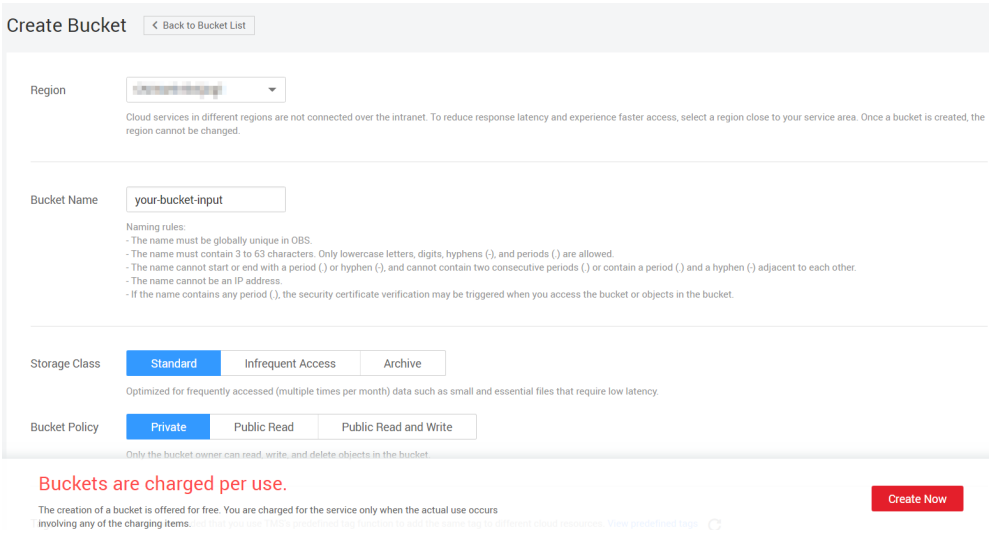
For **Bucket Name**, enter **your-bucket-input**.

For **Storage Class**, select **Standard**.

For **Bucket Policy**, select **Private**.

Click **Create Now**.

**Figure 1-1** Creating the source bucket



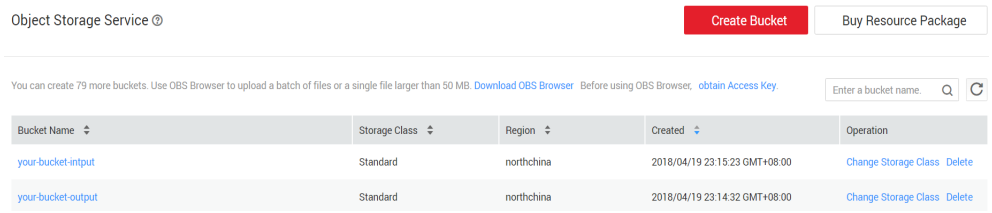

  

**Step 3** Repeat **Step 2** to create the target bucket.

Name the target bucket as **your-bucket-output**, and select the same region and storage class as those of the source bucket.

**Step 4** After creation, check whether the two buckets are displayed in the bucket list, as shown in **Figure 1-2**.

**Figure 1-2** Bucket list




**----End**

## Creating an Agency

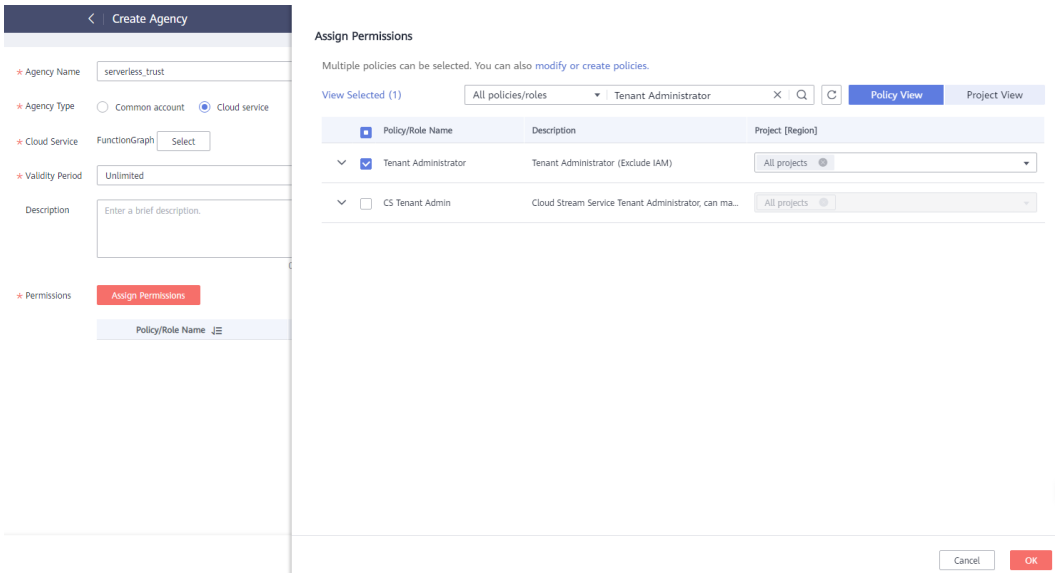
**Step 1** Log in to the **IAM console**, and choose **Agencies** in the navigation pane.

**Step 2** On the **Agencies** page, click **Create Agency**.

**Step 3** Set the agency information.

- For **Agency Name**, enter **serverless_trust**.
- For **Agency Type**, select **Cloud service**.
- For **Cloud service**, select **FunctionGraph**.
- For **Validity Period**, select **Unlimited**.
- Click **Assign Permissions**. On the **Assign Permissions** page, select **Tenant Administrator**.

**Figure 1-3** Creating an agency



NOTE

Users with the Tenant Administrator permission can perform any operations on all cloud resources of the enterprise.

**Step 4** Click **OK**.

**----End**

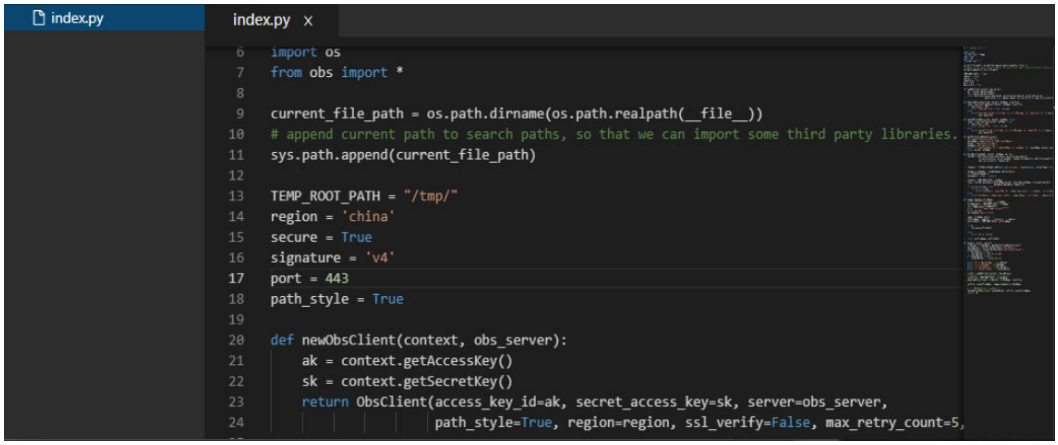
# 1.3 Building a Program

**Download** and use the image compressing program package provided in this example.

## Creating a Deployment Package

This example uses a Python function to compress images. For details about function development, see **Developing Functions in Python**. **Figure 1-4** shows the sample code directory. The service code is not described.

**Figure 1-4** Sample code directory




Under the directory, **index.py** is a handler file. The following code is a snippet of the handler file. Parameter **obs_output_bucket** is the address for storing compressed images and must be configured when you create a function.

```
def handler (event, context):
    srcBucket, srcObjName = getObjInfoFromObsEvent(event)
    obs_address = context.getUserData('obs_address')
    outputBucket = context.getUserData('obs_output_bucket')
    if obs_address is None:
        obs_address = '100.125.15.200'
    if outputBucket is None:
        outputBucket = 'casebucket-out'

    print "*** srcBucketName: " + srcBucket
    print "*** srcObjName:" + srcObjName
    print "*** obs_address: " + obs_address
    print "*** output bucket: " + outputBucket

    client = newObsClient(context, obs_address)
    # download file uploaded by user from obs
    localFile = TEMP_ROOT_PATH + srcObjName
    downloadFile(client, srcBucket, srcObjName, localFile)
```

```
outFile, outputFileName = image_thumbnail(srcObjName)

# Upload converted files to a new OBS bucket.
uploadFileToObs(client, outputBucket, outFile, outputFileName)
return 'OK'
```

## Creating a Function

When creating a function, specify an agency with OBS access permissions so that FunctionGraph can invoke the OBS service.

**Step 1** Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane.

**Step 2** Click **Create Function**.

**Step 3** Set the function information.

1. Set the basic information, as shown in **Figure 1-5**.

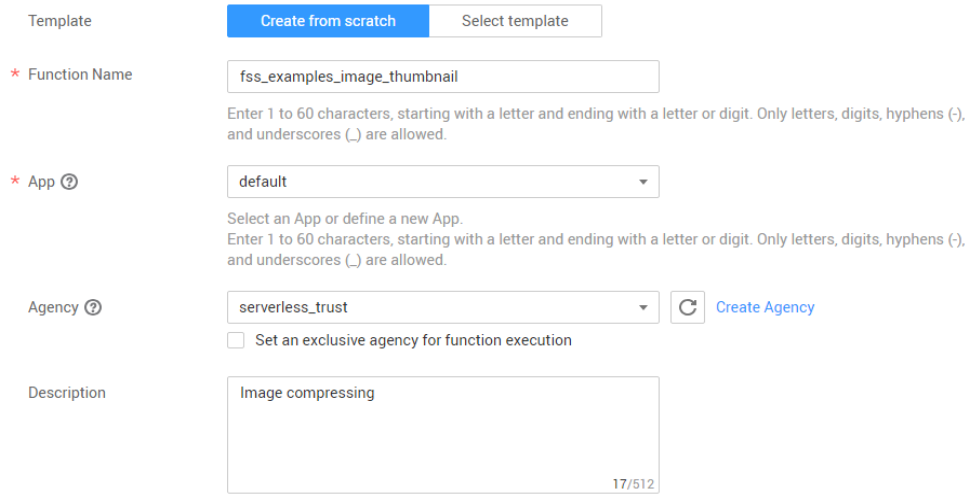
   For **Function Name**, enter **fss_examples_image_thumbnail**.

   For **App**, select **default**.

   For **Description**, enter **Image compressing**.

   For **Agency**, select **serverless_trust** created in **Creating an Agency**.

   **Figure 1-5** Basic information

2. Set the code information, as shown in **Figure 1-6**.

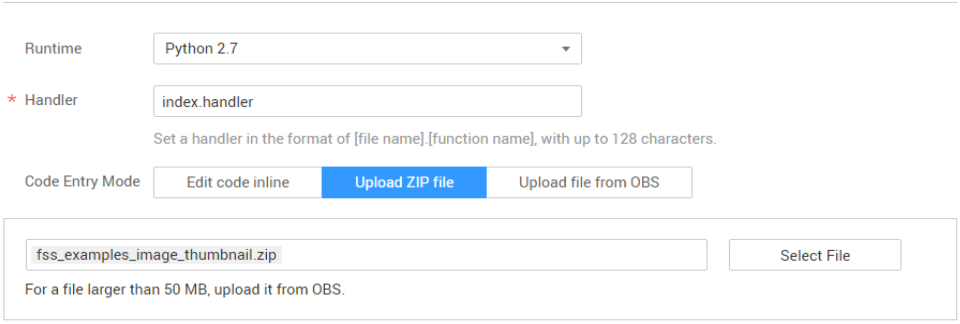
   For **Runtime**, select **Python 2.7**.

   For **Handler**, enter **index.handler**.

   For **Code Entry Mode**, select **Upload ZIP file**, and upload the sample code package **fss_examples_image_thumbnail.zip**.

**Figure 1-6** Code information




3. Click **Create Function**.

**Step 4** On the **fss_examples_image_thumbnail** page, select the **Configuration** tab and set the environment information, as shown in **Figure 1-7**.

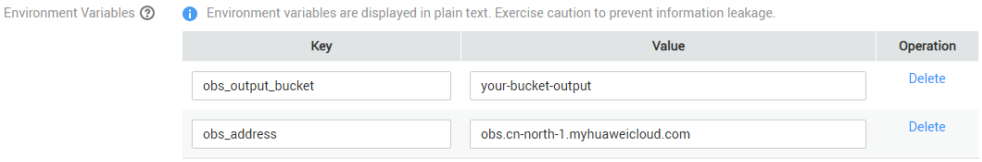
For **Memory**, select **256**.

For **Timeout**, enter **40**.

For **Environment Variables**, define parameter **obs_output_bucket** in the **index.py** file as the key and bucket **your-bucket-output** created in **Creating OBS Buckets** as the value.

The key **obs_address** indicates the address of the OBS bucket defined in the **index.py** file to store processed images. The value **obs.cn-north-1.myhuaweicloud.com** indicates the region where the bucket has been created.

**Figure 1-7** Defining environment variables




**NOTE**

The value of **obs_address** should be in the format "obs.{region}.myhuaweicloud.com". For the region name, see **Regions and Endpoints**.

**Step 5** Click **Save** in the upper right corner.

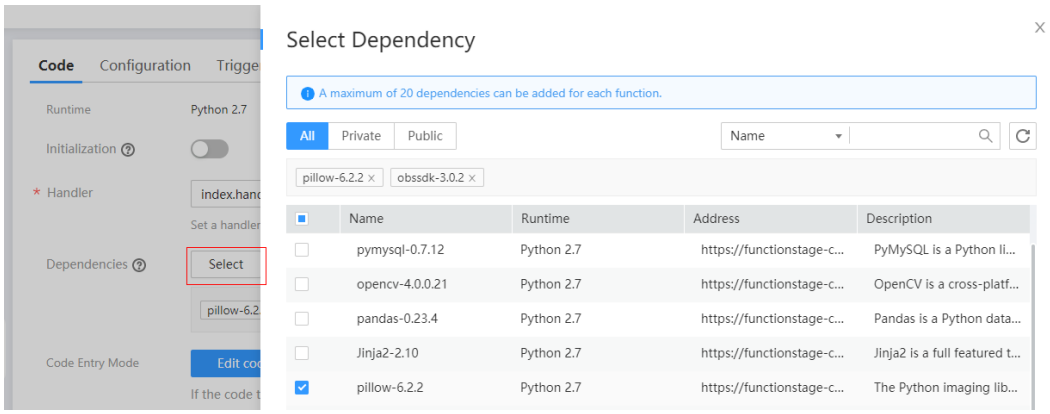
**----End**

## Selecting a Dependency

The sample code depends on OBS and Pillow packages, which need to be imported as dependencies. The procedure is as follows:

**Step 1** On the **fss_examples_image_thumbnail** details page, click the **Code** tab and then click **Select** in the same row as **Dependencies**. The **Select Dependency** dialog box is displayed, as shown in the following figure.




**Step 2** Select the required dependencies and click **OK**.

**Step 3** Click **Save** in the upper right corner.

**----End**

# 1.4 Adding an Event Source

After creating the OBS buckets and function, you can add an event source to the function by creating an OBS trigger. Perform the following procedure:

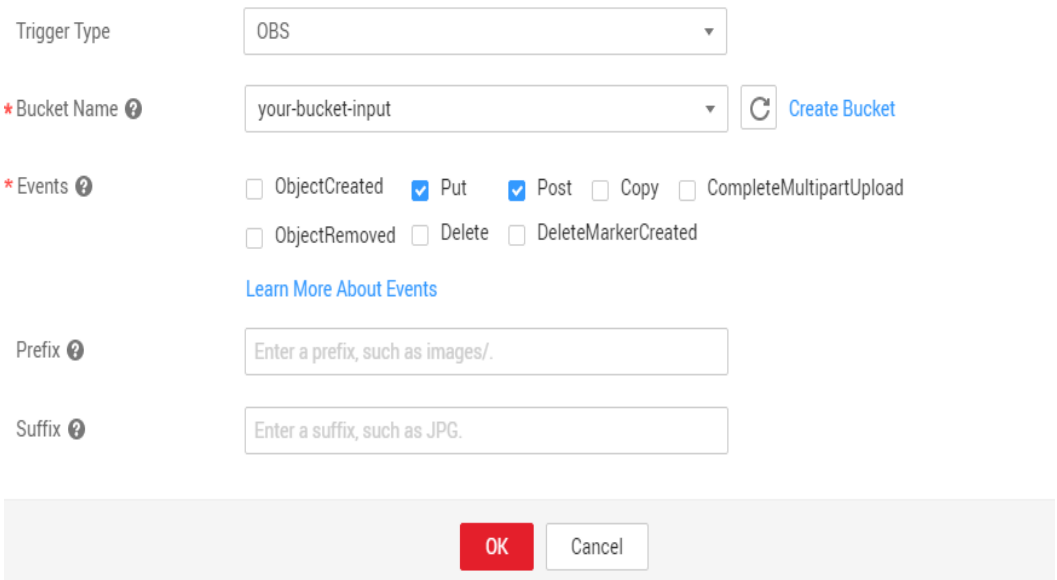
**Step 1** On the **fss_examples_image_thumbnail** page, click the **Triggers** tab and click **Create Trigger**.

**Step 2** Select **OBS** for **Trigger Type**, and set the trigger information, as shown in **Figure 1-8**.

For **Bucket Name**, select **your-bucket-input** created in **Creating OBS Buckets**.

For **Events**, select **Put** and **Post**.

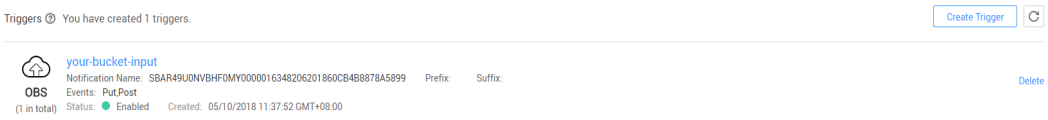
**Figure 1-8** Creating a trigger



**Step 3** Click **OK**. The trigger is created, as shown in **Figure 1-9**.

**Figure 1-9** OBS trigger




**NOTE**

After the OBS trigger is created, when an image is uploaded or updated to bucket **your-bucket-input**, an event is generated to trigger the function.

**----End**

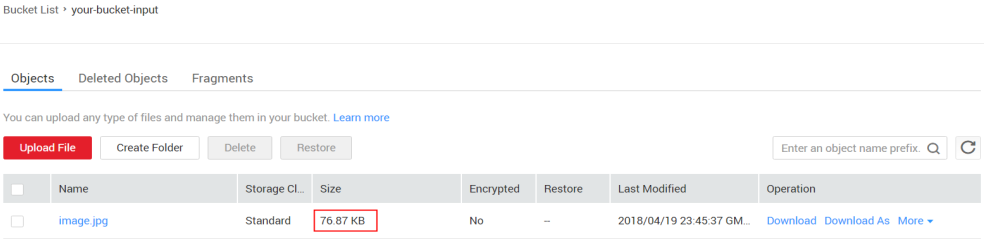
# 1.5 Compressing Images

When an image is uploaded or updated to bucket **your-bucket-input**, an event is generated to trigger the function. The function compresses the image and stores the compressed one into bucket **your-bucket-output**.

## Uploading an Image to Generate an Event

Log in to the **OBS console**, go to the object page of the **your-bucket-input** bucket, and upload the **image.jpg** image, as shown in **Figure 1-10**.

**Figure 1-10** Uploading an image




**NOTE**

The size of the original **image.jpg** file exceeds 28 KB.

## Triggering the Function

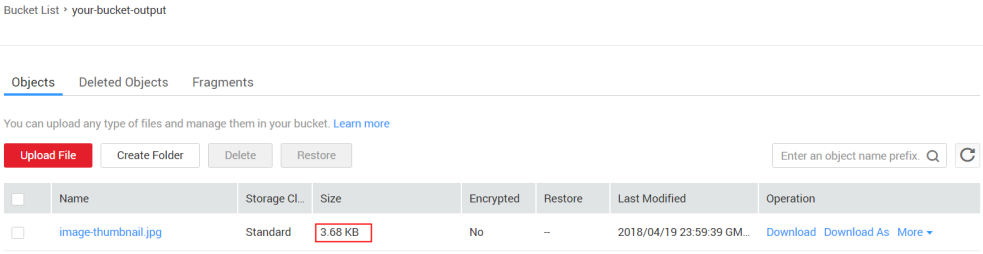
After the image is uploaded to bucket **your-bucket-input**, OBS generates an event to trigger the image compressing function. The function compresses the image and stores the compressed one into bucket **your-bucket-output**. You can view running logs of the function on the **Logs** tab page, as shown in **Figure 1-11**.

**Figure 1-11** Logs


On the **your-bucket-output** page on the OBS console, you can see the compressed image **image-thumbnail.jpg**, whose size is less than 8 KB after compression, as shown in **Figure 1-12**.

**Figure 1-12** Compressing the image



# 2 Watermarking Images

## 2.1 Introduction

The best practice for FunctionGraph guides you through image watermarking based on a function.

### Scenarios

- Upload images to a specified OBS bucket.
- Watermark each uploaded image.
- Upload the processed images to another specified OBS bucket.

  NOTE

  1. This tutorial uses two different OBS buckets.
  2. The function you create must be in the same region (default region) as the OBS buckets.

### Procedure

- Create two buckets on the OBS console.
- Create a function with an OBS trigger.
- Upload an image to one of the buckets.
- The function is triggered to watermark the image.
- The function uploads the processed image to the other bucket.

  NOTE

  After you complete the operations in this tutorial, your account will have the following resources:

  1. Two OBS buckets (respectively used for storing uploaded and processed images)
  2. An image watermarking function
  3. An OBS trigger used for associating the function with the OBS buckets

## 2.2 Preparation

Before creating a function and adding an event source, you need to create two OBS buckets to respectively store uploaded and watermarked images.

After creating the OBS buckets, you must create an agency to delegate FunctionGraph to access OBS resources.

## Creating OBS Buckets

**Precautions**

- Ensure that the source bucket for storing uploaded images, the target bucket for storing processed images, and the function you will create are in the same region. In this tutorial, all of them are in **CN North-Beijing4**, the default region where FunctionGraph resides.
- Use two different OBS buckets. If only one bucket is used, the function will be executed infinitely. (When an image is uploaded to the bucket, the function is triggered to process the image and store the processed image into the bucket again. In this way, the function executes endlessly.)

**Procedure**

**Step 1** Log in to the **OBS console**, and click **Create Bucket**.

**Step 2** On the **Create Bucket** page, set the bucket information, as shown in **Figure 2-1**.

For **Region**, select a region.

For **Data Redundancy Policy**, select **Single-AZ storage**.

For **Bucket Name**, enter **hugb-bucket-input**.

For **Storage Class**, select **Standard**.

For **Bucket Policy**, select **Private**.

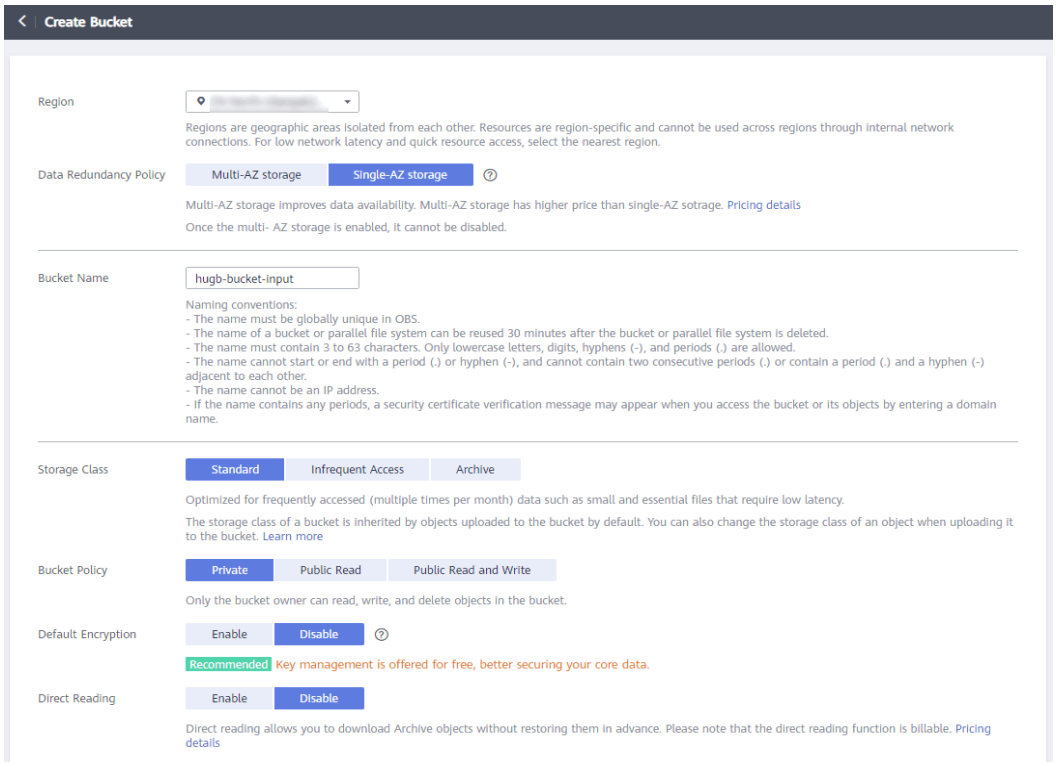
For **Default Encryption**, select **Disable**.

