FunctionGraph

Best Practices

Issue	01
Date	2024-03-26

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property 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 Cloud 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

- The function and the source and destination buckets for storing images must be in the same region.
- 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.

- For **Region**, select a region.
- For **Bucket Name**, enter **your-bucket-input**.
- For **Data Redundancy Policy**, select **Single-AZ storage**.
- For **Default Storage Class**, select **Standard**.
- For **Bucket Policies**, select **Private**.
- **Server-Side Encryption**: Select **Disable**.
- For **Direct Reading**, select **Disable**.

Retain the default values for other parameters and click **Create Now**.

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

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

Step 4 View **your-bucket-input** and **your-bucket-output** in the bucket list.

----End

Creating an Agency

Step 1 In the left navigation pane of the management console, choose **Management & Governance** > **Identity and Access Management** to go to the IAM console. Then 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**.
- Enter a description.

Step 4 Click **Next**. On the **Select Policy/Role** page, select **Tenant Administrator** and click **Next**.

NOTE

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

Step 5 Select an authorization scope that meets your service requirements, and click **OK**.

----End

1.3 Building a Program

Download fss_examples_image_thumbnail.zip (**SHA-256 verification package**) to create an image compressing function from scratch.

Creating a Deployment Package

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

Figure 1-1 Sample code directory

index.py

```
# -*-coding:utf-8 -*-
import ...

LOCAL_MOUNT_PATH = '/tmp/'


def handler(event, context):
    ak = context.getAccessKey()
    sk = context.getSecretKey()
    if ak == "" or sk == "":
        context.getLogger().error('Failed to access OBS because no temporary '
                                  'AK, SK, or token has been obtained. Please '
                                  'set an agency.')
        return 'Failed to access OBS because no temporary AK, SK, or token ' \
               'has been obtained. Please set an agency. '

    obs_endpoint = context.getUserData('obs_endpoint')
    if not obs_endpoint:
        return 'obs_endpoint is not configured'

    output_bucket = context.getUserData('output_bucket')
    if not output_bucket:
        return 'output_bucket is not configured'

    compress_handler = ThumbnailHandler(context)
    records = event.get("Records", None)
    return compress_handler.run(records[0])
```

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

```
def handler(event, context):
    ak = context.getAccessKey()
    sk = context.getSecretKey()
    if ak == "" or sk == "":
        context.getLogger().error('Failed to access OBS because no temporary '
                                  'AK, SK, or token has been obtained. Please '
                                  'set an agency.')
        return 'Failed to access OBS because no temporary AK, SK, or token ' \
               'has been obtained. Please set an agency. '

    obs_endpoint = context.getUserData('obs_endpoint')
    if not obs_endpoint:
        return 'obs_endpoint is not configured'

    output_bucket = context.getUserData('output_bucket')
    if not output_bucket:
        return 'output_bucket is not configured'

    compress_handler = ThumbnailHandler(context)
    records = event.get("Records", None)
    return compress_handler.run(records[0])
```

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.

After setting the basic information, click **Create Function**.

- For **Function Name**, enter **fss_examples_image_thumbnail**.
- For **Agency**, select **serverless_trust** created in **Creating an Agency**.
- For **Runtime**, select **Python3.6**

Step 4 On the **fss_examples_image_thumbnail** details page, configure the following information:

1. On the **Code** tab, choose **Upload** > **Local ZIP**, upload the sample code **fss_examples_image_thumbnail.zip**, and click **OK**.
2. Choose **Configuration** > **Basic Settings**, set the following parameters, and click **Save**.
   - For **Memory**, select **256**.
   - For **Execution Timeout**, enter **40**.
   - For **Handler**, retain the default value **index.handler**.
   - For **App**, retain the default value **default**.
   - For **Description**, enter **Image compressing**.
3. Choose **Configuration** > **Environment Variables**, set environment variables, and click **Save**.

   output_bucket: the output bucket parameter defined in **index.py**. Set the value to **your-bucket-output**, the bucket created in **Creating OBS Buckets**.

obs_endpoint: the bucket address parameter defined in **index.py**. Set the value to **obs.region.myhuaweicloud.com**.

Table 1-1 Environment variable description

Environment Variable	Description
obs_endpoint	OBS endpoint. To obtain the OBS endpoint, see **Regions and Endpoints**.
output_bucket	OBS bucket for storing output images.

----End

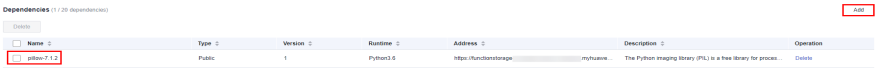
Selecting a Dependency

The sample code depends on the Pillow package, which should be imported as dependencies. The procedure is as follows:

Step 1 Go to the **fss_examples_image_thumbnail** details page, click the **Code** tab, and click **Add** in the **Dependencies** area at the bottom.

Step 2 Add public dependency **pillow-7.1.2**.

Figure 1-2 Adding dependencies



----End

NOTE

You do not need to configure the reference after adding a dependency, as it is preconfigured in the function code of the demo package.

1.4 Adding an Event Source

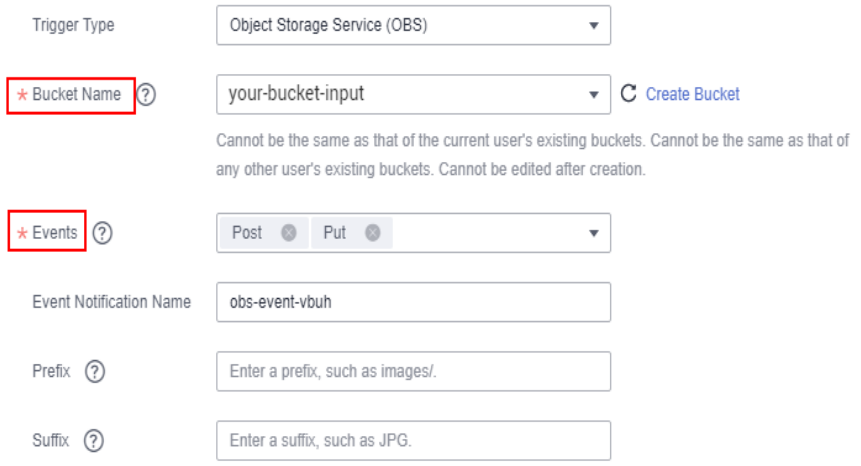
After creating the OBS buckets and function, you can add an event source to the function by creating an OBS trigger. Perform the following procedure:

Step 1 On the **fss_examples_image_thumbnail** page, choose **Configuration** > **Triggers**, and click **Create Trigger**.

Step 2 Select **Object Storage Service (OBS)** for **Trigger Type**, and set the trigger information, as shown in **Figure 1-3**.

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

Figure 1-3 Creating a trigger



Step 3 Click **OK**.

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

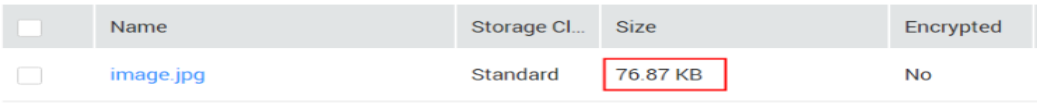
1.5 Compressing Images

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 1-4**.

Figure 1-4 Uploading an image



NOTE

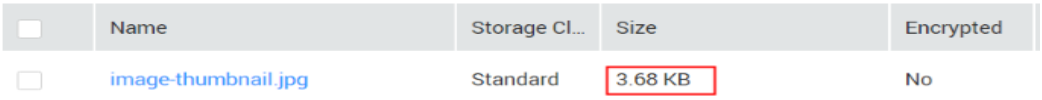
The size of the original **image.jpg** file exceeds 28 KB.

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.

Go to the **Objects** page of the **your-bucket-output** bucket and view the size of the compressed image.

Figure 1-5 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

- The function and the source and destination buckets for storing images must be in the same region.
- 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.

- For **Region**, select a region.
- For **Data Redundancy Policy**, select **Single-AZ storage**.
- For **Bucket Name**, enter **hugb-bucket-input**.
- For **Default Storage Class**, select **Standard**.
- For **Bucket Policies**, select **Private**.
- For **Server-Side Encryption**: select **Disable**
- For **Direct Reading**, select **Disable**.

Click **Create Now**.

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

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

Step 4 View **hugb-bucket-input** and **hugb-bucket-output** in the bucket list.

----End

Creating an Agency

Step 1 In the left navigation pane of the management console, choose **Management & Governance** > **Identity and Access Management** to go to the IAM console. Then 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**.
- Enter a description.