For **Direct Reading**, select **Disable**.

Click **Create Now**.

**Figure 2-1** Creating an OBS bucket

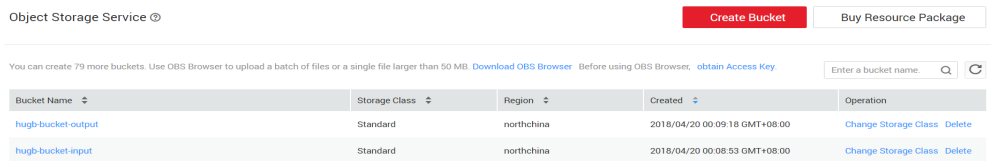



**Step 3** Repeat **Step 2** to create the target bucket.

Name the target bucket as **hugb-bucket-output**, and select the same region and storage class as those of the source bucket.

**Step 4** After creation, check whether the two buckets are displayed in the bucket list, as shown in **Figure 2-2**.

**Figure 2-2** Bucket list




**----End**

## Creating an Agency

**Step 1** Log in to the **IAM console**, and choose **Agencies** in the navigation pane.

**Step 2** On the **Agencies** page, click **Create Agency**.

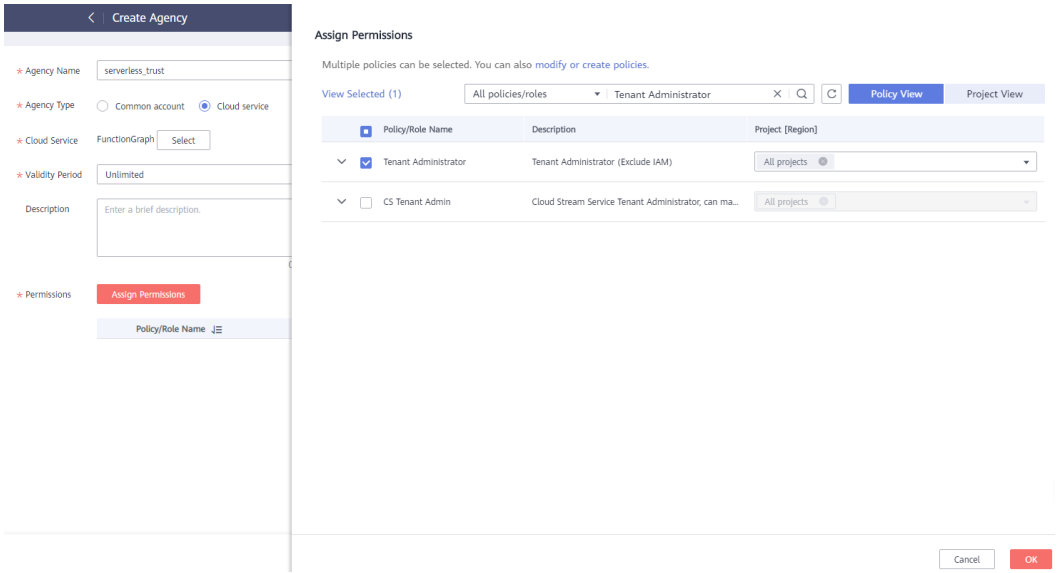
**Step 3** Set the agency information.

- For **Agency Name**, enter **serverless_trust**.
- For **Agency Type**, select **Cloud service**.
- For **Cloud service**, select **FunctionGraph**.
- For **Validity Period**, select **Unlimited**.

- Click **Assign Permissions**. On the **Assign Permissions** page, select **Tenant Administrator**.

**Figure 2-3** Creating an agency




**NOTE**

Users with the Tenant Administrator permission can perform any operations on all cloud resources of the enterprise.

**Step 4** Click **OK**.

**----End**

# 2.3 Building a Program

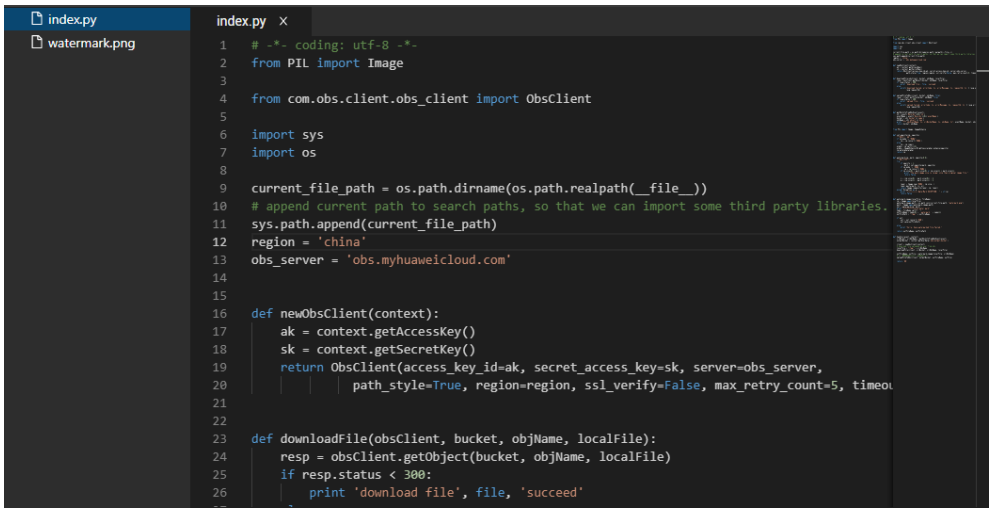
**Download** and use the image watermarking program package provided in this example.

## Creating a Deployment Package

This example uses a Python function to watermark images. For details about function development, see **Developing Functions in Python**. **Figure 2-4** shows the sample code directory. The service code is not described.

**Figure 2-4** Sample code directory




Under the directory, **index.py** is a handler file. The following code is a snippet of the handler file. Parameter **obs_output_bucket** is the address for storing watermarked images and must be configured when you create a function.

```
def handler(event, context):
    srcBucket, srcObjName = getObjInfoFromObsEvent(event)
    outputBucket = context.getUserData('obs_output_bucket')

    client = newObsClient(context)
    # download file uploaded by user from obs
    localFile = "/tmp/" + srcObjName
    downloadFile(client, srcBucket, srcObjName, localFile)

    outFileName, outFile = watermark_image(localFile, srcObjName)
    # Uploads converted files to a new OBS bucket.
    uploadFileToObs(client, outputBucket, outFileName, outFile)

    return 'OK'
```

## Creating a Function

When creating a function, specify an agency with OBS access permissions so that FunctionGraph can invoke the OBS service.

**Step 1** Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane.

**Step 2** Click **Create Function**.

**Step 3** Set the function information.

1. Set the basic information, as shown in **Figure 2-5**.

   For **Function Name**, enter **fss_examples_image_watermark**.

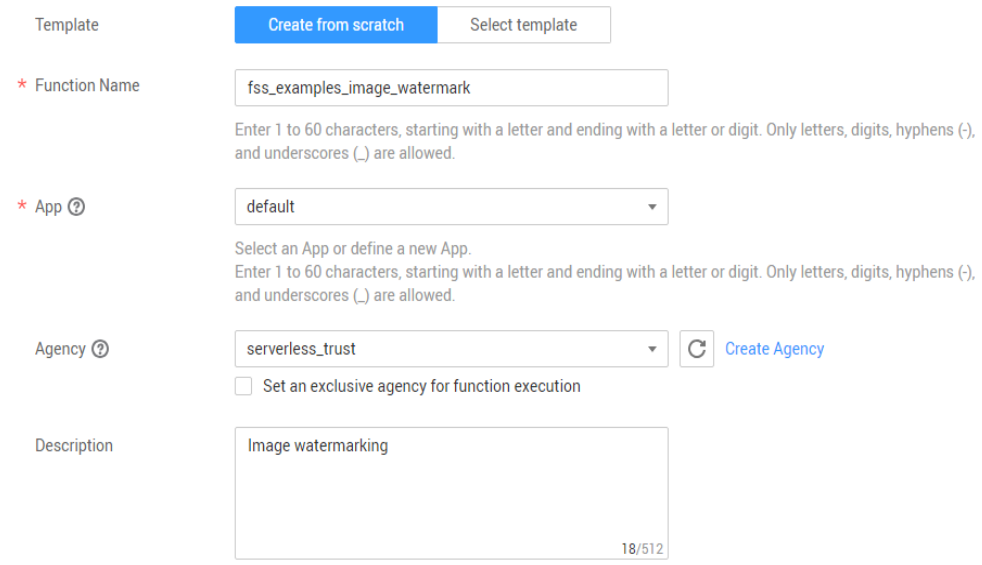
   For **App**, select **default**.

   For **Description**, enter **Image watermarking**.

   For **Agency**, select **serverless_trust** created in **Creating an Agency**.

**Figure 2-5** Basic information



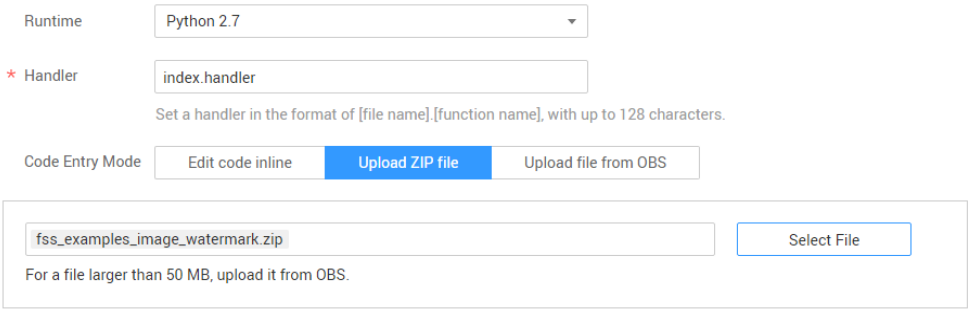

2. Set the code information, as shown in **Figure 2-6**.

   For **Runtime**, select **Python 2.7**.

   For **Handler**, enter **index.handler**.

   For **Code Entry Mode**, select **Upload ZIP file**, and upload the sample code package **fss_examples_image_watermark.zip**.

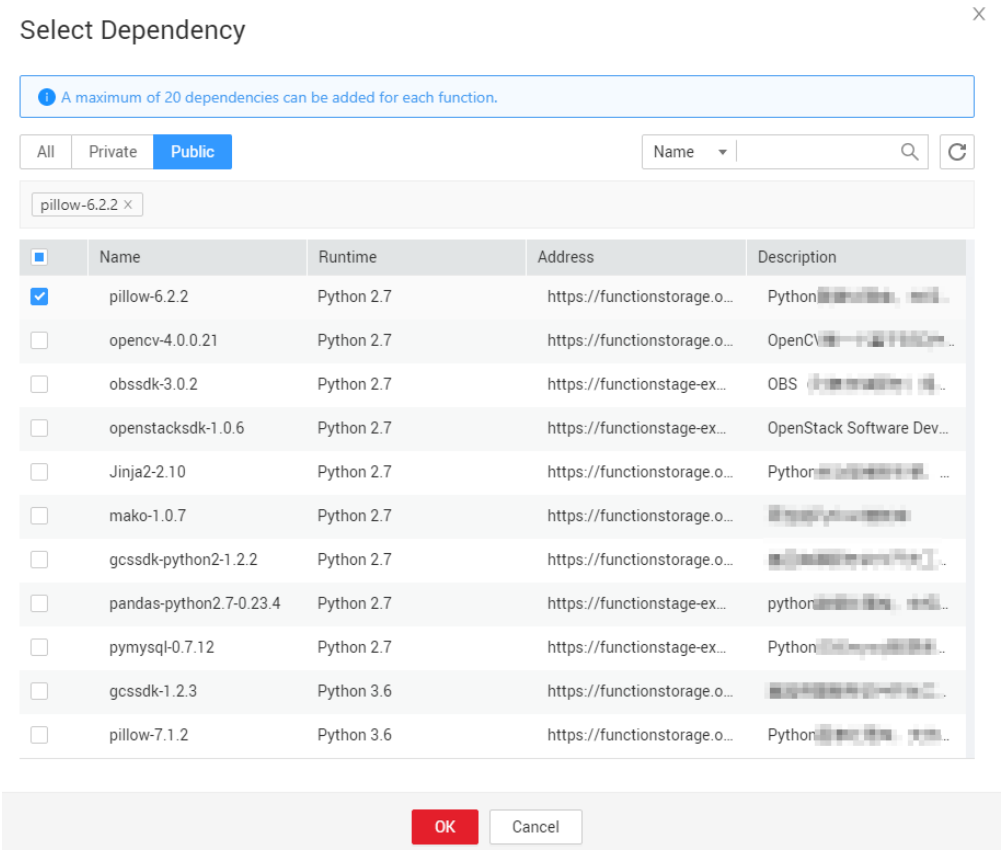
   **Figure 2-6** Code information




3. Click **Create Function**.

**Step 4** On the details page of the **fss_examples_image_watermark** function, click the **Code** tab, and add the **pillow-6.2.2** dependency.




**Step 5** On the **fss_examples_image_watermark** page, select the **Configuration** tab and set the environment information, as shown in **Figure 2-7**.

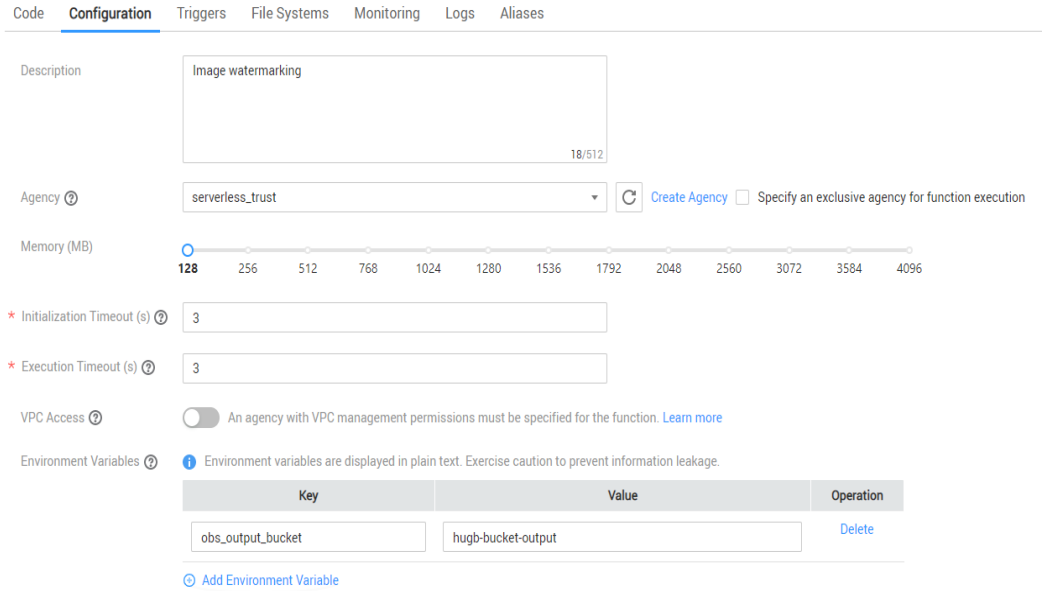
For **Memory**, select **128**.

For **Timeout**, enter **3**.

For **Environment Variables**, define parameter **obs_output_bucket** in the **index.py** file as the key and bucket **hugb-bucket-output** created in **Creating OBS Buckets** as the value. **obs_region** indicates the region where the OBS bucket **obs_output_bucket** is located. The following figure is for reference only. Replace the following value with the actual value.

**Figure 2-7** Environment information




**Step 6** Click **Save**.

**----End**

# 2.4 Adding an Event Source

After creating the OBS buckets and function, you can add an event source to the function by creating an OBS trigger. Perform the following procedure:

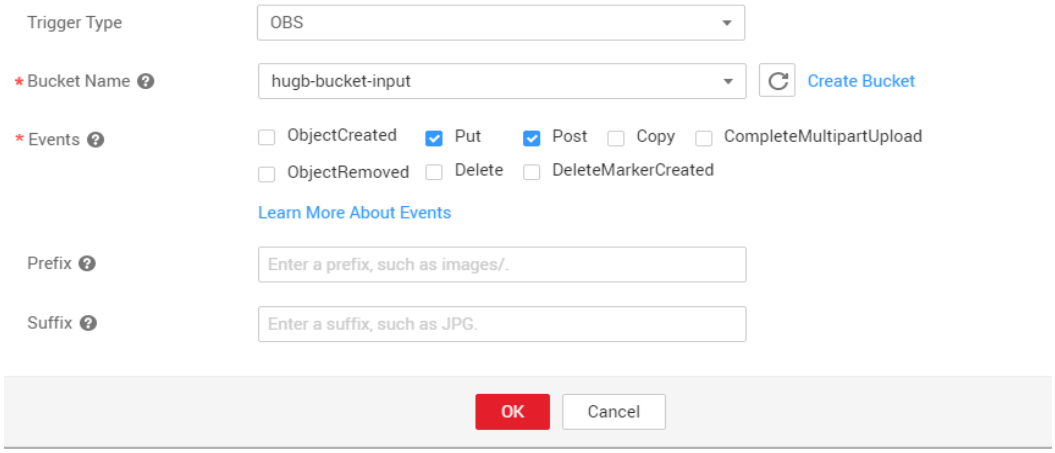
**Step 1** On the **fss_examples_image_watermark** page, click the **Triggers** tab and click **Create Trigger**.

**Step 2** Select **OBS** for **Trigger Type**, and set the trigger information, as shown in **Figure 2-8**.

For **Bucket Name**, select **hugb-bucket-input** created in **Creating OBS Buckets**.

For **Events**, select **Put** and **Post**.

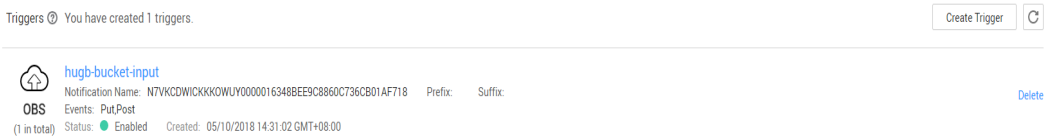
**Figure 2-8** Creating an OBS trigger



**Step 3** Click **OK**. The trigger is created, as shown in **Figure 2-9**.

**Figure 2-9** OBS trigger




**NOTE**

After the OBS trigger is created, when an image is uploaded or updated to bucket **hugb-bucket-input**, an event is generated to trigger the function.

**----End**

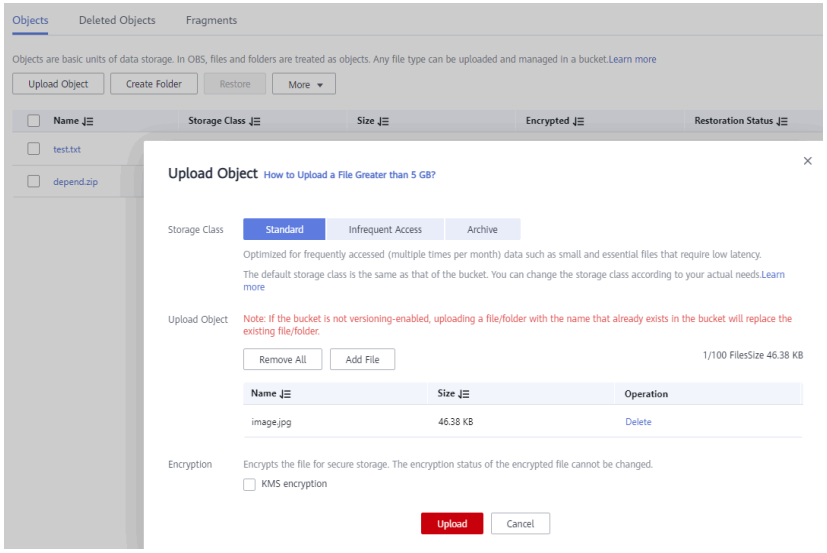
# 2.5 Watermarking Images

When an image is uploaded or updated to bucket **hugb-bucket-input**, an event is generated to trigger the function. The function watermarks the image and stores the watermarked one into bucket **hugb-bucket-output**.

## Uploading an Image to Generate an Event

Log in to the **OBS console**, go to the object page of the **hugb-bucket-input** bucket, and upload the **image.jpg** image, as shown in **Figure 2-10**.

**Figure 2-10** Uploading an image

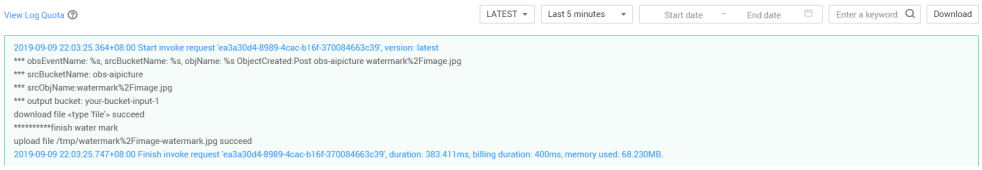



## Triggering the Function

After the image is uploaded to bucket **hugb-bucket-input**, OBS generates an event to trigger the image watermarking function. The function watermarks the image and stores the watermarked one into bucket **hugb-bucket-output**. You can view running logs of the function on the **Logs** tab page, as shown in **Figure 2-11**.

**Figure 2-11** Logs




On the **hugb-bucket-output** page in the OBS console, you can see the watermarked image **image-watermark.jpg**, as shown in **Figure 2-12**. In the **Operation** column, click **Download** to download the image and view the watermarking effect, as shown in **Figure 2-13**.

**Figure 2-12** Output image

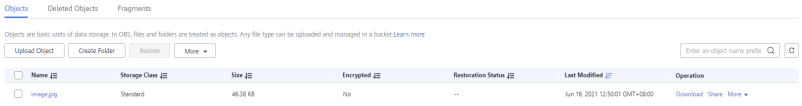



**Figure 2-13** Watermarked image

# 3 Filtering Sensitive Information

## 3.1 Introduction

The best practice for HUAWEI CLOUD FunctionGraph guides you through sensitive information filtering based on a function.

### Scenarios

When using the Distributed Message Service (DMS) to store important information, such as messages, determine whether messages stored in a specified DMS queue contain sensitive information, such as political statements and advertisements, and are not suitable for publishing.

### Procedure

- On the DMS console, create a message queue and consumer group, and create a message.
- On the FunctionGraph console, create a function with a DMS trigger to poll the created queue for messages.
- The function reads and processes the created message, and determines whether the information contained in the message is suitable for publishing.

## 3.2 Preparation

When creating a function on the FunctionGraph console, ensure that the size of the ZIP file to be uploaded does not exceed 50 MB. If it exceeds 50 MB, upload the ZIP file to an OBS bucket and then reference the file from the bucket. In this example, you need to create an OBS bucket to save the program package.

Additionally, you need to create a DMS queue and consumer group before adding an event source to the function.

After creating the OBS bucket, you must create an agency to delegate FunctionGraph to access DMS resources.

## Creating an OBS bucket

**Step 1** Log in to the **OBS console**, and click **Create Bucket**.

**Step 2** On the **Create Bucket** page, set the bucket information, as shown in **Figure 3-1**.

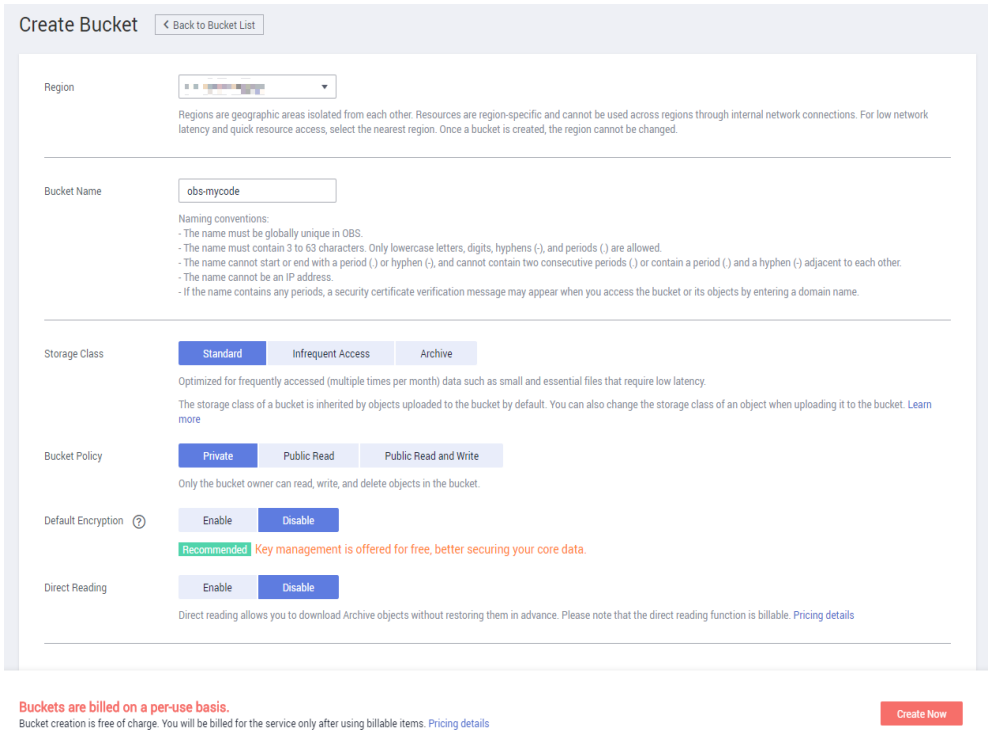
For **Region**, select a region.

For **Bucket Name**, enter **obs-mycode**.

For **Storage Class**, select **Standard**.

For **Bucket Policy**, select **Private**.

**Figure 3-1** Creating a bucket




**Step 3** Click **Create Now**.

**----End**

## Creating a DMS Queue and Consumer Group

### Creating a Queue

**Step 1** Log in to the **DMS console**, and choose **Queue Manager** in the navigation pane.

**Step 2** On the **Queue Manager** page, click **Create Queue**.

**Step 3** In the **Create Queue** dialog box, set the queue information according to **Table 3-1**. The parameter marked with an asterisk (*) is mandatory.

**Table 3-1** Information required for creating a queue

| Parameter | Description |
| --- | --- |
| *Queue Name | Enter **queue-test**. |
| Queue Type | Select **Standard**. |
| Dead Letter Queue | Use the default settings to disable the dead letter queue function. |
| Description | This example does not require a description. |

NOTE

For details about the parameters required for creating a queue, see **Creating a Queue**.

**Step 4** Click **OK**.

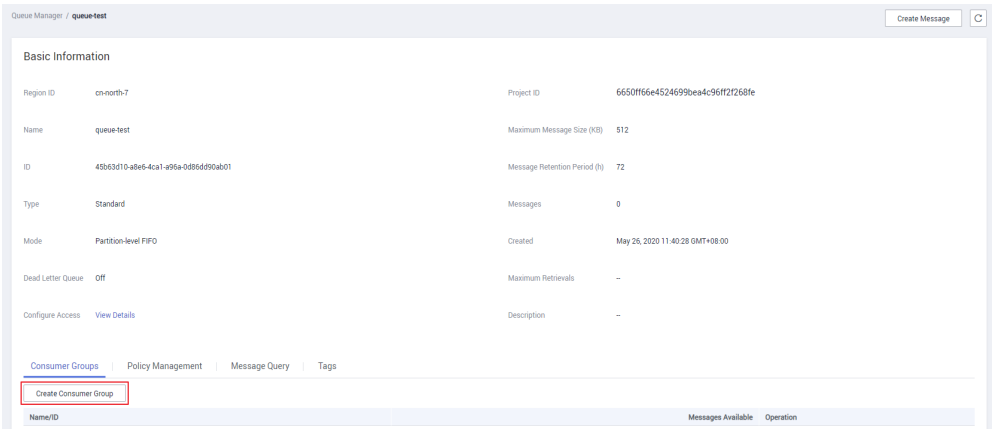
**----End**

**Creating a Consumer Group**

**Step 1** On the **Queue Manager** page, click **queue-test**.

**Step 2** Click **Create Consumer Group**, as shown in **Figure 3-2**.

**Figure 3-2** Creating a consumer group




**Step 3** In the **Create Consumer Group** dialog box, enter a consumer group name according to **Table 3-2**. The parameter marked with an asterisk (*) is mandatory.

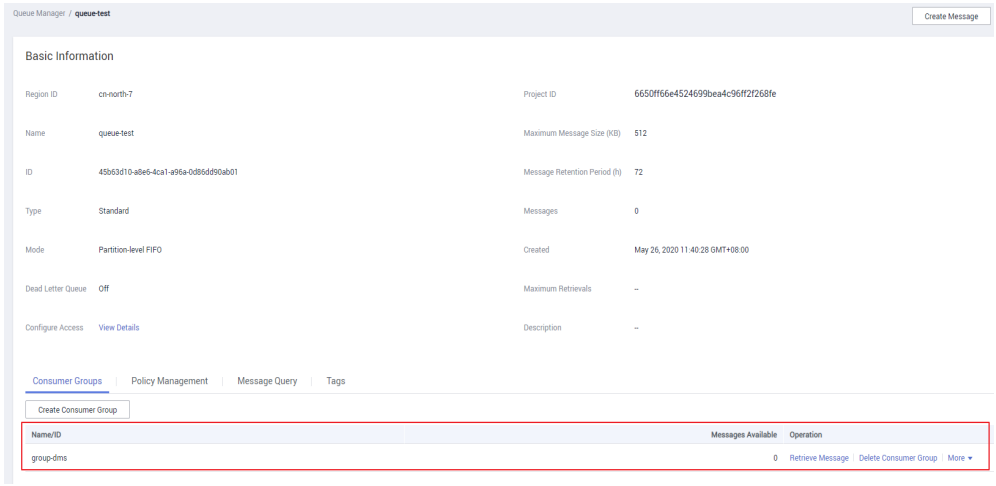
**Table 3-2** Information required for creating a consumer group

| Parameter | Description |
| --- | --- |
| *Consumer Group Name | Enter **group-dms**. |

**Step 4** Click **OK**. The consumer group is created as shown in **Figure 3-3**.

**Figure 3-3** Consumer group




**----End**

## Creating an Agency

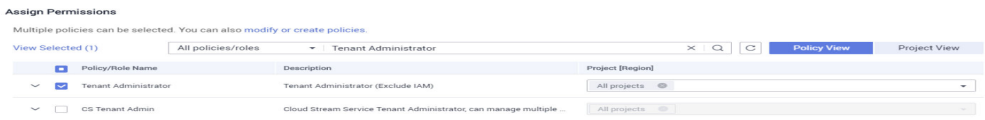
**Step 1** Log in to the **IAM console**, and choose **Agencies** in the navigation pane.

**Step 2** On the **Agencies** page, click **Create Agency**.

**Step 3** Set the agency information.

- For **Agency Name**, enter **serverless_dms**.
- For **Agency Type**, select **Cloud service**.
- For **Cloud service**, select **FunctionGraph**.
- For **Validity Period**, select **Unlimited**.
- Click **Assign Permissions**. On the **Assign Permissions** page, select **Tenant Administrator**, as shown in **Figure 3-4**.

**Figure 3-4** Creating an agency




**NOTE**

Users with the Tenant Administrator permission can perform any operations on all cloud resources in the current region.

**Step 4** Click **OK**.

**----End**

# 3.3 Building a Program

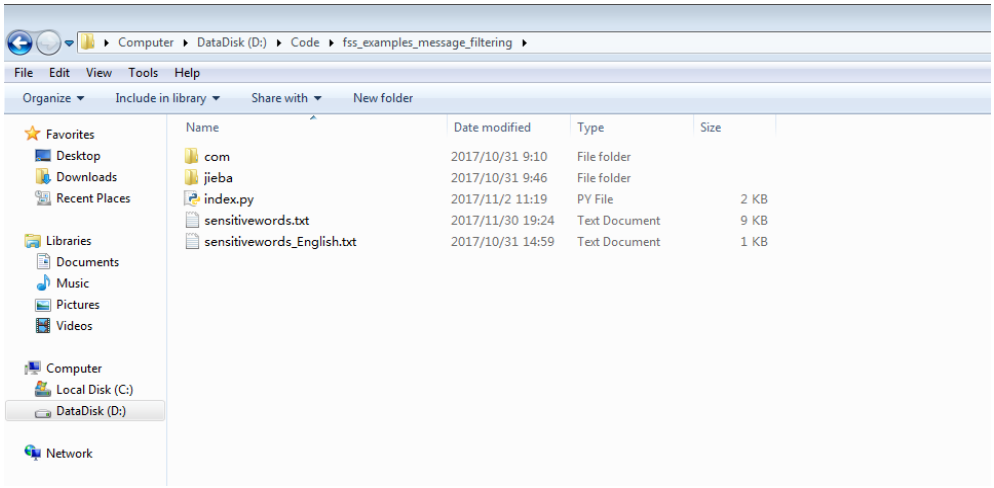
**Download** and use the information filtering program package provided in this example.

## Creating a Deployment Package

This example uses a Python function to filter sensitive information. For details about function development, see **Developing Functions in Python**. **Figure 3-5** shows the sample code directory. The service code is not described.

**Figure 3-5** Sample code directory




Under the directory, the **sensitivewords.txt** file is a sensitive word library, which allows you to define sensitive words and put each word in a separate line.

This tutorial uses the Jieba word segmentation module (one of the best Chinese character segmentation modules in Python) and sensitive word library. Both of them must be placed under the **fss_examples_message_filtering** directory.

**index.py** is a handler file used for executing the function. A code snippet is as follows:

```
# -*- coding:utf-8 -*-
import json
import jieba
import os
import sys

reload(sys)
sys.setdefaultencoding('utf-8')

def is_ok_publish(body, over = 0.1):
    '''
If 10% of the words in a message body are sensitive words, the message is not suitable for publishing.
    '''
    words = list(jieba.cut(body))    #Uses the Jieba word segmentation module to segment incoming messages. For example, if the body is I love China; the words included in this message body are ["I","love","China"].

    filename = os.environ.get('RUNTIME_CODE_ROOT') + '/sensitivewords.txt'    #Reads information from the sensitive word library (in the correct file path). The sensitivewords file is a sensitive word list uploaded together with the code.
    with open(filename)as file:
        sensitive_words = file.read().decode('gbk').split('\r\n')        #The newline character may vary depending on the operating system (OS). It has been verified that the newline character in this code works in a Windows OS.
    num = 0    #Calculates the number of sensitive words that appear in both a message and the sensitive word library.
    for each in (set(words) & set(sensitive_words)):
        num = num + 1
```

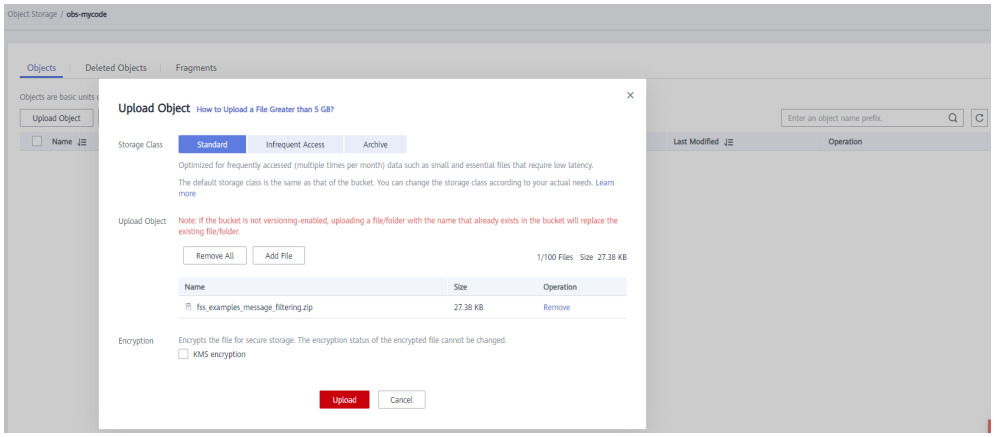
  

```
    length = len(set(words))
    rate = float(num)/length
    if(rate >= over):      #If the number of sensitive words in a message exceeds 10% of the total words, the
messages is sensitive and will not be published.
        return False
    return True    #Publishes the message.

def handler (event, context):

    msg = event[´Messages´]   #Pulls messages from a specified DMS queue.
    body = msg[0]["body"]      #Reads the body of a message.

    flag = is_ok_publish(body) #Determines whether the body of the message is sensitive.
```

## Uploading Code to the OBS Bucket

Log in to the **OBS console**, go to the **Objects** page of the **obs-mycode** bucket, and upload the **sample program package** to this bucket, as shown in **Figure 3-6**.

**Figure 3-6** Uploading the code package




On the **fss_examples_message_filtering** page, file link **https://obs-mycode.obs.myhuaweicloud.com/fss_examples_message_filtering.zip** is displayed.

## Creating a Function

When creating a function, specify an agency with DMS access permissions so that FunctionGraph can invoke the DMS service.

**Step 1** Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane.

**Step 2** Click **Create Function**.

**Step 3** Set the function information.

1. Set the basic information, as shown in **Figure 3-7**.

   For **Function Name**, enter **fss_examples_message_filtering**.

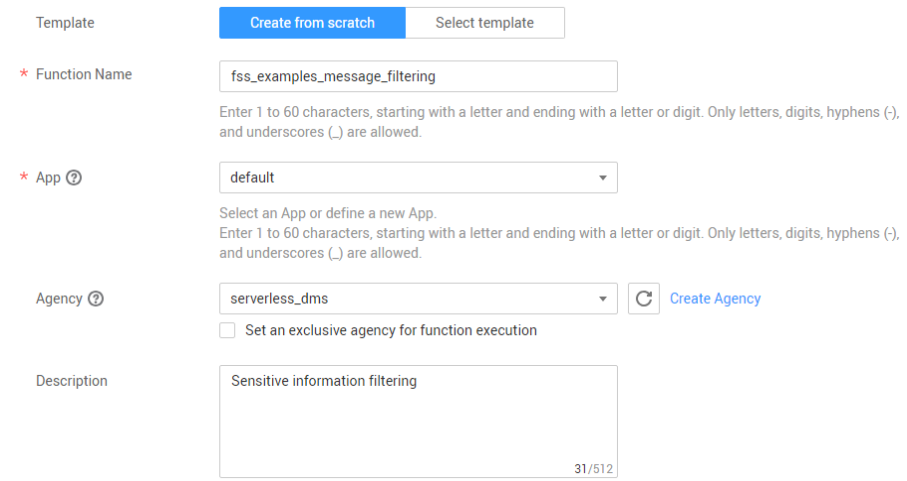
   For **App**, select **default**.

   For **Description**, enter **Sensitive information filtering**.

   For **Agency**, select **serverless_dms** created in **Creating an Agency**.

**Figure 3-7** Basic information



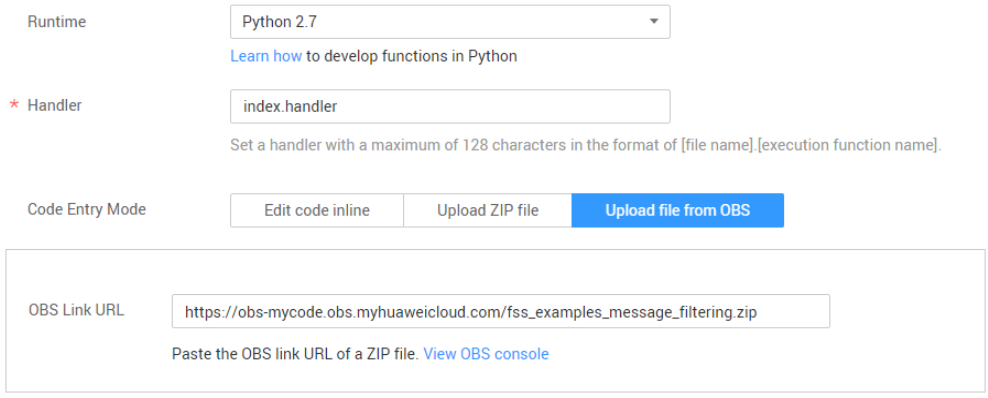

2. Set the code information, as shown in **Figure 3-8**.

   For **Runtime**, select **Python 2.7**.

   For **Handler**, enter **index.handler**.

   For **Code Entry Mode**, select **Upload file from OBS**, and paste OBS link URL **https://obs-mycode.obs.myhuaweicloud.com/fss_examples_message_filtering.zip** obtained in **Uploading Code to the OBS Bucket**.

   **Figure 3-8** Code information




3. Click **Create Function**.

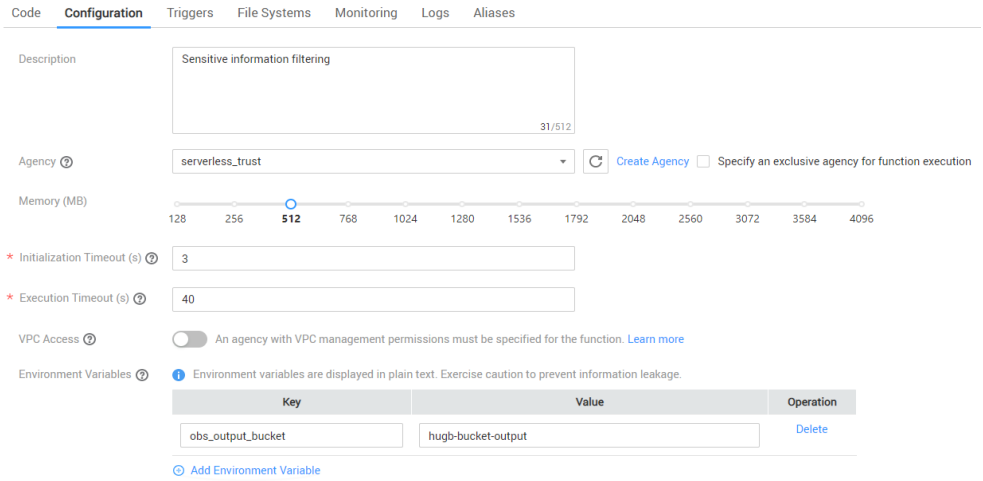
**Step 4** On the **fss_examples_message_filtering** page, select the **Configuration** tab and set the environment information, as shown in **Figure 3-9**.

For **Memory**, select **512**.

For **Timeout**, enter **40**.

**Figure 3-9** Environment information




**Step 5** Click **Save**.

**----End**

# 3.4 Adding an Event Source

After creating the function, you can add an event source by creating a DMS trigger. Perform the following procedure:

**Step 1** On the **fss_examples_message_filtering** page, click the **Triggers** tab and click **Create Trigger**.

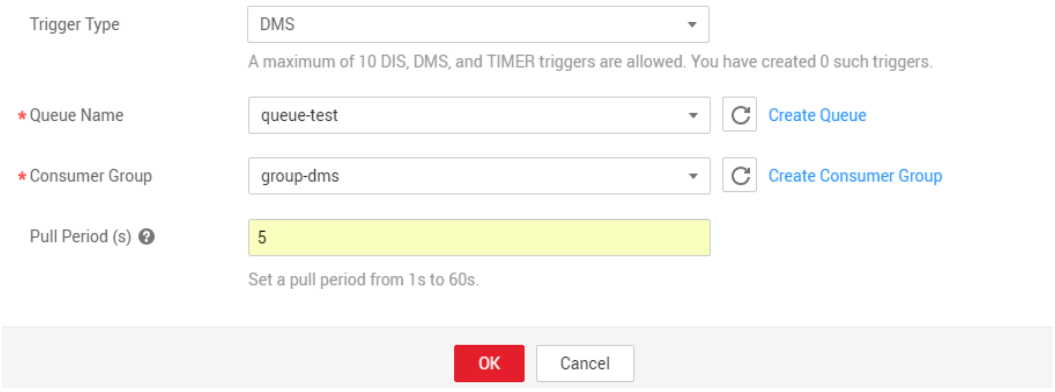
**Step 2** Select **DMS** for **Trigger Type**, and set the trigger information, as shown in **Figure 3-10**.

For **Queue Name**, select **queue-test**.

For **Consumer Group**, select **group-dms**.

For **Pull Period**, enter **5**.

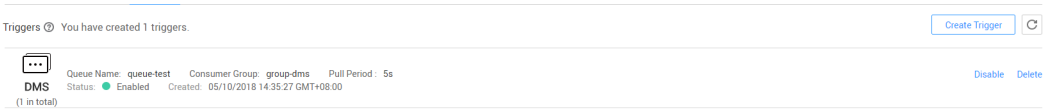
**Figure 3-10** Creating a trigger




**Step 3** Click **OK**. The trigger is created, as shown in **Figure 3-11**.

**Figure 3-11** DMS trigger




**NOTE**

After the DMS trigger is created, the function polls consumer group **group-dms** for messages every 5s. When a message is available, a DMS event is generated to trigger the function.

**----End**

# 3.5 Filtering Sensitive Information

When a message is created in queue **queue-test** for consumer group **group-dms** to consume, the DMS trigger detects the message and generates an event to trigger the function. Then the function runs to determine whether the message contains sensitive information.

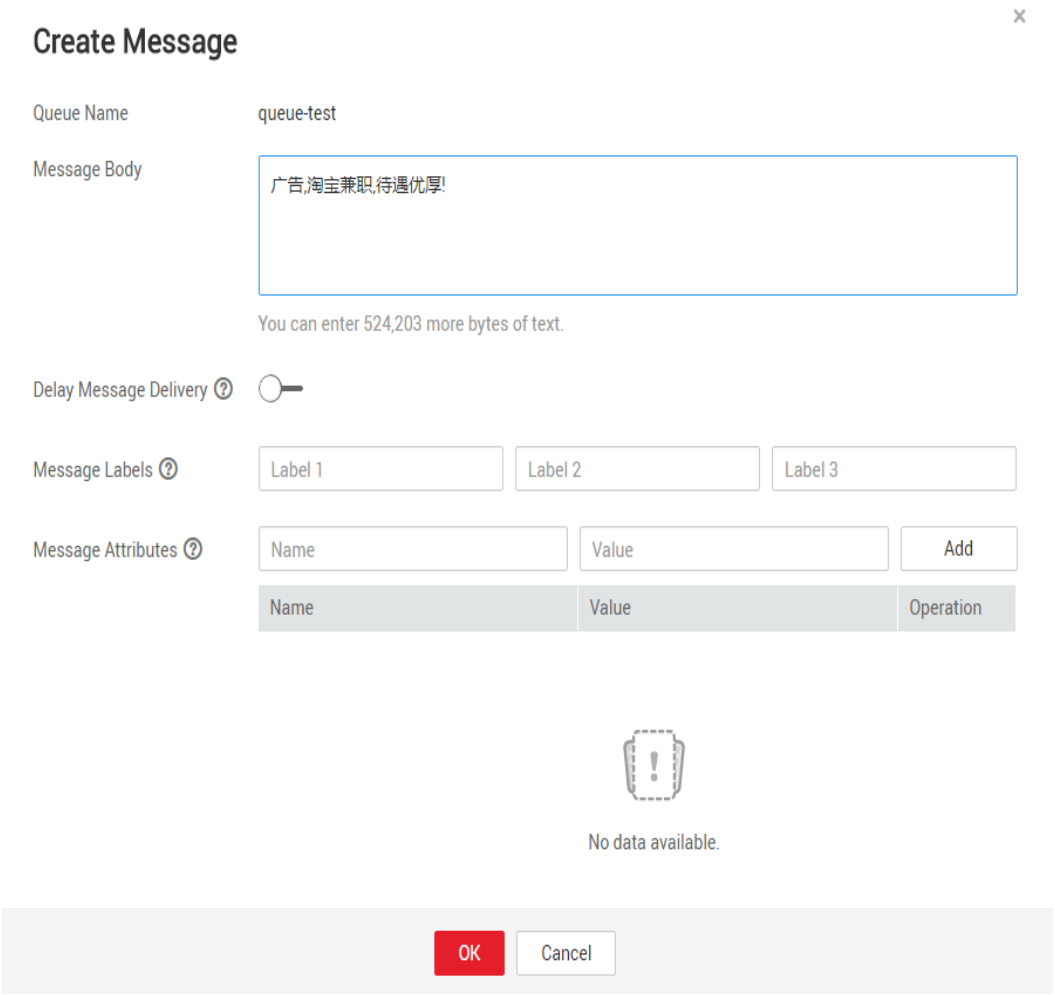
## Creating a Message

Perform the following steps to create a message:

1. Log in to the **DMS console**, and choose **Queue Manager** in the navigation pane.
2. Click **queue-test**, and then click **Create Message**.
3. In the **Create Message** dialog box, enter a message (such as "Well-paid opportunities for working part time") in the **Message Body** area, as shown in **Figure 3-12**.
4. Click **OK**.