Step 4 Click **Next**. On the **Select Policy/Role** page, select **Tenant Administrator** and click **Next**.

NOTE

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

Step 5 Select an authorization scope that meets your service requirements, and click **OK**.

----End

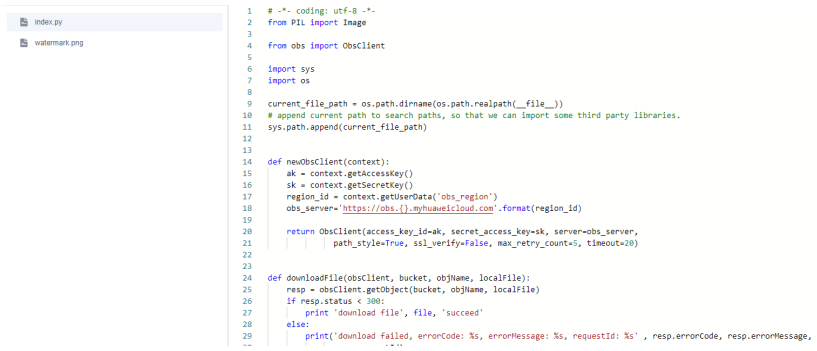
2.3 Building a Program

Download watermark.zip to create an image watermarking function from scratch.

Creating a Deployment Package

This example uses a Python function to watermark images. For details about function development, see **Developing Functions in Python**. **Figure 2-1** shows the sample code directory. The service code is not described.

Figure 2-1 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)
    # Upload 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**.

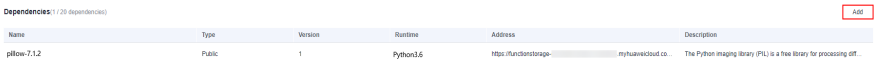
Step 3 Set the function information.

After setting the basic information, click **Create**.

- For **Function Name**, enter **fss_examples_image_watermark**.
- For **Agency**, select **serverless_trust** created in **Creating an Agency**.
- For **Runtime**, select **Python 3.6**.

Step 4 Go to the **fss_examples_image_watermark** details page, click the **Code** tab, click **Add** in the **Dependencies** area at the bottom, and add the public dependency **pillow-7.1.2**.

Figure 2-2 Adding a dependency



Step 5 On the **fss_examples_image_watermark** details page, configure the following information:

1. On the **Code** tab, choose **Upload** > **Local ZIP**, upload the sample code **watermark.zip**, and click **OK**.
2. Choose **Configuration** > **Basic Settings**, set the following parameters, and click **Save**.
   - For **Memory**, select **128**.
   - For **Execution Timeout**, enter **3**.
   - For **Handler**, retain the default value **index.handler**.
   - For **App**, retain the default value **default**.
   - For **Description**, enter **Image watermarking**.
3. Choose **Configuration** > **Environment Variables**, set environment variables, and click **Save**. The following figure is for reference only. Replace the following values with the actual values.

   obs_output_bucket: the output bucket parameter defined in **index.py**. Set the value to **hugb-bucket-output**, the bucket created in **Creating OBS Buckets** for storing watermarked images.

   obs_region: the region where the bucket **obs_output_bucket** is located.

Table 2-1 Environment variable description

Environment Variable	Description
obs_region	Region to which the OBS bucket belongs. The value must be the same as the region to which the function belongs.
obs_output_bucket	OBS bucket for storing watermarked images.

----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:

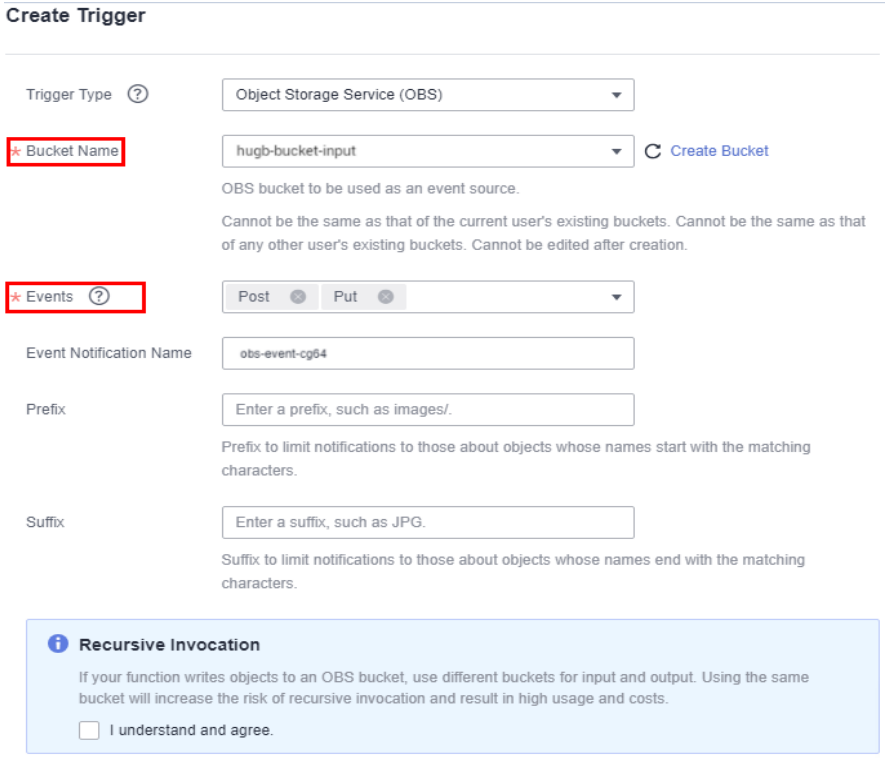
Step 1 On the **fss_examples_image_watermark** page, click the **Triggers** tab and click **Create Trigger**.

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

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

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

Figure 2-3 Creating an OBS trigger



Step 3 Click **OK**.

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

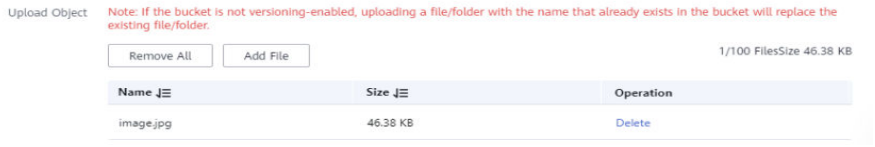
2.5 Watermarking Images

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 2-4**.

Figure 2-4 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 **fss_examples_image_watermark** on the **Logs** tab page.

The **Objects** page of the bucket **hugb-bucket-output** displays the watermarked image **image.jpg**, as shown in **Figure 2-5**. In the **Operation** column, click **Download** to download the image and view the watermarking effect, as shown in **Figure 2-6**.

Figure 2-5 Output image

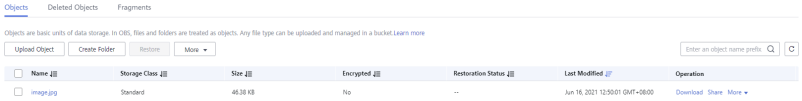


Figure 2-6 Watermarked image

3 Processing DIS Data

3.1 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.

3.2 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.

For **Default Subnet**, use the default settings.

Step 3 Confirm the configuration information and click **Create Now**.