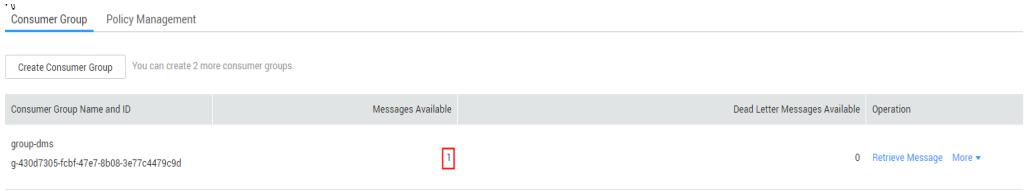
  

**Figure 3-12** Creating a message




After creating the message, you can see the number of messages that can be retrieved in the **group-dms** row, as shown in **Figure 3-13**.

**Figure 3-13** Number of messages that can be retrieved

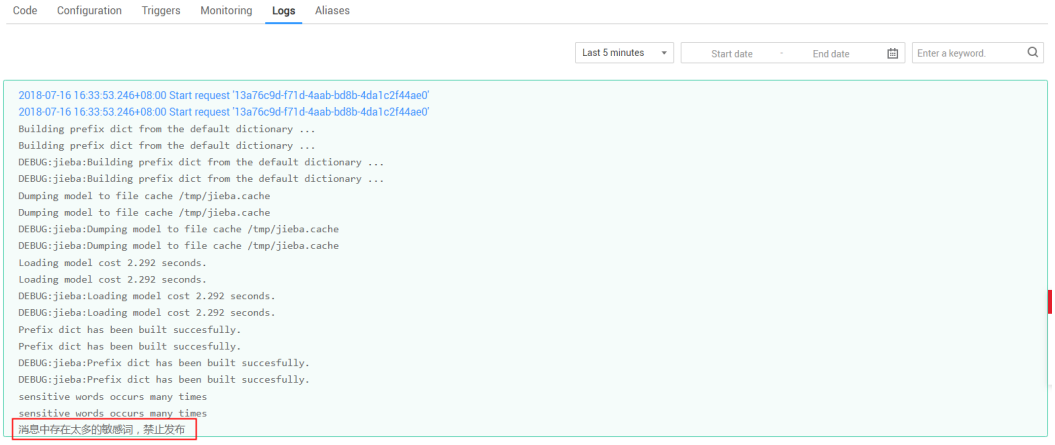



## Triggering the Function

When detecting the message, the trigger invokes the function. On the **fss_examples_message_filtering** page on the FunctionGraph console, as shown in **Figure 3-14**, you can view a function running log indicating that the message cannot be published because it contains too many sensitive words.

**Figure 3-14** Log information



# 4 Processing DIS Data

## 4.1 Introduction

The best practice for FunctionGraph guides you through DIS data processing based on a function.

### Scenarios

When using the Data Ingestion Service (DIS) to collect real-time Internet of Things (IoT) data streams, process the collected data, for example, convert its format, and then store the processed data into the CloudTable Service (CloudTable).

### Procedure

- Create a Virtual Private Cloud (VPC) and cluster.
- Build a data processing program and package the code.
- Create a function on the FunctionGraph console.
- Configure a DIS event to test the data processing function.

## 4.2 Preparation

This tutorial demonstrates how to convert the format of DIS data and store the converted data into CloudTable. To achieve this purpose, you need to create a VPC and then create a cluster on the CloudTable console.

Before creating a function, you must create an agency to delegate FunctionGraph to access DIS and CloudTable resources.

### Creating a VPC

**Step 1** Log in to the **VPC console** and click **Create VPC**.

**Step 2** Set the private cloud information.

For **Basic Information**, enter name **vpc-cloudtable**, and use the default settings of other parameters, as shown in **Figure 4-1**.

For **Subnet Settings**, use the default settings, as shown in **Figure 4-2**.

**Figure 4-1** Basic information

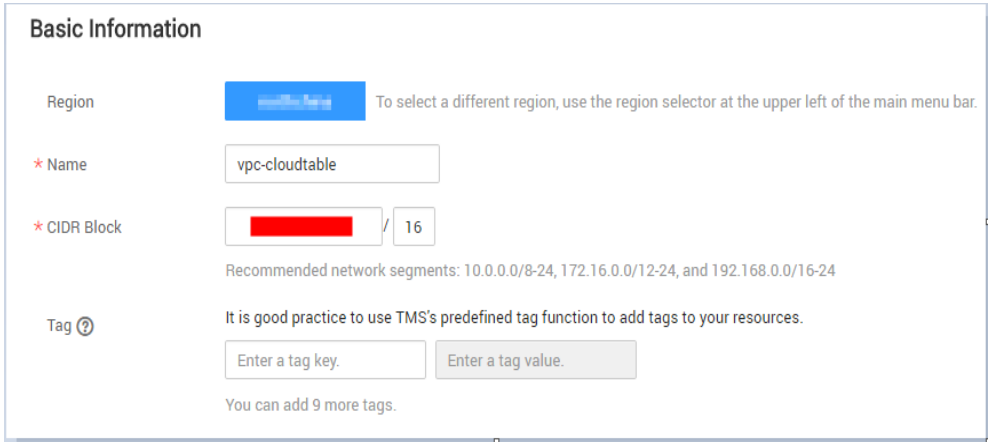
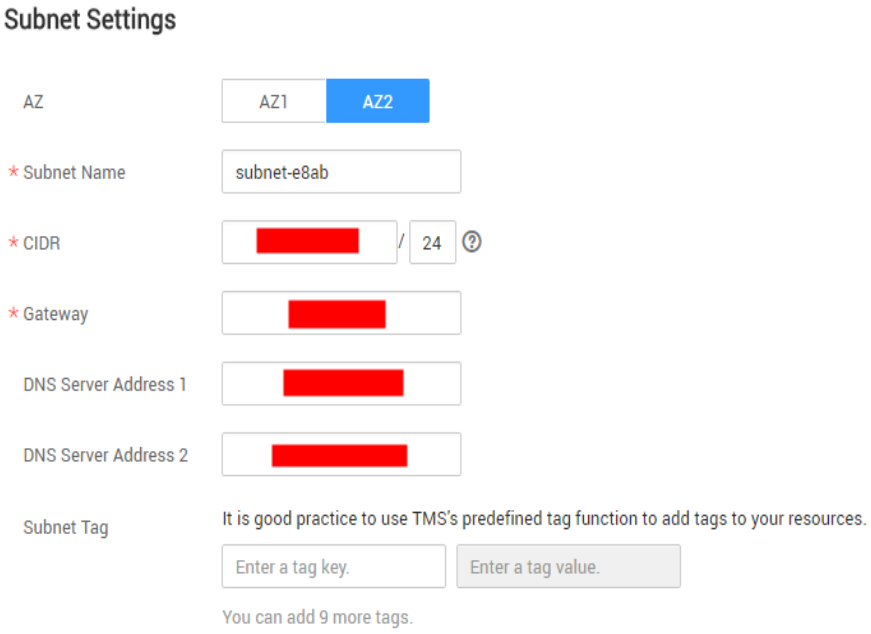



**Figure 4-2** Subnet settings




**Step 3** Confirm the configuration information and click **Submit**.

**----End**

## Creating a Cluster

**Step 1** Log in to the **CloudTable console**. On the **Cluster Mode** page, click **Buy Cluster**.

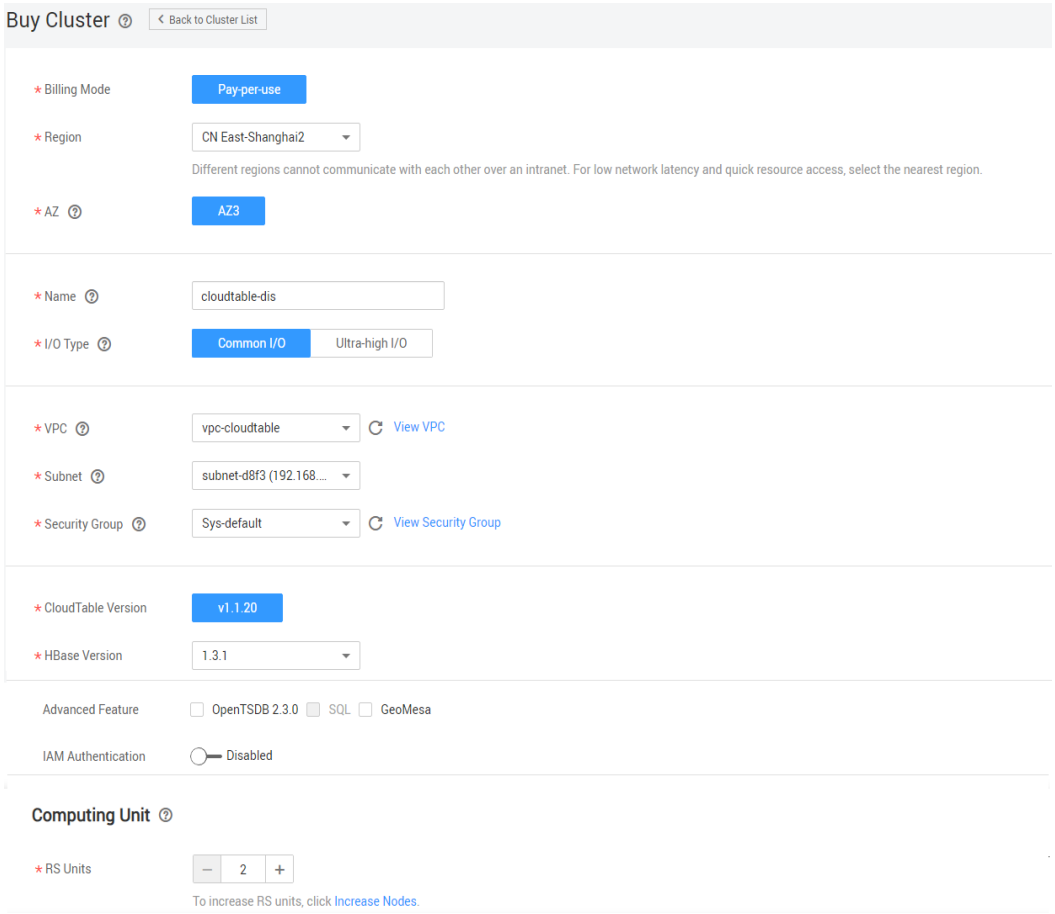
**Step 2** Set the cluster information.

- **Region**: Use the default region.
- **Name**: Enter "cloudtable-dis".
- **VPC**: Select **vpc-cloudtable** created in **Creating a VPC**.

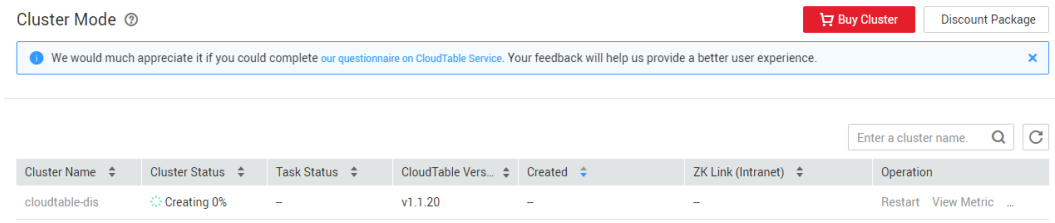
- Retain the default values for other parameters.

**Figure 4-3** Buying a cluster




**Step 3** Confirm the configuration information and click **Next**.

**Figure 4-4** Creating a cluster




**NOTE**

Creating a cluster takes a long time. You can check the creation progress according to **Figure 4-4**.

**----End**

## Creating an Agency

**Step 1** Log in to the **IAM console**, and choose **Agencies** in the navigation pane.

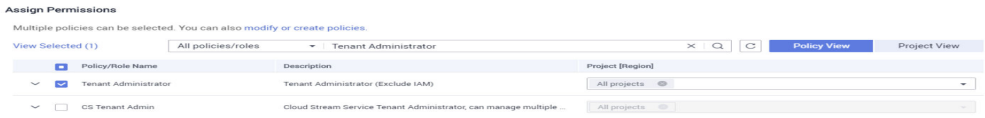
  

**Step 2** On the **Agencies** page, click **Create Agency**.

**Step 3** Set the agency information.

- For **Agency Name**, enter **DISDemo**.
- For **Agency Type**, select **Cloud service**.
- For **Cloud service**, select **FunctionGraph**.
- For **Validity Period**, select **Unlimited**.
- Click **Assign Permissions**. On the **Assign Permissions** page, select **Tenant Administrator**, as shown in **Figure 4-5**.

**Figure 4-5** Creating an agency




**Step 4** Click **OK**.

**----End**

# 4.3 Building a Program

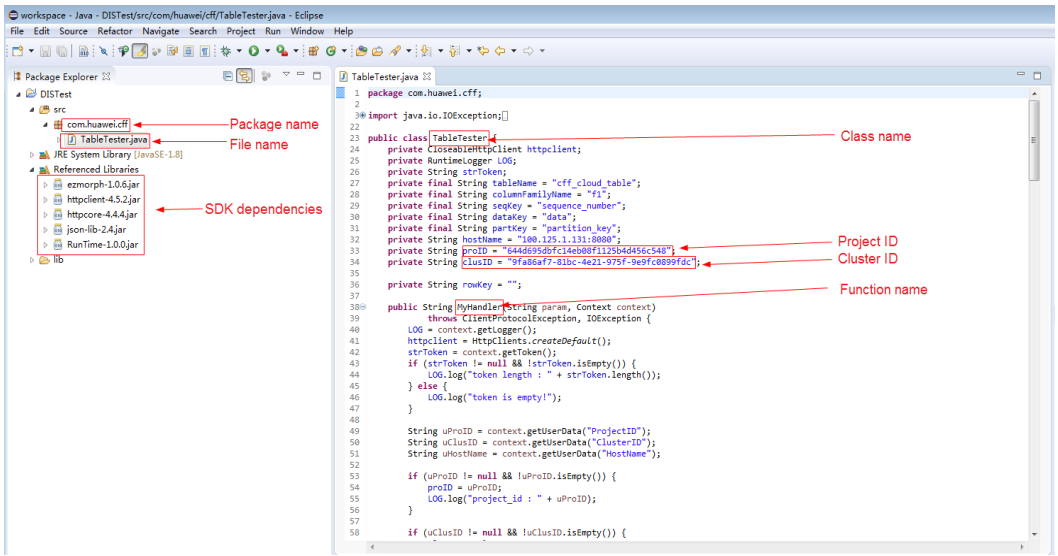
Download and use the **source code** and **program package** (including function dependencies) provided in this example for format conversion of DIS stream data.

## Creating a Project

This example uses a Java function to convert the format of DIS stream data. For details about function development, see **Developing Functions in Java**. The service code is not described.

Download the sample source code package **fss_examples_dis_cloudtable_src.zip**, decompress the file, and import it to Eclipse, as shown in **Figure 4-6**.

**Figure 4-6** Sample code



In the sample code, change the **proID** (project ID) and **clusID** (cluster ID), and save the changes.

To obtain the project ID, perform the following steps:

1. Under the current login account in the upper right corner, choose **My Credentials**, as shown in **Figure 4-7**.
2. Obtain the project ID in the project list, as shown in **Figure 4-8**.

**Figure 4-7** My Credentials

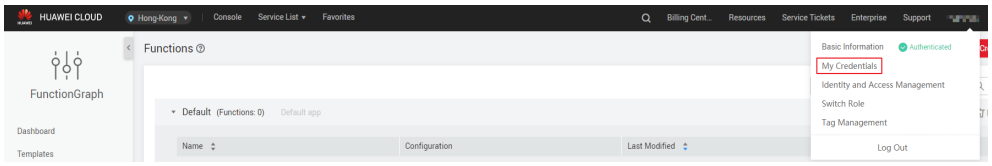
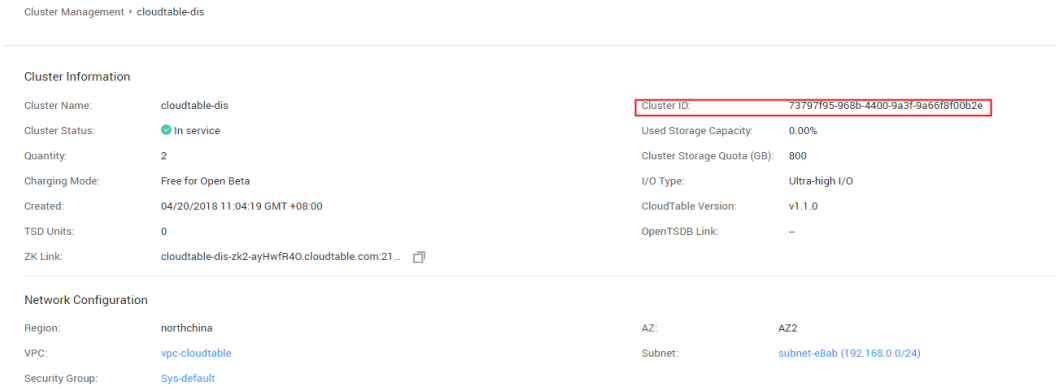



**Figure 4-8** Project ID


To obtain the cluster ID, perform the following steps:

1. Log in to the **CloudTable console**.
2. In the navigation pane, choose **Cluster Management**. Click cluster **cloudtable-dis** created in **Creating a Cluster**.
3. On the **cloudtable-dis** page that is displayed, find the cluster ID, as shown in **Figure 4-9**.

**Figure 4-9** Cluster ID




When creating a function on the FunctionGraph console, set a handler in the format of [*package name*].[*file name*].[*function name*], for example, **com.huawei.cff.TableTester.MyHandler** for the preceding code.

## Packaging the Code

Use Eclipse to package the code into a JAR file named **Table Tester.jar** according to the following figures.

**Figure 4-10** Exporting the code

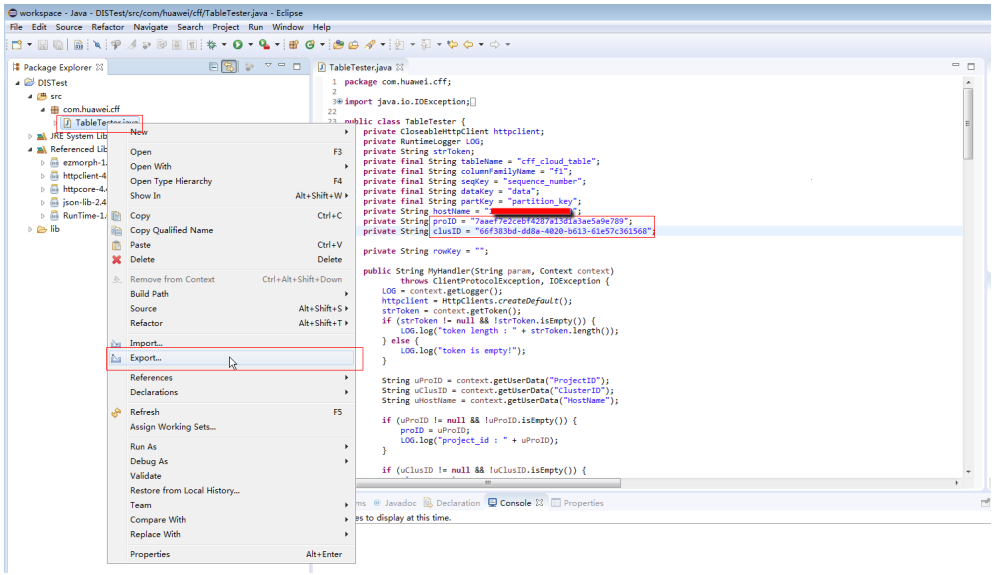


**Figure 4-11** Selecting a file type

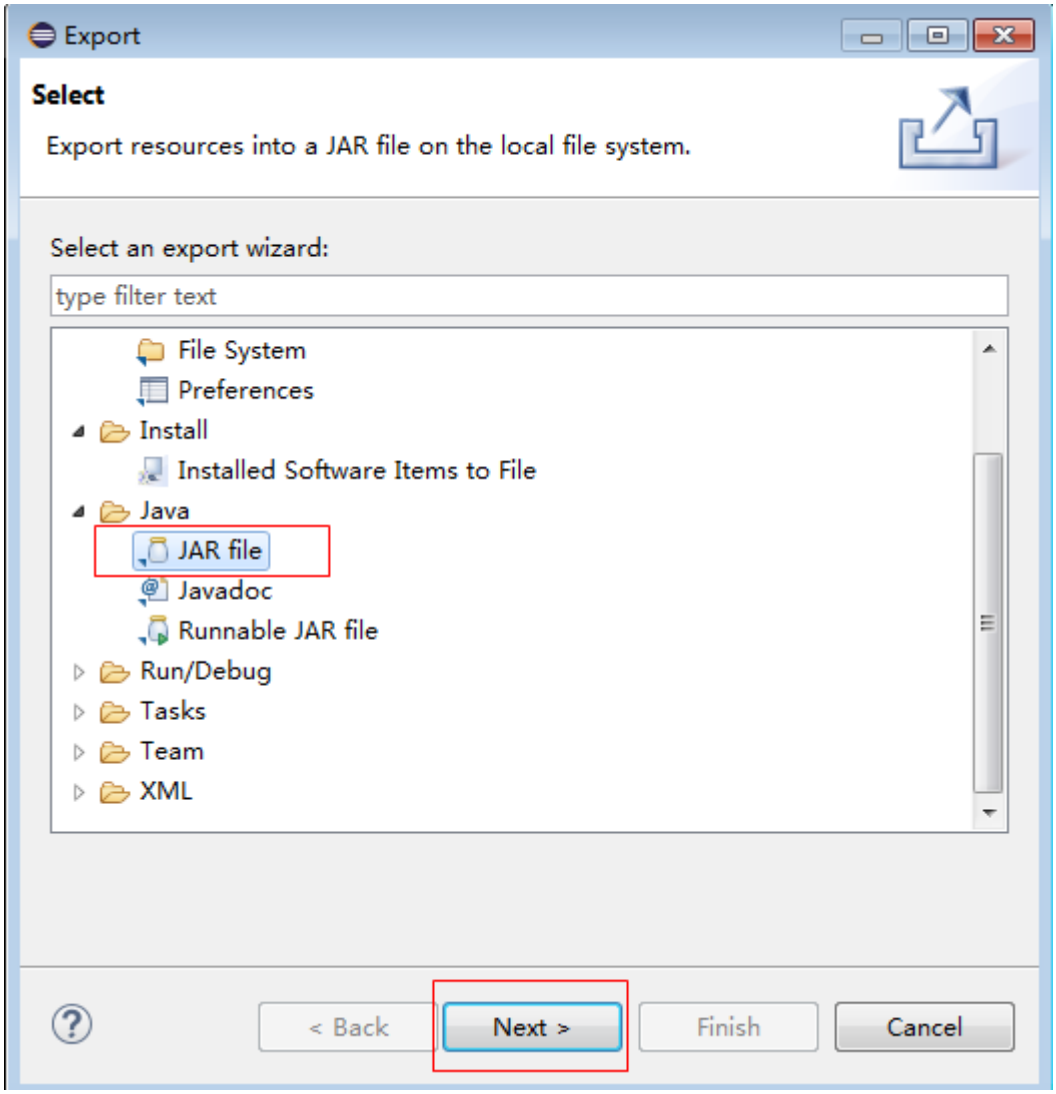
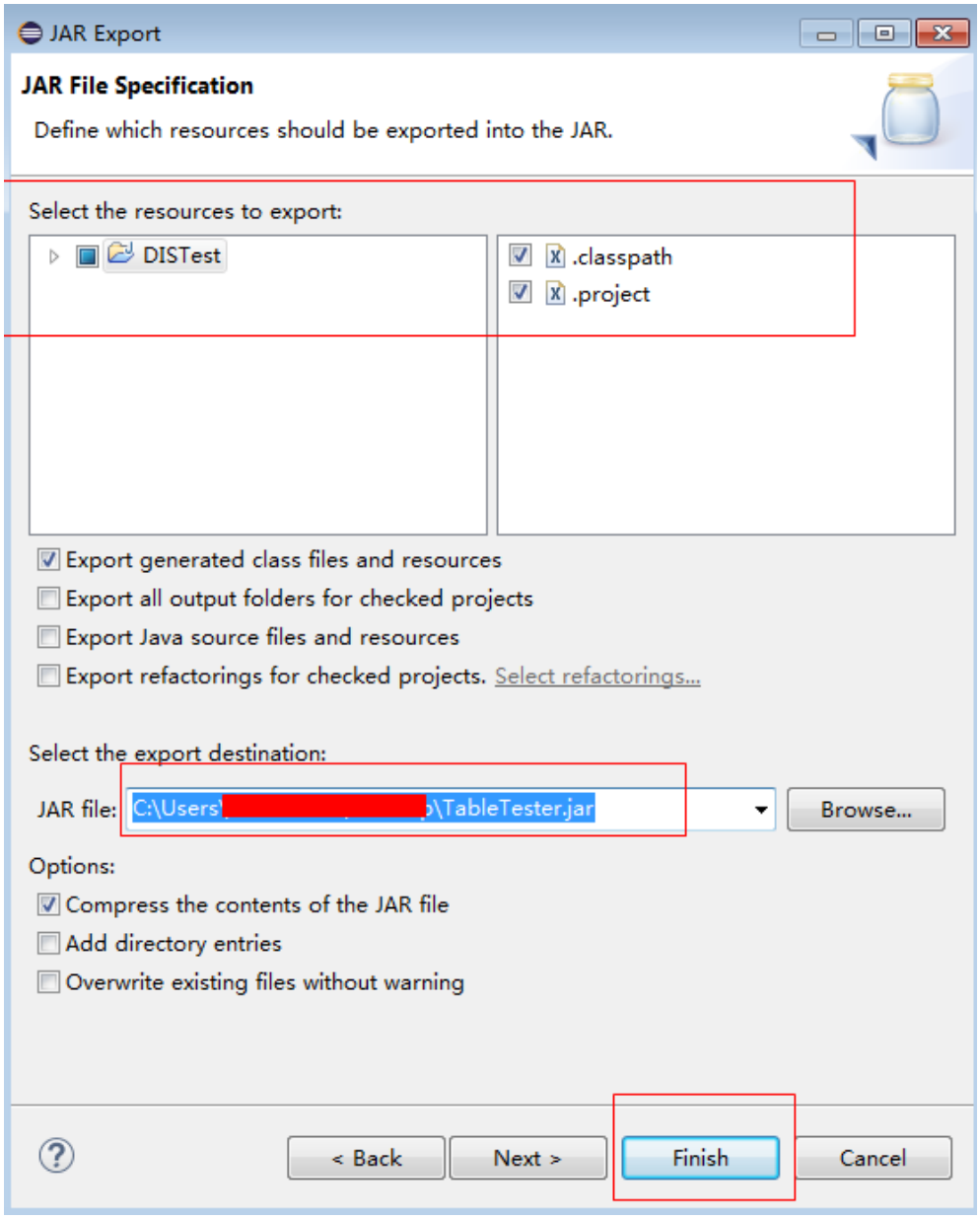


**Figure 4-12** Publishing the code file




Package the function dependencies by performing the following steps:

1. **Download program package** **fss_examples_dis_cloudtable.zip**, and decompress it, as shown in **Figure 4-13**.
2. Use **Table Tester.jar** to replace **DIS Test.jar**, as shown in **Figure 4-14**.
3. Package all of the files into **disdemo.zip**, as shown in **Figure 4-15**.

**Figure 4-13** File directory before replacement

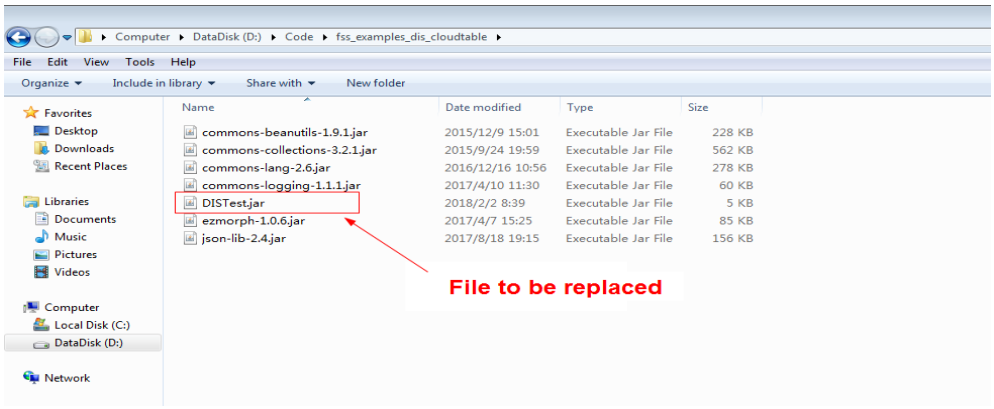



**Figure 4-14** File directory after replacement

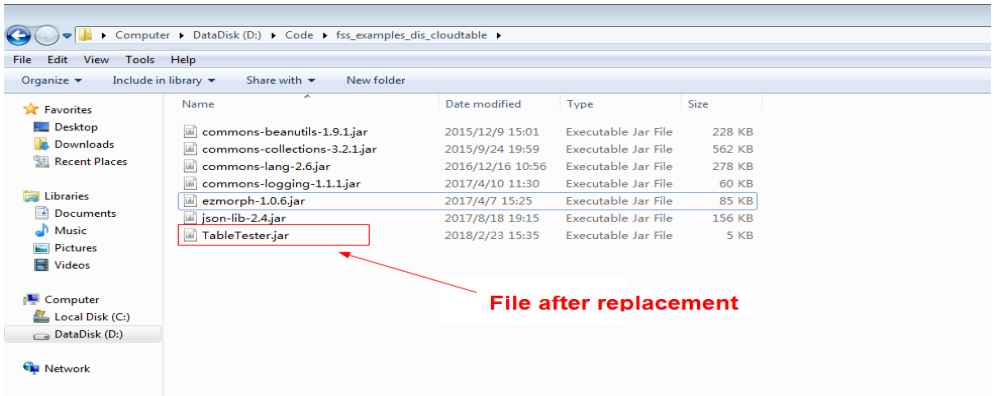
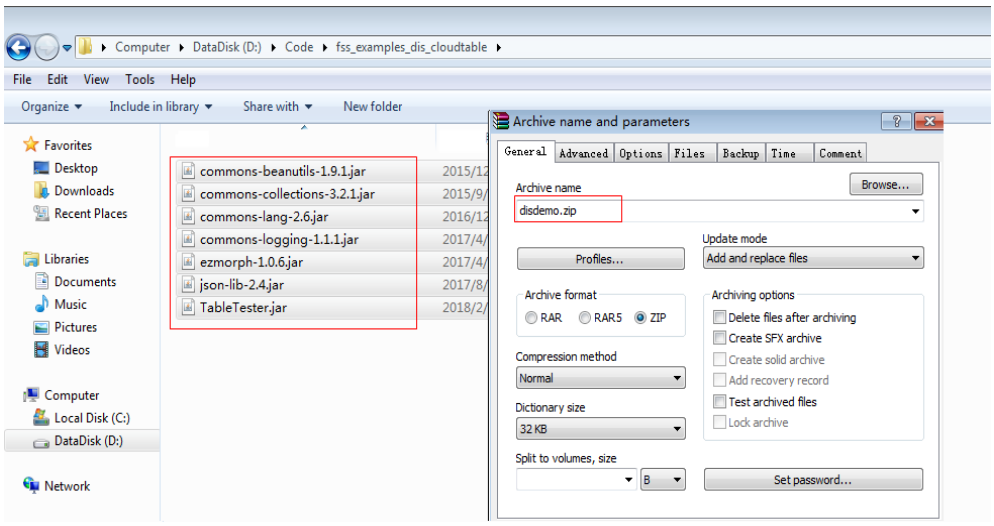



**Figure 4-15** Packaging the files




## Creating a Function

When creating a function, specify an agency to delegate FunctionGraph to access DIS and CloudTable resources.

**Step 1** Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane.

**Step 2** Click **Create Function**.

**Step 3** Set the function information, as shown in **Figure 4-16**.

For **Template**, select **Create from scratch**.

For **Function Name**, enter **DISDemo**.

For **App**, select **default**.

For **Agency**, select **DISDemo** created in **Preparation**.

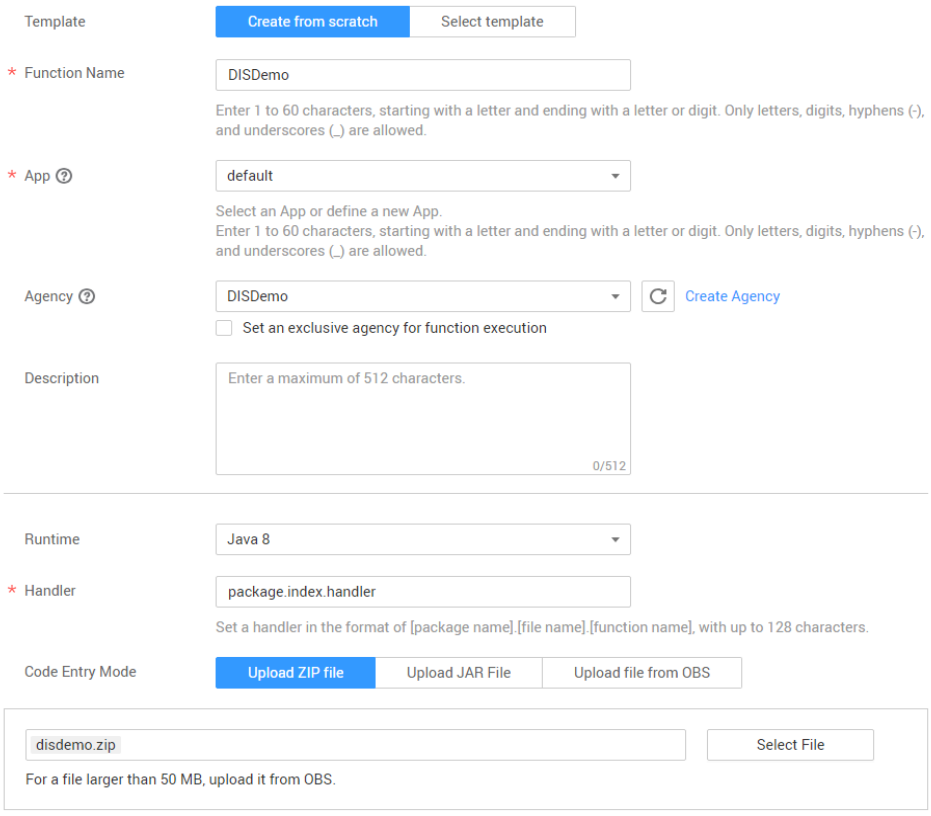
For **Runtime**, select **Java 8**.

For **Handler**, enter **com.huawei.cff.TableTester.MyHandler**.

For **Code Entry Mode**, select **Upload ZIP file**.

Upload code package **disdemo.zip** created in **Packaging the Code**.

**Figure 4-16** Creating a function




**Step 4** Click **Create Function**.

**----End**

## Modifying Function Configurations

After the function is created, the default memory is 128 MB, and the default timeout is 3s, which are insufficient for the data processing. Perform the following steps to modify the configurations.

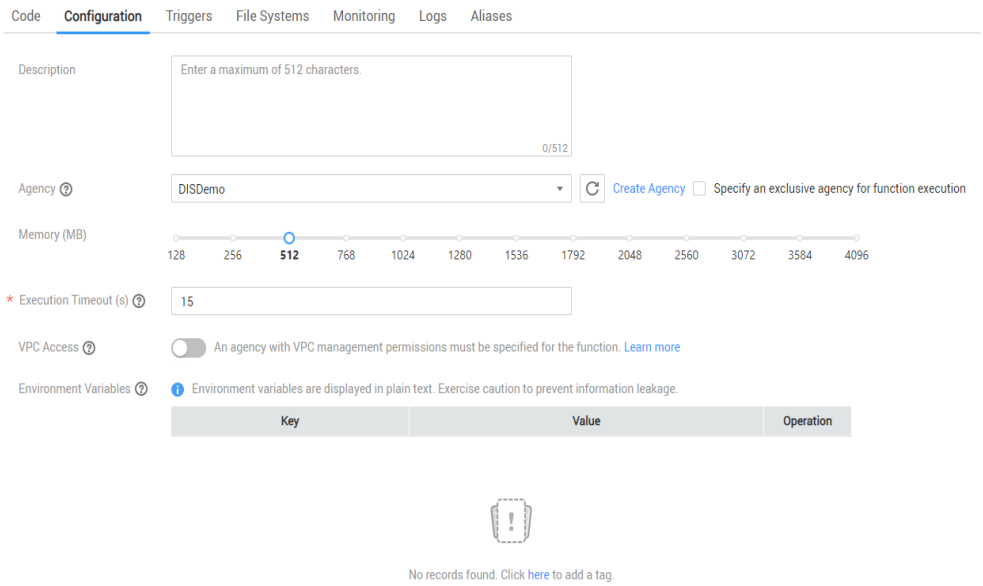
**Step 1** On the **DISDemo** page, click the **Configuration** tab and modify the configuration information, as shown in **Figure 4-17**.

For **Memory**, select **512**.

For **Timeout**, enter **15**.

Keep other parameters unchanged.

**Figure 4-17** Function configurations




**Step 2** Click **Save**.

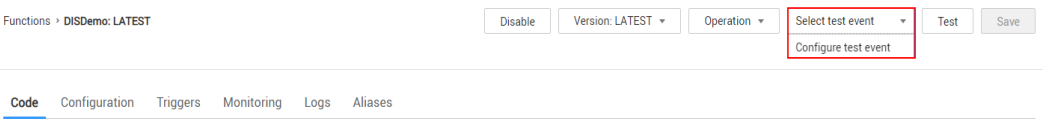
**----End**

# 4.4 Adding an Event Source

After creating the function, you can add an event source by creating a DIS trigger. Perform the following procedure:

**Step 1** On the **DISDemo** page, select **Configure test event**, as shown in **Figure 4-18**.

**Figure 4-18** Configuring a test event




**Step 2** In the **Configure Test Event** dialog box, set the test event information, as shown in **Figure 4-19**.

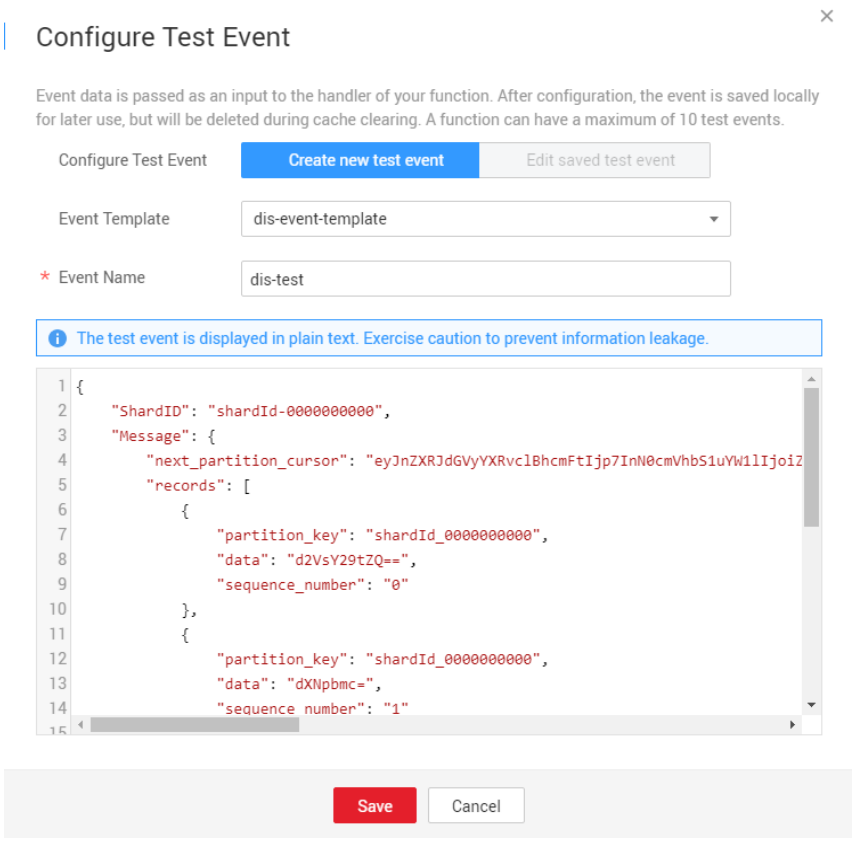
For **Configure Test Event**, select **Create new test event**.

For **Event Template**, select **dis-event-template**.

For **Event Name**, enter **dis-test**.

**Figure 4-19** Test event




**Step 3** Click **Save**.

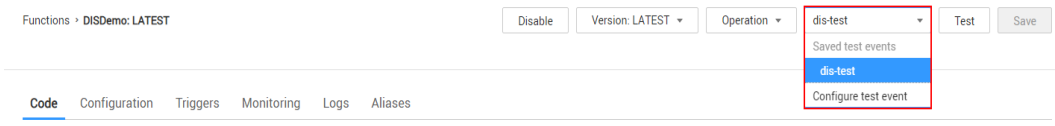
**----End**

# 4.5 Processing Data

Perform the following procedure to process simulated stream data:

**Step 1** On the **DISDemo** page, select test event **dis-test**, and click **Test** to test the function, as shown in **Figure 4-20**.

**Figure 4-20** Selecting a test event




**Step 2** After the function is executed successfully, check the logs shown in **Figure 4-21**. For all logs of the function, go to the **Logs** tab page.

**Figure 4-21** Function execution result


**----End**

# 5 Building an AI Application for Identifying Pornographic Images

## 5.1 Introduction

The best practice for FunctionGraph guides you through pornographic image identification based on functions.

### Objective

In this example, FunctionGraph interconnects with the Artificial Intelligence (AI) Service to identify pornographic images. You can create an APIG trigger to provide external API services for pornographic image identification.

To visually demonstrate how to identify pornographic images, this example provides a frontend page for you to upload images and view the identification results.

Using FunctionGraph to build AI applications has the following advantages:

- Simplifies the development process and focuses on service implementation.
- Flexibly interconnects with cloud services to improve development efficiency.
- Scales automatically and bills per use to prevent waste of resources.
- Requires no maintenance and provides high availability at low costs.

### Scenarios

This example uses the following cloud services:

- Content Moderation: provides an API for image identification. For details, see **Content Moderation**.
- FunctionGraph: builds a pornographic image identification function. For details, see the official website of FunctionGraph.
- API Gateway: provides an API for pornographic image identification. For details, see **API Gateway**.
- OBS: hosts frontend code to provide a static website. For details, see **Object Storage Service**.

**Figure 5-1** shows the simulated image identification page.

**Figure 5-1** Image identification

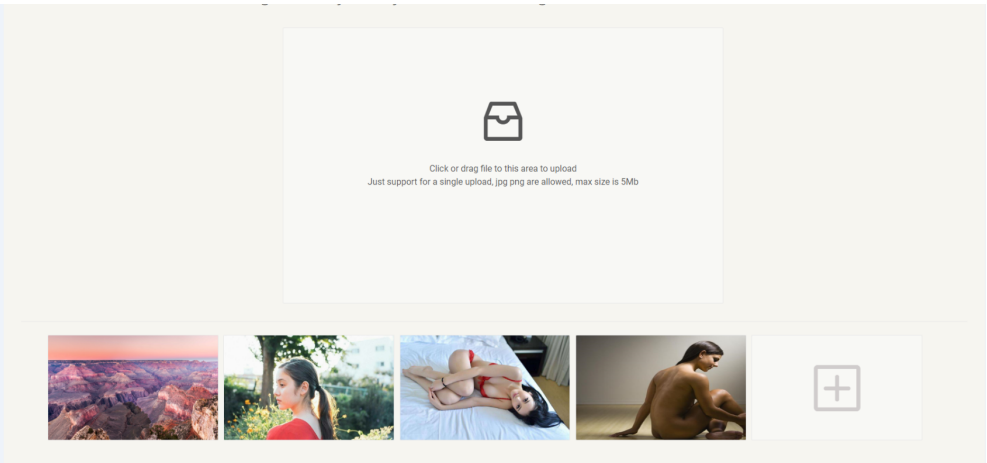
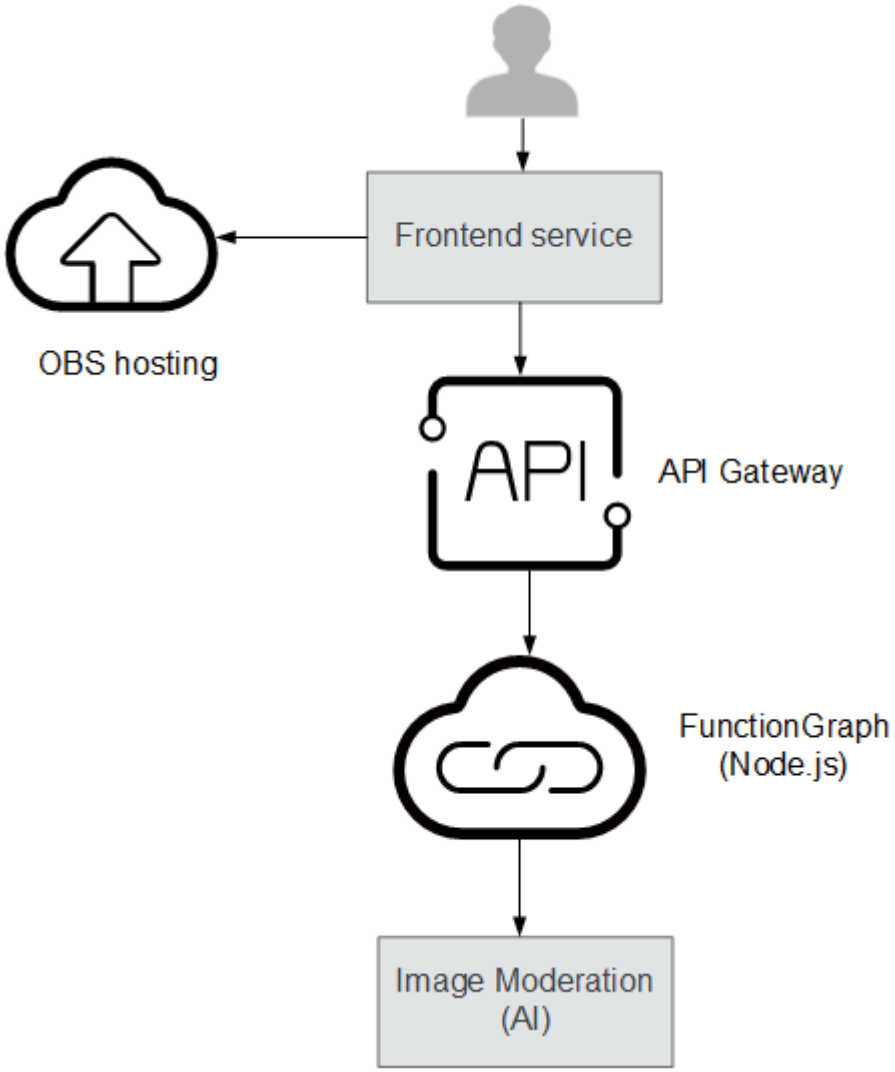



**Figure 5-2** illustrates the principle of image identification.

**Figure 5-2** Principle of image identification




- OBS hosts frontend code to provide a static website. A frontend page is provided for uploading images.
- After an image is uploaded, the frontend code calls the API provided by API Gateway to trigger the backend function.
- During execution, the function calls the image moderation API.
- The Content Moderation service determines whether the uploaded image is acceptable and returns the identification result.

## Resource Preparation

- Apply for the Content Moderation service.
- Apply for the API Gateway service.
- Apply for the FunctionGraph service.
- Apply for the OBS service.

### Procedure

1. Create the following resources:
   - One agency: enables FunctionGraph to access API Gateway, Image Moderation, and OBS resources.
   - One API group: a collection of APIs. API providers classify APIs into different API groups and manage APIs by API group.
   - One OBS bucket: hosts the frontend program to provide a static website.
2. Build a backend function.
   - Create a backend function to implement image identification.
   - Create an APIG trigger for the backend function and provide an API to invoke the backend function.
3. Build a frontend function.
   - Create a frontend function to provide a frontend page.
   - Create an APIG trigger for the frontend function and provide an API to invoke the frontend function.
4. Upload an image and check whether the image is acceptable.

### Price Description

- The Image Moderation, API Gateway, and FunctionGraph services used in this example are under an OBT, and can be used for free during this period. You will be charged only after the OBT is ended. For details, see the *Product Introduction* of each service.
- For OBS, you will be charged based on the following rules. For details, see **OBS Pricing Details**.

## 5.2 Preparation

### Creating an Agency

An agency is an entrusting relationship created by an administrator to grant specific permissions to other accounts. After an entrusting relationship is established, the administrator of the entrusted enterprise can manage cloud resources on behalf of the entrusting enterprise by switching the agency. This ensures secure and efficient service management.

You must create an agency to enable FunctionGraph to access API Gateway, Image Moderation, and OBS resources.

**Step 1** Log in to the **IAM console**, and choose **Agencies** in the navigation pane.

**Step 2** On the **Agencies** page, click **Create Agency**.

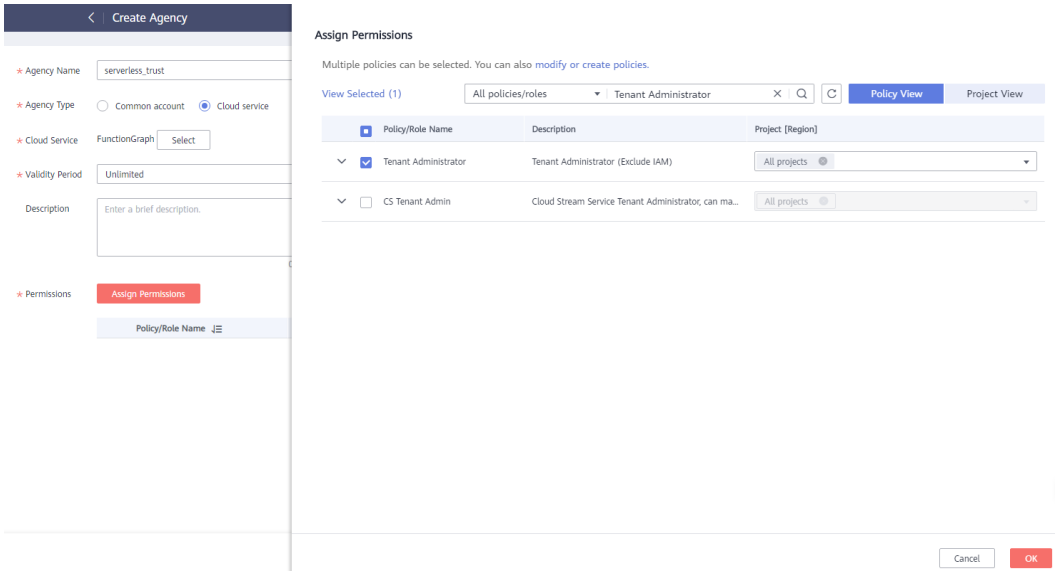
**Step 3** Set the agency information.

- For **Agency Name**, enter **serverless_trust**.
- For **Agency Type**, select **Cloud service**.
- For **Cloud service**, select **FunctionGraph**.
- For **Validity Period**, select **Unlimited**.

- Click **Assign Permissions**. On the **Assign Permissions** page, select **Tenant Administrator**.

**Figure 5-3** Creating an agency




**NOTE**

Users with the Tenant Administrator permission can perform any operations on all cloud resources of the enterprise.

**Step 4** Click **OK**.

**----End**

## Creating an API group

An API group is a collection of APIs used for the same type of services. One API group can be considered as a service. You can manage APIs by API group.

When creating an APIG trigger, select an API group in the same region as FunctionGraph. Create an API group by following the procedure described in this section.

**Step 1** Log in to the **API Gateway console**, and choose **Shared Gateway** > **API Publishing** > **API Groups**.

**Step 2** Click **Create API Group**.

**Step 3** Set the API group information, as shown in **Table 5-1**.

**Table 5-1** Information required for creating an API group

| Parameter | Description |
| --- | --- |
| Name | Enter **function**. |

| Parameter | Description |
| --- | --- |
| Description | This example does not require a description. |

**Step 4** Click **OK**.

**----End**

## Creating an OBS bucket

Create an OBS bucket to host the frontend program and provide a static website.

**Step 1** Log in to the **OBS console**, and click **Create Bucket**.

**Step 2** Set the bucket information according to **Figure 5-4**.

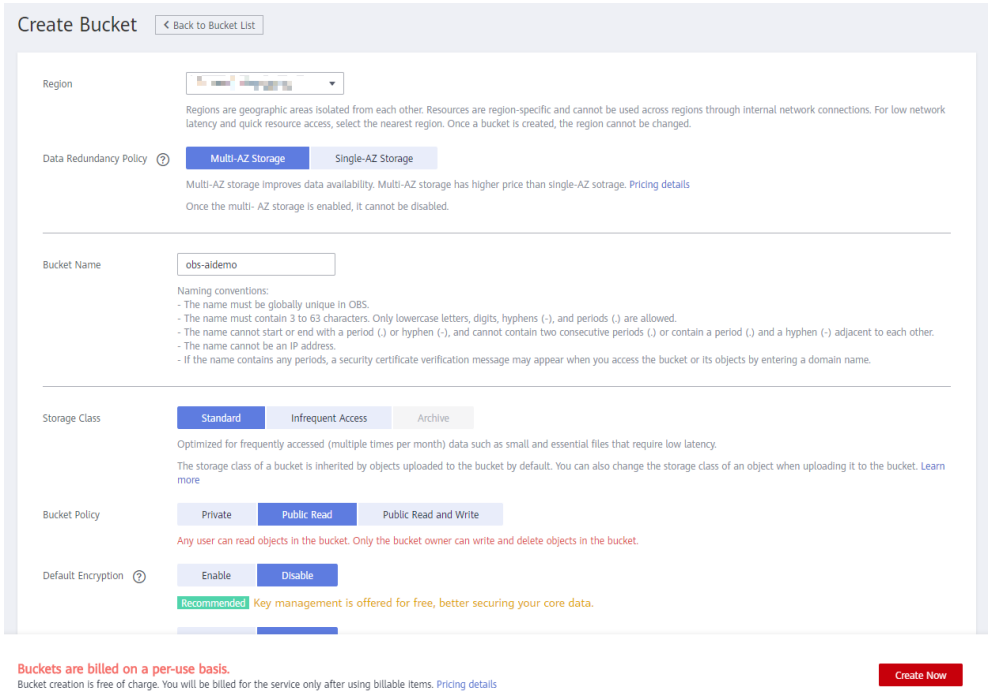
For **Region**, select a region.

For **Bucket Name**, enter **obs-aidemo**.

For **Storage Class**, select **Standard**.

For **Bucket Policy**, select **Public Read**.

**Figure 5-4** Creating a bucket




**----End**

# 5.3 Building a Backend Function

## Creating a Backend Function

**Step 1** Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane.

**Step 2** Click **Create Function**.

**Step 3** Set the function information, as shown in **Figure 5-5**.

- For **Template**, select **Create from scratch**.
- For **Function Name**, enter **AIDemo**.
- For **App**, select **default**.
- For **Agency**, select **serverless_trust**.

  The selected agency must have permissions to access API Gateway, Image Moderation, and OBS resources. For details on how to create an agency, see **Creating an Agency**.
- For **Runtime**, select **Node.js 6.10**.
- For **Handler**, enter **index.handler**.
- For **Code Entry Mode**, select **Edit code inline**.
- Enter the following code in the code box:

```
const https = require('https');
const fs = require('fs');
const ObsClient = require('esdk-obs-nodejs');    //References an OBS SDK.
let PAYLOAD = {};
let CONTEXT = null;
const PATH = '/tmp/image.png'
exports.handler = function (event, context, callback) {
    CONTEXT = context;

    //Cross-origin verification
    if (event.httpMethod === "OPTIONS") {
        console.log('OPTIONS request, pass.');
        const result = {
            body: '',
            headers: {
                "Content-Type":"application/json",
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type,Accept",
                "Access-Control-Allow-Methods": "GET,POST,PUT,DELETE"
            },
            statusCode: 200,
            isBase64Encoded: false
        };
        callback(null, result);
        return;
    }

    new Promise(function(resolve, reject) {
        var token = context.getToken();
        const options = {
            hostname: 'moderation.cn-north-1.myhuaweicloud.com',
            port: 443,
```

```
            path: '/v1.0/moderation/image',
            method: 'POST',
            headers: {
                'Content-Type': 'application/json;charset=utf8',
                'X-Auth-Token': token
            }
        };
        PAYLOAD = constructPayload(event);
        console.log("Sending request to AIS...")
        const req = https.request(options, (res) => {
            var status = res.statusCode
            console.log("status:",  status)
            res.on('data', (d) => {
                resolve(d.toString());
            });
        });

        req.on('error', (e) => {
            console.error(e);
        });
        req.write(JSON.stringify(PAYLOAD));
        req.end();
    }).then(function(value) {
        // check image and save if porn
        const resultObj = JSON.parse(value);
        const bucket = context.getUserData('BUCKET');

        const apigBody = {
          result: resultObj.result.detail.porn,
          suggestion: resultObj.result.suggestion,
          eiInfo: resultObj.result,
        };
        saveImage(PAYLOAD.image, resultObj.suggestion || '', bucket).then((err) => {
            // return apig result
            const result = {
                body: JSON.stringify(apigBody),
                headers: {
                    "Content-Type":"application/json",
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type,Accept",
                    "Access-Control-Allow-Methods": "GET,POST,PUT,DELETE"
                },
                statusCode: 200,
                isBase64Encoded: false
            };
            callback(null, result);
        });
    });
}
function constructPayload(event) {
    if (event.httpMethod != "POST") {
        return {
            "image": "",
            "url": "https://bjbucket.obs.myhuaweicloud.com/70923411.jpg",//Example URL.
Replace it with the actual URL.
            "categories": [
                "porn"
            ],
            "threshold" : ""
        }
    }
    const bodyStr = new Buffer(event.body, 'base64').toString('ascii');
```

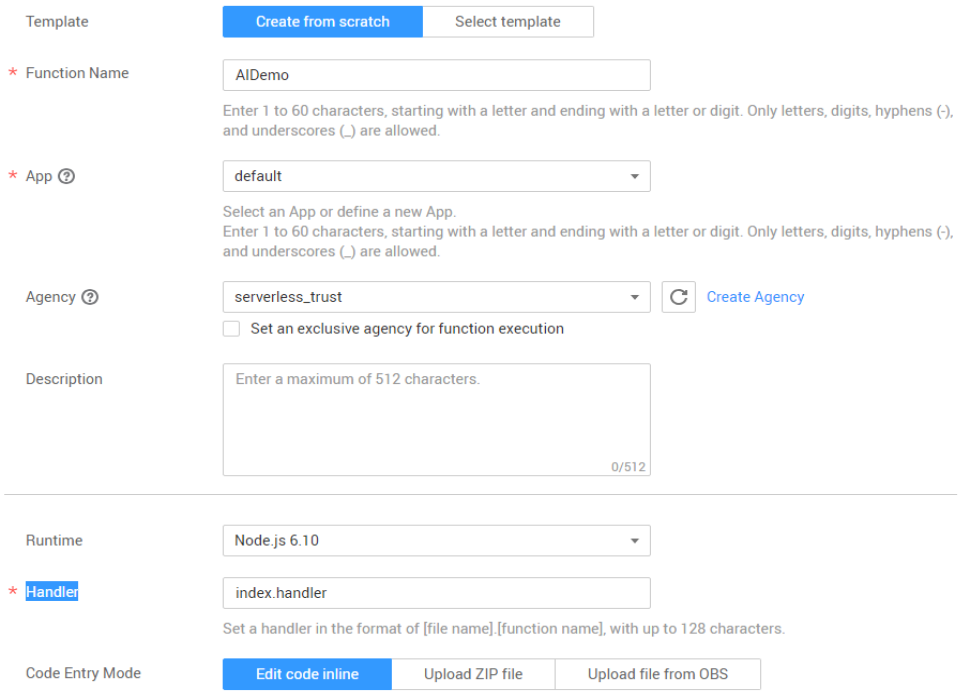
  

```
    const body = JSON.parse(bodyStr);
    if (body.url != null && body.url.trim() != "") {
        return {
            "image": "",
            "url": body.url,
            "categories": [
                "porn"
            ],
            "threshold" : ""
        }
    } else {
        return {
            "image": body.image,
            "url": "",
            "categories": [
                "porn"
            ],
            "threshold" : ""
        };
    }
}

function saveImage(image, imageInfo, bucket) {
    return new Promise((resolve, reject) => {
        if (/*imageInfo !== 'pass'*/false) {
            console.log('>>>>>>>not pass, start save image');
            writeImage(image).then((err) => {
                if (!err) {
                    uploadImage(PATH, bucket).then((err) => {
                        resolve(err);
                    });
                } else {
                    resolve(err)
                }
            });
        } else {
            resolve(null);
        }
    });
}


function writeImage(image) {
    return new Promise((resolve, reject) => {
        const imageBuffer = new Buffer(image, 'base64');
        fs.writeFile(PATH, imageBuffer, function(err) {
            console.log('>>>>>>>>>>write file:', err);
            resolve(err);
        });
    });
}

function uploadImage(path, bucket) {
    return new Promise((resolve, reject) => {
        // upload path to bucket;
        const client = buildOBSClient(CONTEXT);
        client.putObject({
            Bucket: bucket,
            Key: `image-${(new Date()).valueOf()}.png`,
            SourceFile: path
        }, (err, result) => {
            if (err) {
```

```
            console.error('Upload Error-->' + err);
        } else {
            console.log('Upload Status-->' + result.CommonMsg.Status + '. uploading file
OK!');
        }
        resolve(err);
    });
  });
}

function buildOBSClient(context) {
  // Obtains a temporary AK and SK to access OBS. An agency is required to access IAM.
  const my_access_key = context.getAccessKey();
  const my_secret_access_key = context.getSecretKey();
  console.log('begin create obs client!');
  return new ObsClient({
    access_key_id: my_access_key,
    secret_access_key: my_secret_access_key,
    server: 'obs.cn-north-1.myhuaweicloud.com'//Example. Replace it with the actual
value.
  });
}
```

**Figure 5-5** Function configurations




**Step 4** In the right pane of the page, review the function configuration and billing information, and click **Create Function**.

**----End**

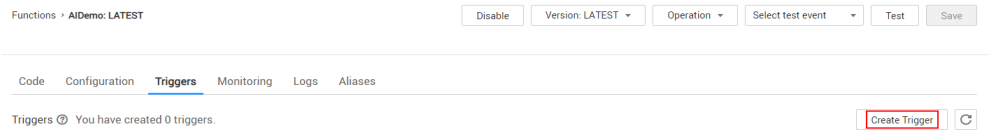
## Creating an APIG Trigger for the Backend Function

**Step 1** Log in to the **FunctionGraph console**, choose **Functions**, and click **AIDemo** to go to the function details page.

**Step 2** Click the **Triggers** tab and click **Create Trigger**, as shown in **Figure 5-6**.

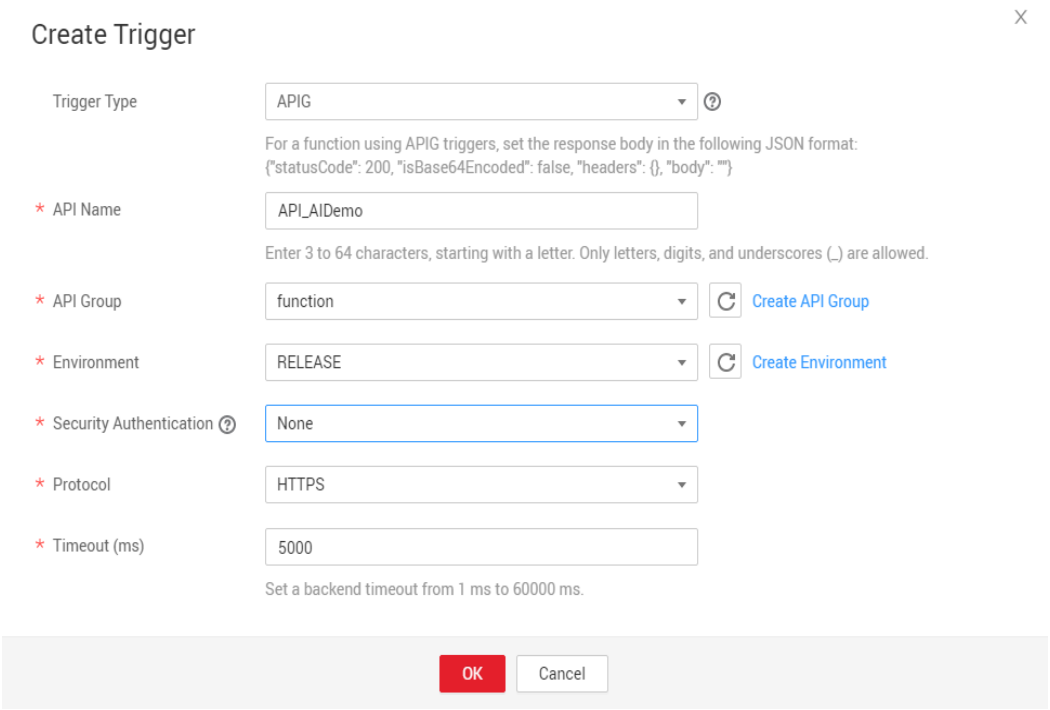
**Figure 5-6** Creating a trigger




**Step 3** Select **APIG** for **Trigger Type**, and set the trigger information, as shown in **Figure 5-7**.

- For **API Name**, enter **API_AIDemo**.
- For **API Group**, select **function**. For details on how to create an API group, see **Creating an API group**.
- For **Environment**, select **RELEASE**.
- For **Security Authentication**, select **None**.
- For **Protocol**, select **HTTPS**.
- For **Timeout (ms)**, enter **5000**.

**Figure 5-7** Creating an APIG trigger




**NOTE**

After you create an APIG trigger for the **RELEASE** environment, an API is generated and published on the API Gateway console. The API can be directly called by the function.

**Step 4** Click **OK**.

NOTE

The invocation address of the APIG trigger for the backend function is **https://d237af6bc4214cc383b5f2b40a3af234.apigw.cn-north-1.huaweicloud.com/AIDemo**. You will use this address when building a frontend function.

**----End**

# 5.4 Building a Frontend Function

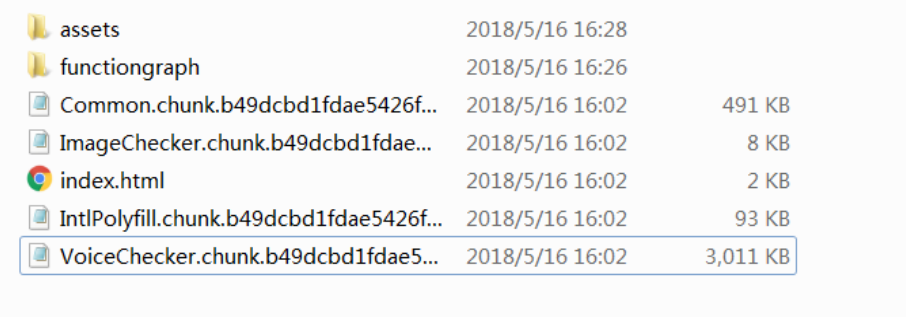
You can deploy a frontend page on OBS as a static resource, and use OBS to host your static website with a customized domain name.

You can download and use the frontend program package provided in this example.

## Downloading the Program Package

Download the **frontend program package** **functiongraph-demo.zip**, and decompress it, as shown in **Figure 5-8**.

**Figure 5-8** Program package directory




In the **/functiongraph/assets/config/apis.json** file, change the value of **checkImage** into the URL (**https://d237af6bc4214cc383b5f2b40a3af234.apigw.cn-north-1.huaweicloud.com/AIDemo**) of the APIG trigger created in **Creating an APIG Trigger for the Backend Function**, as shown in **Figure 5-9**.

**Figure 5-9** Changing the value of **checkImage**

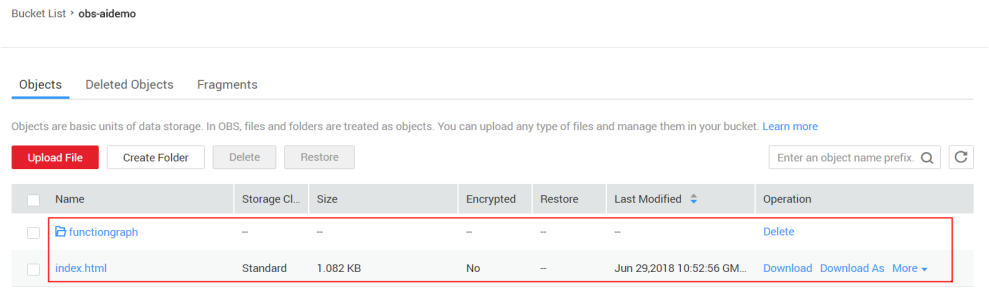
```
"checkImage": "https://d237af6bc4214cc383b5f2b40a3af234.apigw.cn-north-1.huaweicloud.com/AIDemo"
```

## Uploading the Code Files to an OBS Bucket

Log in to the **OBS console**, click bucket **obs-aidemo** created in **Creating an OBS bucket**, and upload the code files to the bucket, as shown in **Figure 5-10**.

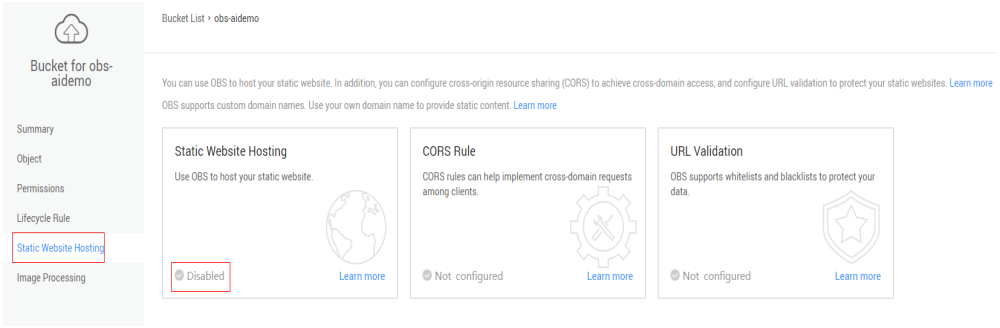
**Figure 5-10** Frontend program package directory




## Setting Static Website Hosting

1. On the **obs-aidemo** page, choose **Static Website Hosting**, and click **Disabled**, as shown in **Figure 5-11**.

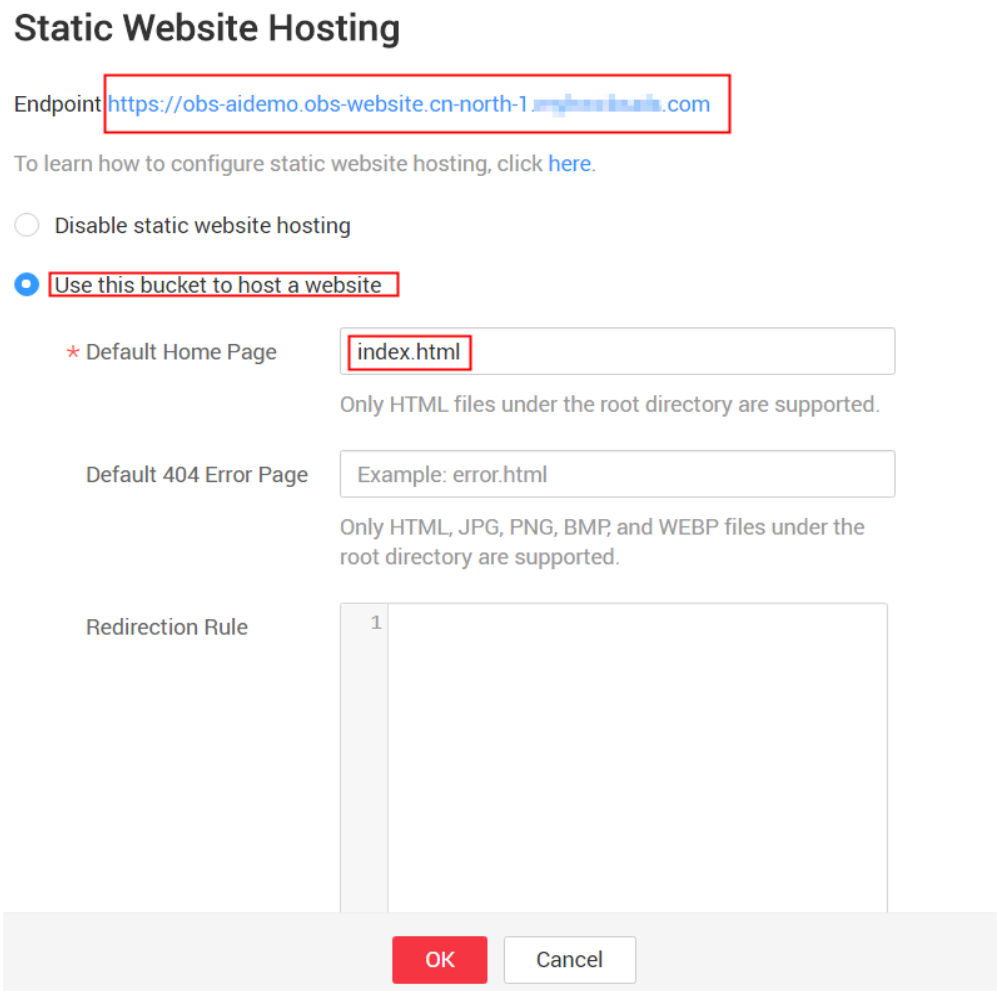
   **Figure 5-11** Static website hosting

2. In the **Static Website Hosting** dialog box, set the hosting information according to **Figure 5-12**.
   - Click **Use this bucket to host a website**.
   - For **Default Home Page**, enter **index.html**.
   - Leave other parameters blank.
   - Record the static website address, for example: **https://obs-aidemo.obs-website.cn-north-1.myhuaweicloud.com**.

**Figure 5-12** Static website hosting configurations

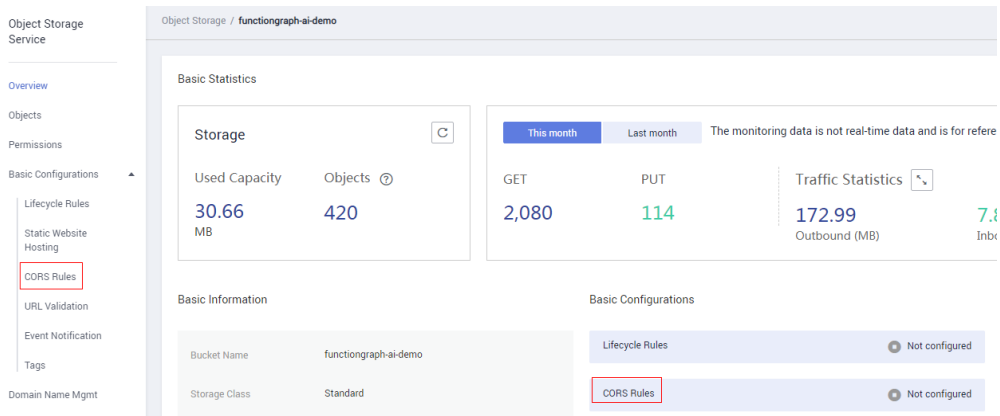



3. Click **OK**.

## Configuring an OBS CORS Rule

1. On the CORS rule page of the **obs-aidemo** bucket, click **CORS Rules**. The CORS rule configuration page is displayed, as shown in **Figure 5-13**.

**Figure 5-13** Configuring an OBS CORS rule



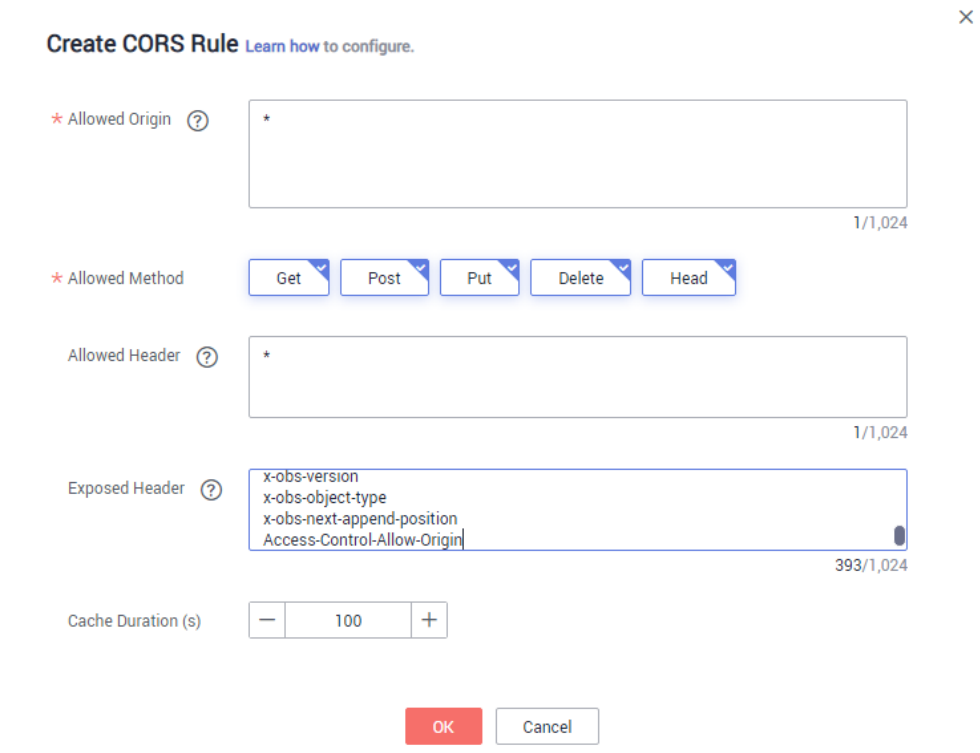

  

2. On the CORS rule configuration page, click **Create**. The CORS rule creation page is displayed, as shown in **Figure 5-14**.
   - For **Allowed Origin**, enter an asterisk (*).
   - For **Allowed Method**, select all methods.
   - For **Allowed Header**, enter an asterisk (*).
   - For **Exposed Header**, enter the following headers.

```
ETag
x-obs-request-id
Content-Type
Content-Length
Cache-Control
Content-Disposition
Content-Encoding
Content-Language
Expires
x-obs-id-2
x-reserved-indicator
x-obs-version-id
x-obs-copy-source-version-id
x-obs-storage-class
x-obs-delete-marker
x-obs-expiration
x-obs-website-redirect-location
x-obs-restore
x-obs-version
x-obs-object-type
x-obs-next-append-position
Access-Control-Allow-Origin
```

   - For **Cache Duration (s)**, enter **100**.

**Figure 5-14** Creating a CORS rule



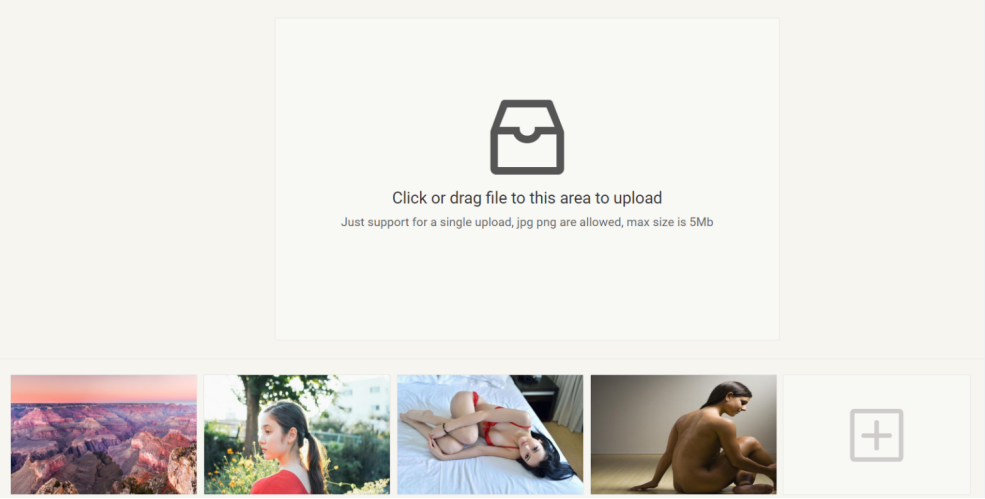
### Enabling API CORS

Log in to the **API Gateway console** and enable CORS for **API_AIDemo** created in **Step 3**. For details, see **Enabling CORS**.

## 5.5 Image Identification

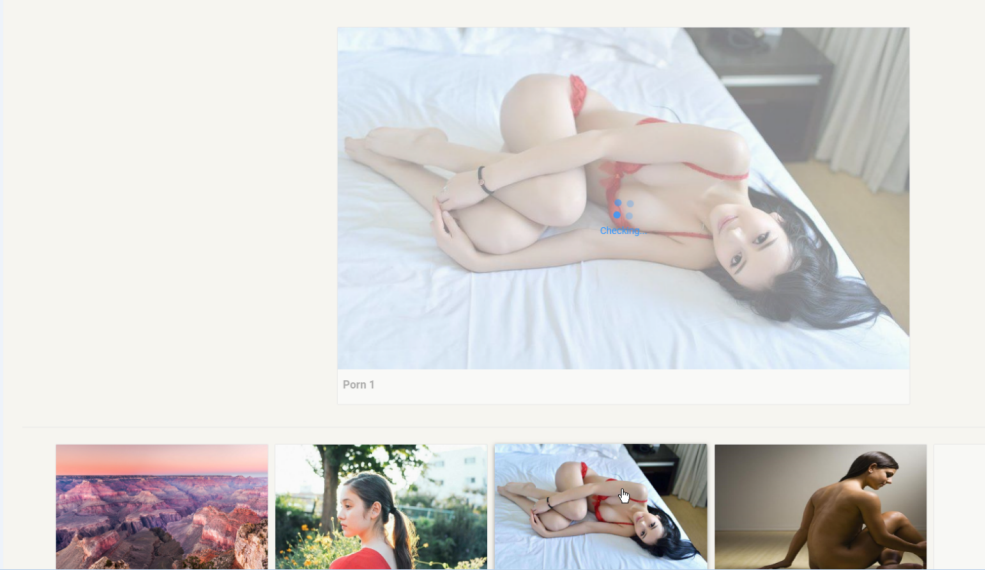
1. In the address bar of the browser, enter the website address **https://obs-aidemo.obs-website.cn-north-1.myhuaweicloud.com** obtained in **Building a Frontend Function**, as shown in **Figure 5-15**.

   **Figure 5-15** Image identification page

2. Upload an image to start identification, as shown in **Figure 5-16**.

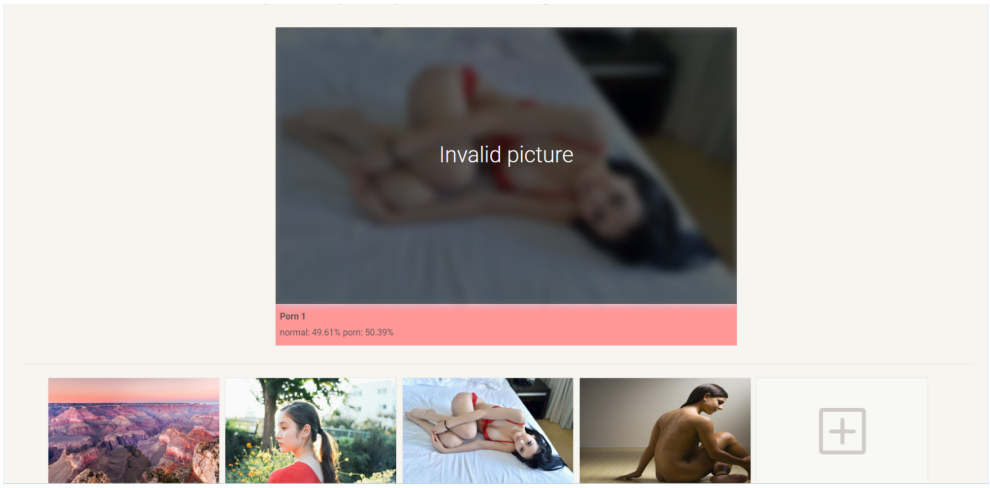
   **Figure 5-16** Uploading an image

3. Check the identification result, as shown in **Figure 5-17**. The uploaded image is unacceptable.

**Figure 5-17** Identification result



# 6 Integrating with LTS to Analyze Logs in Real Time

## 6.1 Introduction

### Scenarios

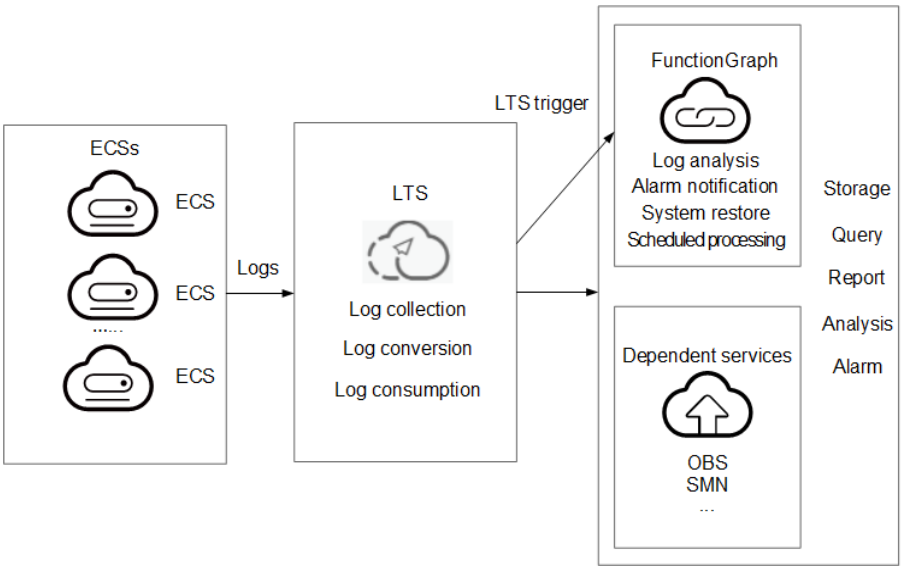
Quickly collect, process, and convert task logs of servers, such as ECSs, through Log Tank Service (LTS).

Obtain log data based on an LTS trigger created on FunctionGraph, analyze and process key information in the logs by using a customized function, and then filter alarm logs.

Use SMN to push alarm messages to service personnel by SMS message or email.

Store processed log data in a specified OBS bucket for subsequent processing. The processing workflow is shown in **Figure 6-1**.

**Figure 6-1** Processing workflow



### Values

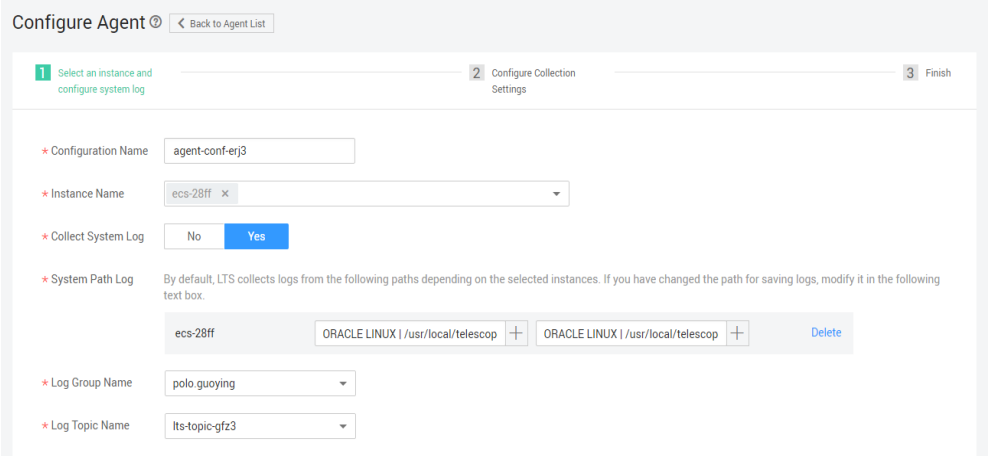
- Quickly collects and converts logs through LTS.
- Processes and analyzes data in response to log events in a serverless architecture, which features automatic scaling, no operation and maintenance, and pay-per-use billing.
- Sends alarm notifications through SMN.

## 6.2 Preparation

### Collecting and Storing Logs

- Create a log group, for example, **polo.guoying** on the LTS console. For details, see **Creating a Log Group**.
- Create a log stream, for example, **lts-topic-gfz3** on the LTS console. For details, see **Creating a Log Stream**.
- Configure an agent to collect logs from servers, such as ECSs, to a specified log group, as shown in **Figure 6-2**. For details, see **Installing the ICAgent**.

**Figure 6-2** Configuring an agent




### Pushing Alarm Messages

- Create a topic named **fss_test** on the SMN console. For details, see **Creating a Topic**.
- Add subscriptions to the **fss_test** topic to push alarm messages to specified email endpoints. For details, see **Adding Subscriptions**.
- Define an environment variable named **SMN_Topic** with value **fss_test** to push alarm messages to the subscription endpoints under the **fss_test** topic.

  NOTE

  Alarm messages of a subscribed topic can be pushed through email, SMS messages, and HTTP/HTTPS.

  In this example, when log events trigger the specified function through an LTS trigger, the function filters alarm logs and pushes alarm message to the subscription endpoints.

### Processing Cloud Data

Create an OBS bucket and object, and configure event notifications.

1. Create a bucket and an object on the OBS console, as shown in **Figure 6-3**. For details, see **Creating a Bucket**.

   **Figure 6-3** Creating a bucket

   **NOTE**

   Name the bucket as **logstore** and the object as **log.txt** to store log data.

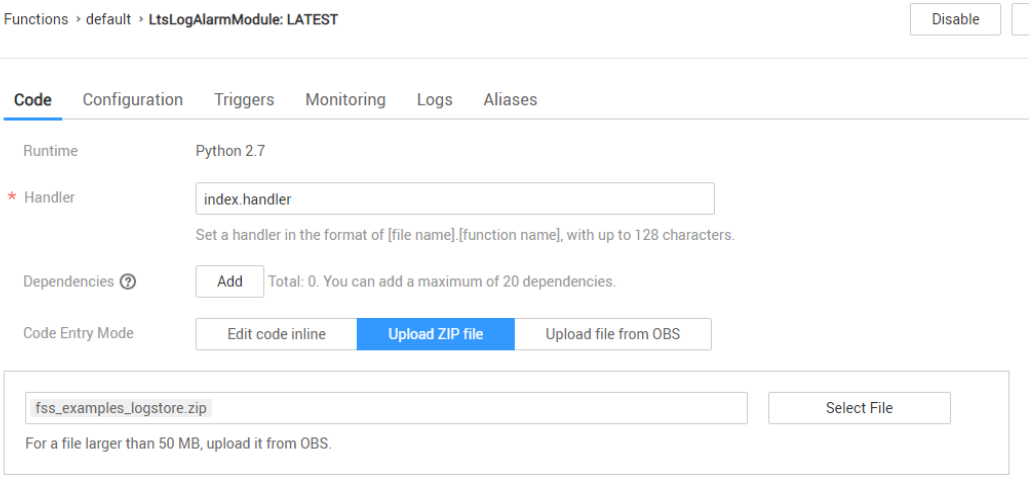
## 6.3 Building a Program

You can **download** and use the alarm log extracting program package provided in this example.

### Creating a Function

Create a function by uploading the **sample code package** to extract logs, as shown in **Figure 6-4**. For details, see **Creating a Function**.

**Figure 6-4** Creating a function




This function performs Base64 decoding on received log event data, extracts alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR**, and then stores the extracted logs in the specified OBS bucket. Set log extraction conditions based on the content of your service logs.

### Setting Environment Variables

On the **Configuration** tab page of the preceding function, set environment variables to pass the bucket address, bucket name, and object name, as shown in **Table 6-1**.

**Table 6-1** Environment variables

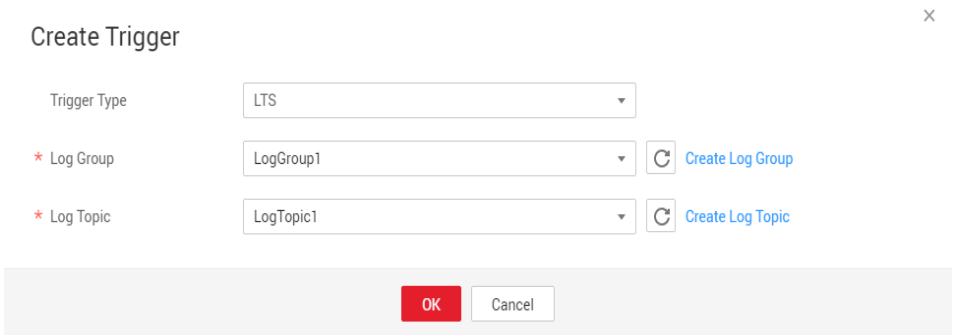
| Environment Variable | Description |
| --- | --- |
| obs_address | OBS endpoint. To obtain the OBS endpoint, see **Regions and Endpoints**. For example, the OBS endpoint in North China is **obs.cn-north-1.myhuaweicloud.com**. |
| obs_store_bucket | Name of the target bucket for storing logs. |
| obs_store_objName | Name of the target file for storing logs. |
| SMN_Topic | SMN topic. |

Set the environment variables by following the procedure in **Environment Variables**.

## 6.4 Adding an Event Source

Create an LTS trigger by using the log group and log topic created in **Preparation**, and configure the trigger information according to **Figure 6-5**.

**Figure 6-5** Creating an LTS trigger




When the accumulated log size or log retention period meets a specified threshold, LTS log data is consumed, which triggers the function associated with the log group.

# 6.5 Processing Log Data

Email notifications will be received from SMN if alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR** are generated, as shown in **Figure 6-6**. You can also view details of the alarm logs by opening the **log.txt** file in the specified bucket, as shown in **Figure 6-7**.

**Figure 6-6** Email notification

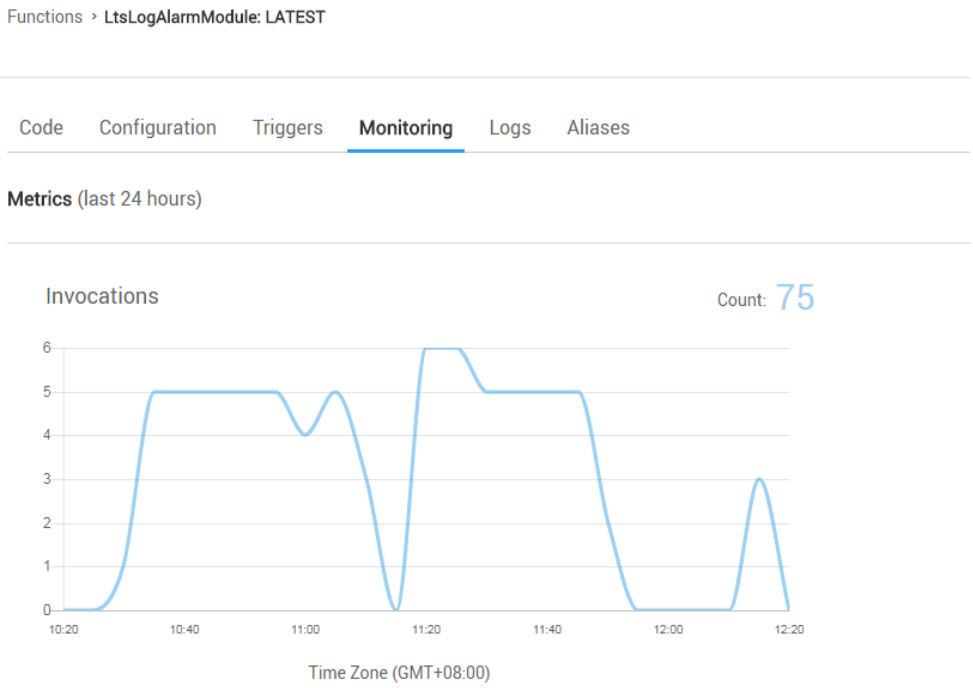
Get warning message.The content of message is:["{\"ip\": \"192.168.1.98\", \"line_no\": 616, \"host_name\": \"ecs-testagent.novalocal\", \"time\": 1530009653059, \"path\": \"/usr/local/telescope/log/common.log\", \"message\": \"2018-06-26/18:40:53 [WRN] [config.go:82] The projectId or instanceId of config.json is not consistent with metadata, use metadata.\\\\n\", \"log_uid\": \"663d6930-792d-11e8-8b09-286ed488ce70\"}"]

**Figure 6-7** Alarm log details

"\"{\\\"message\\\":\\\"2018-06-26/18:40:53 [WRN] [config.go:82] The projectId or instanceId of config.json is not consistent with metadata, use
metadata.\\\\n\\\",\\\"time\\\":1530009653059,\\\"host_name\\\":\\\"ecs-testagent.novalocal\\\",\\\"ip\\\":\\\"192.168.1.98\\\",\\\"path\\\":\\\"/usr/local/telescope/log/common.log\\\",\\\"log_uid\\\":\\\"663d6930-792d-11e8-8b09-286ed488ce70\\\",\\\"line_no\\\":616}\""

On the **Monitoring** tab page of the function, check the number of invocations, as shown in **Figure 6-8**.

**Figure 6-8** Function metrics




# 6.6 Other Application Scenarios

FunctionGraph and Log Tank Service (LTS) can be used to process cloud logs, push alarm messages, and store logs in a specified Object Storage Service (OBS) bucket. You can use FunctionGraph and LTS in multiple scenarios. For example, you can create a timer trigger to periodically analyze and process log data in an OBS bucket.

# 7 Integrating with CTS to Analyze Login/Logout Security

## 7.1 Introduction

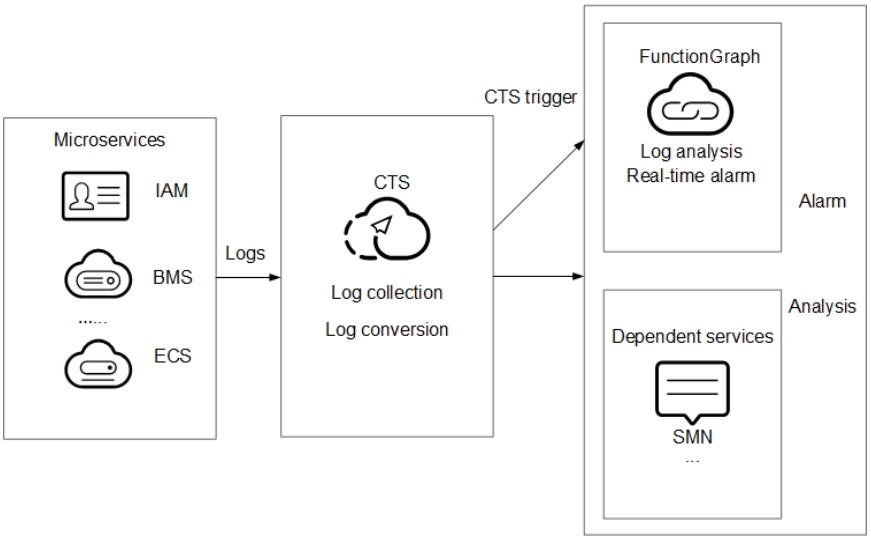
### Scenarios

Collect real-time records of operations on cloud resources.

Create a Cloud Trace Service (CTS) trigger to obtain records of subscribed cloud resource operations; analyze and process the operation records, and report alarms.

Use SMN to push alarm messages to service personnel by SMS message or email. The processing workflow is shown in **Figure 7-1**.

**Figure 7-1** Processing workflow




### Values

- Quickly analyzes operation records collected by CTS and filters out operations from specified IP addresses.

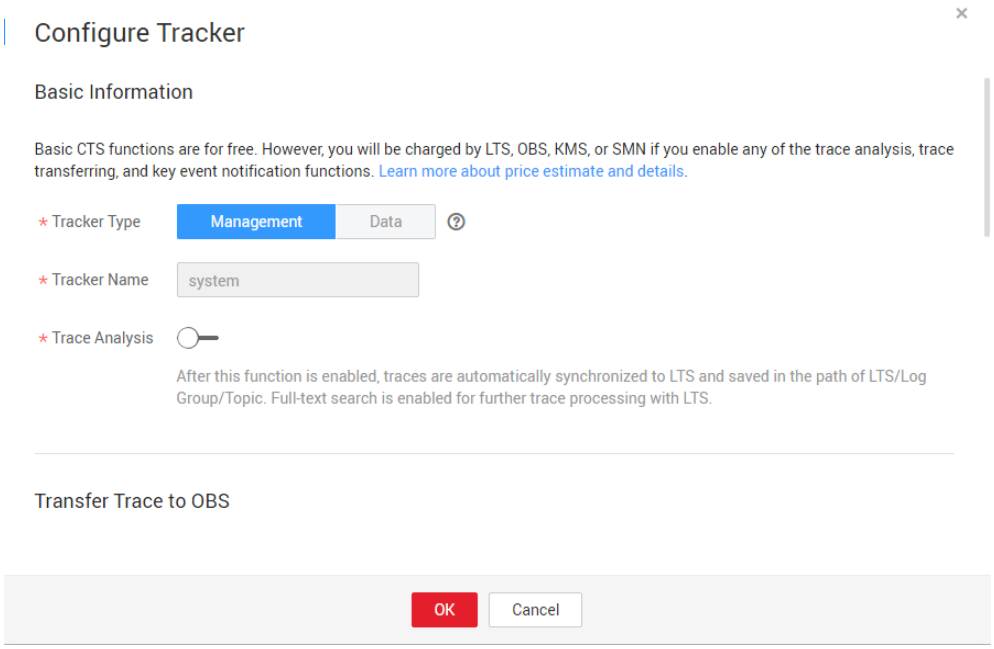
- Processes and analyzes data in response to log events in a serverless architecture, which features automatic scaling, no operation and maintenance, and pay-per-use billing.
- Sends alarm notifications through SMN.

# 7.2 Preparation

## Enabling CTS

Configure the tracker on CTS, as shown in **Figure 7-2**. For details, see **Configuring a Tracker**.

**Figure 7-2** Configuring the tracker




## Creating an Agency

**Step 1** Log in to the **IAM console**, and choose **Agencies** in the navigation pane.

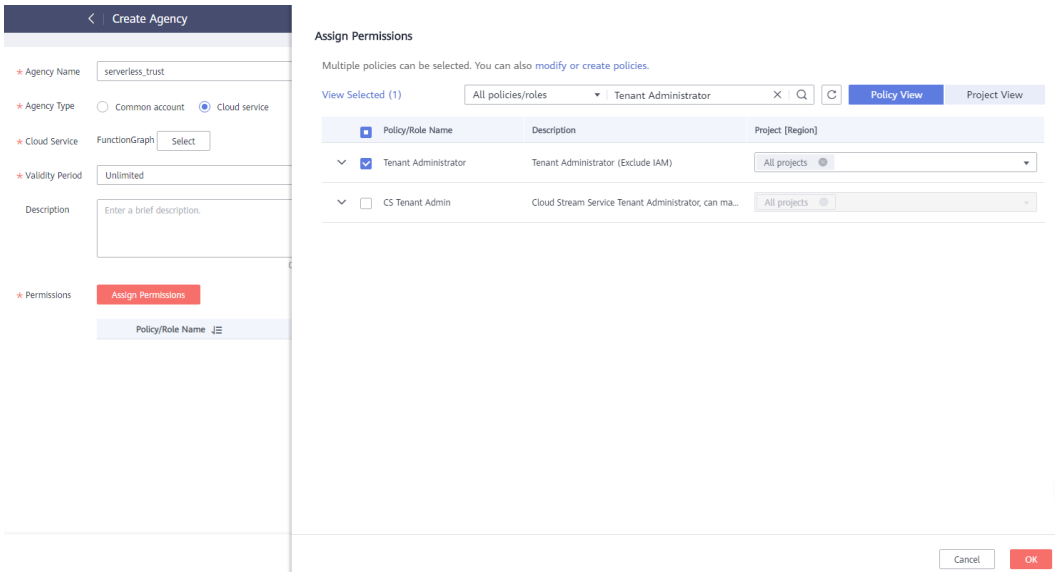
**Step 2** On the **Agencies** page, click **Create Agency**.

**Step 3** Set the agency information.

- For **Agency Name**, enter **serverless_trust**.
- For **Agency Type**, select **Cloud service**.
- For **Cloud service**, select **FunctionGraph**.
- For **Validity Period**, select **Unlimited**.
- Click **Assign Permissions**. On the **Assign Permissions** page, select **Tenant Administrator**.

**Figure 7-3** Creating an agency




**NOTE**

Users with the Tenant Administrator permission can perform any operations on all cloud resources of the enterprise.

**Step 4** Click **OK**.

**----End**

## Pushing Alarm Messages

- Create a topic named **cts_test** on the SMN console. For details, see **Creating a Topic**.
- Add subscriptions to the **cts_test** topic to push alarm messages to specified email endpoints. For details, see **Adding Subscriptions**.

  **NOTE**

  Alarm messages of a subscribed topic can be pushed through email, SMS messages, and HTTP/HTTPS.

  In this example, when operation log events trigger the specified function, the function filters operations not from an IP address whitelist and pushes alarm message to the subscription endpoints.

# 7.3 Building a Program

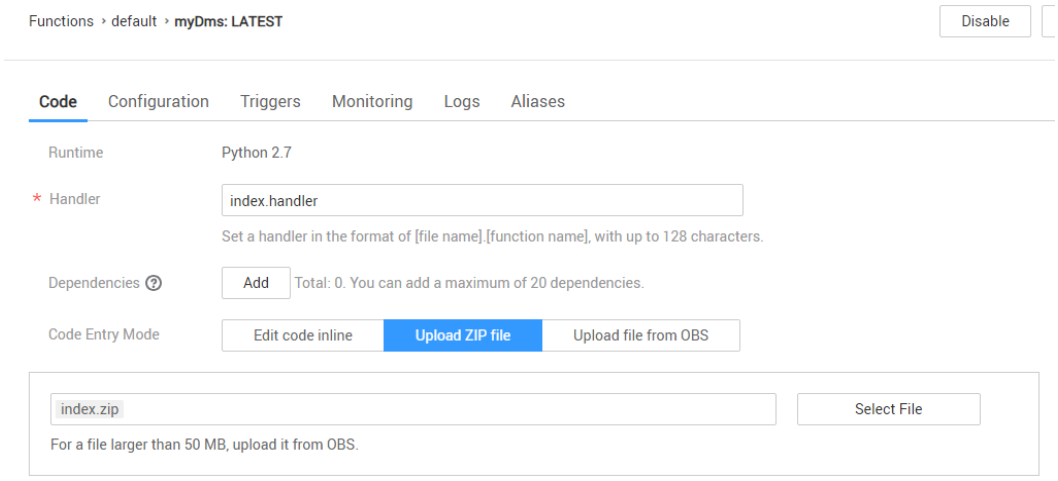
You can **download** and use the alarm log analysis program package provided in this example.

## Creating a Function

Create a function by uploading the **sample code package** to analyze logs, as shown in **Figure 7-4**. For details, see **Creating a Function**.

**Figure 7-4** Creating a function




This function analyzes received operation records, filters logins or logouts from unauthorized IP addresses using a whitelist, and sends alarms under a specified SMN topic. This function can be used to build an account security monitoring service.

## Setting Environment Variables

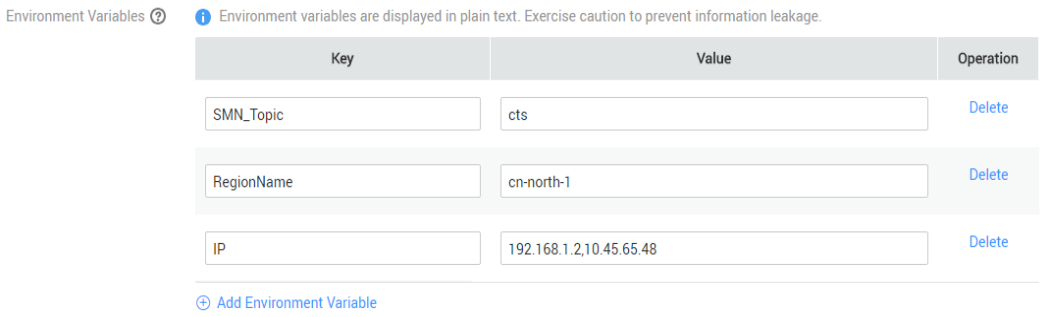
On the **Configuration** tab page of the function details page, set the environment variables listed in **Table 7-1**.

**Table 7-1** Environment variables

| Environment Variable | Description |
| --- | --- |
| SMN_Topic | SMN topic. |
| RegionName | Region name. |
| IP | IP address whitelist. |

Set the environment variables (**Figure 7-5**) by following the procedure in **Environment Variables**.

**Figure 7-5** Setting environment variables

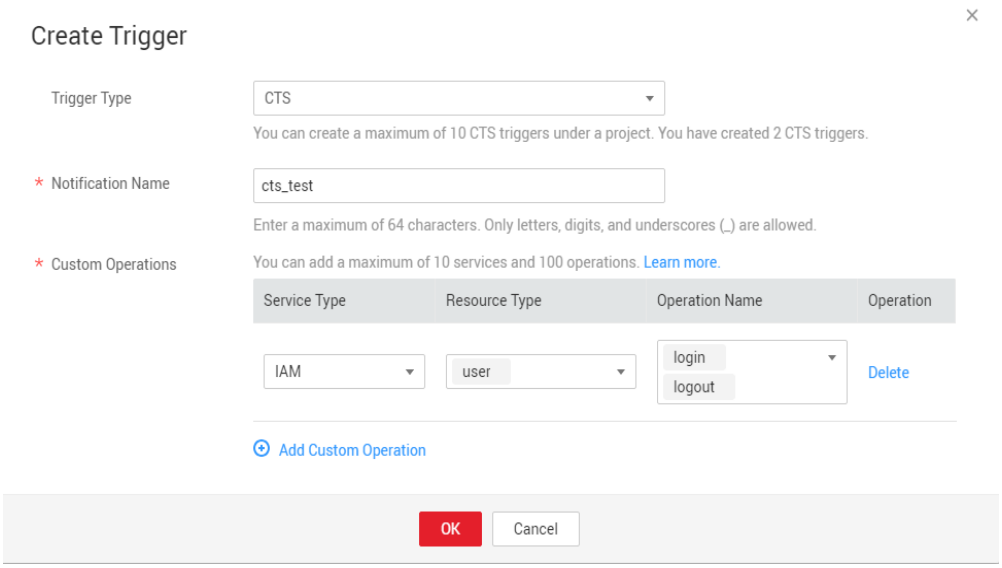


# 7.4 Adding an Event Source

Create a CTS trigger, as shown in **Figure 7-6**.

**Figure 7-6** Creating a CTS trigger




CTS records the logins and logouts of users on IAM.

# 7.5 Processing Operation Records

When a user performs login or logout using an account, the subscription service log will be triggered and a function will be directly invoked. The system then checks whether the IP address of the current login or logout account is in the whitelist based on function code. If the IP address is not in the whitelist, Simple Message Notification (SMN) will send notifications shown in **Figure 7-7**.

**Figure 7-7** Email notification

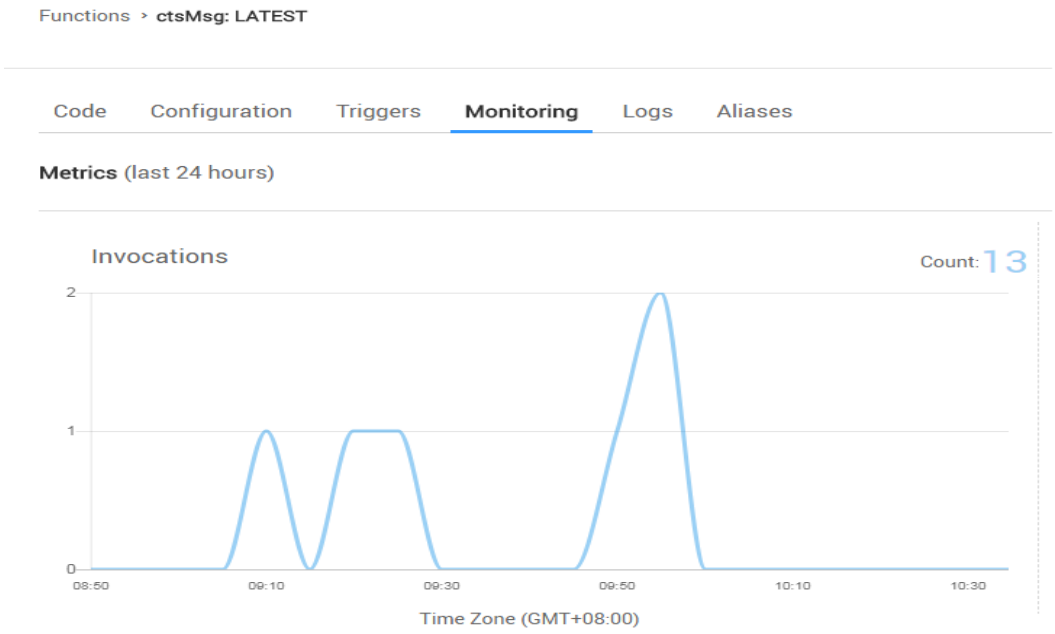
```
Illegal operation[  IP:10.65.56.139,  Action:login]
-------
```

The email contains the unauthorized IP address and user operation (login or logout).

On the **Monitoring** tab page of the function, check the number of invocations, as shown in **Figure 7-8**.

**Figure 7-8** Function metrics



# 8 Periodically Starting or Stopping HUAWEI CLOUD ECSs

## Introduction

If you need to start or stop your ECSs at specified time, you can use FunctionGraph to call the corresponding ECS APIs.

## Preparation

Obtain the program package for periodically **starting** or **stopping** ECSs.

## Building a Program

**Step 1** Create an agency named **EcsOperation** that delegates FunctionGraph to access other cloud services.

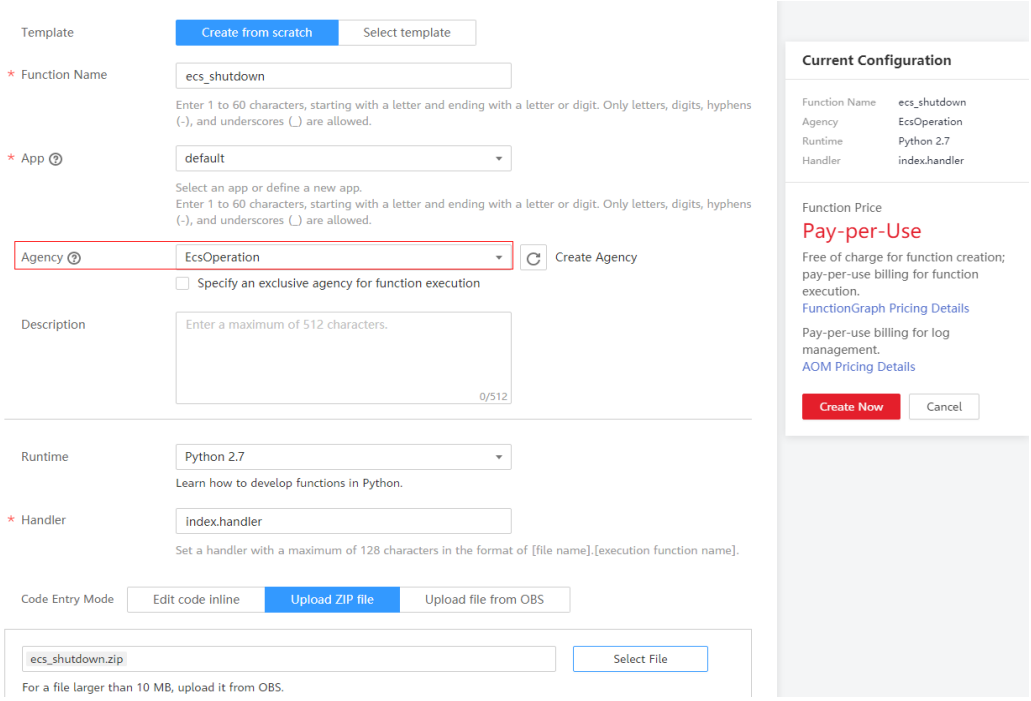
NOTE

An agency is required if FunctionGraph accesses other cloud services. For details on how to create an agency, see **Creating an Agency**.

**Step 2** Create a function.

Create a function for periodically starting or stopping ECSs. You need to upload the program package (for **starting** or **stopping** ECSs) as shown in **Figure 8-1** and select the agency created in **Step 1**. For details, see **Creating a Function**.

**Figure 8-1** Creating a function for periodically stopping ECSs




**Step 3** Set environment variables.

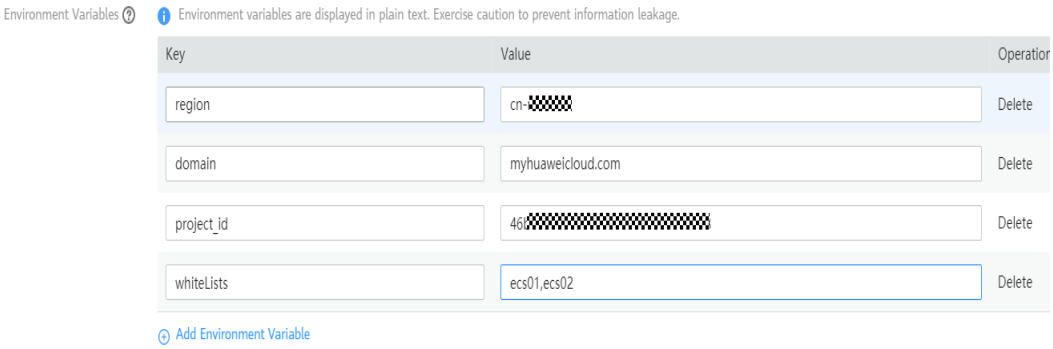
On the **Configuration** tab page, set environment variables according to **Table 8-1**.

**Table 8-1** Environment variables

| Environment Variable | Description |
| --- | --- |
| region | Region where your ECSs are located, for example, **cn-north-4**. |
| domain | Endpoint of the ECS service in the format of "{region}.{domain}", for example, **cn-north-4.myhuaweicloud.com**. To obtain the endpoint information, see **Regions and Endpoints**. |
| projectId | ID of the project to which the ECSs belong. |
| whiteLists | • If you want to periodically start certain ECSs, specify the names of the ECSs that do not need to be started and separate them with commas (,).<br>• If you want to periodically stop certain ECSs, specify the names of the ECSs that do not need to be stopped and separate them with commas (,). |

Set the environment variables (**Figure 8-2**) by following the procedure in **Environment Variables**.

**Figure 8-2** Setting environment variables



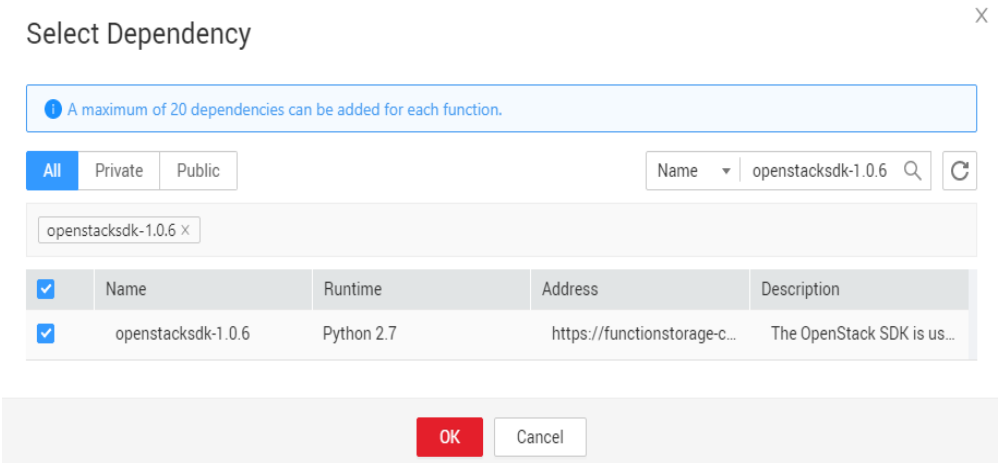

**NOTE**

- This practice does not have limitations on the region for function execution. For example, if your function runs in **CN North-Beijing1** and you want to start or stop ECSs deployed in **CN North-Beijing4**, change the **projectId**, **region**, and **domain** parameters to those of the **CN North-Beijing4** region.
- If a large number of ECSs need to be started or stopped, increase the execution timeout for your function.
- In **Table 8-1**, all environment variables except **domain** and **whiteLists** are mandatory. If you do not specify a domain, the default value **myhuaweicloud.com** of the **domain** parameter in the program package will be used. For **whiteLists**, specify the names of the ECSs to be started or stopped and separate them with commas (,).

**Step 4** Add a dependency.

On the **Code** tab page, select the **openstacksdk-1.0.6** dependency, as shown in **Figure 8-3**. To download the **openstacksdk-1.0.6** dependency, visit **HUAWEI CLOUD SDKs**.

**Figure 8-3** Selecting a dependency



For more information, see **Configuring Dependencies for a Function**.

**----End**

## Adding an Event Source

Create a timer trigger and set the trigger parameters according to **Figure 8-4**.

**Figure 8-4** Creating a timer trigger