----End

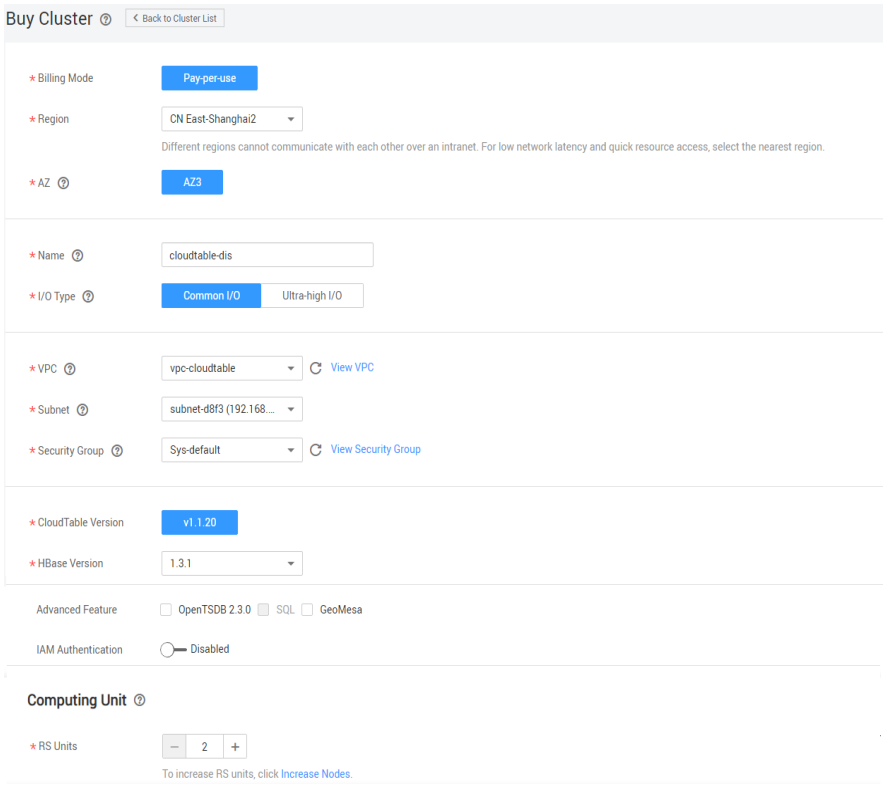
Creating a Cluster

Step 1 In the left navigation pane of the management console, choose **Analytics** > **CloudTable Service** to go to the CloudTable console. On the **Cluster Mode** page, click **Buy Cluster**.

Step 2 Set the cluster information.

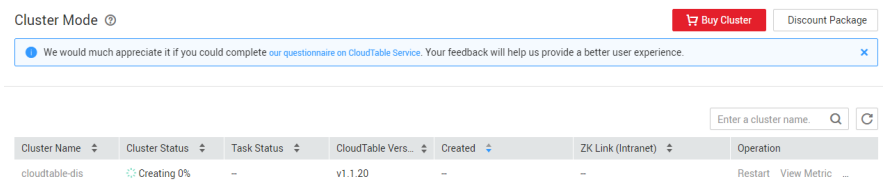
- **Region**: Use the default region.
- **Name**: Enter "cloudtable-dis".
- **VPC**: Select **vpc-cloudtable** created in **Creating a VPC**.
- Retain the default values for other parameters.

Figure 3-1 Buying a cluster



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

Figure 3-2 Creating a cluster



NOTE

Creating a cluster takes a long time. You can check the creation progress according to **Figure 3-2**.

----End

Creating an Agency

Step 1 In the left navigation pane of the management console, choose **Management & Governance** > **Identity and Access Management** to go to the IAM console. Then choose **Agencies** in the navigation pane.

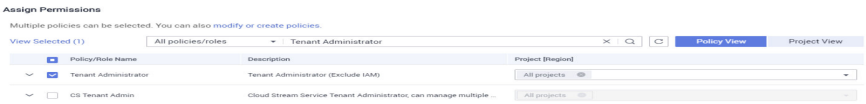
Step 2 On the **Agencies** page, click **Create Agency**.

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**.

Step 4 Click **Next**. On the **Select Policy/Role** page, select **Tenant Administrator** and click **Next**.

Figure 3-3 Creating an agency



NOTE

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

Step 5 Click **OK**.

----End

3.3 Building a Program

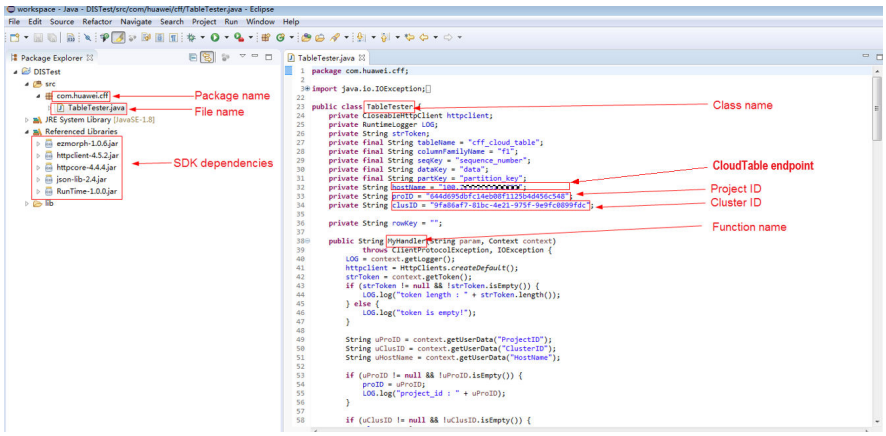
Download the **source code** and **program package** (including function dependencies) to create a function from scratch for converting DIS stream data formats.

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 3-4**.

Figure 3-4 Sample code



In the sample code, modify **proID** (project ID), **clusID** (cluster ID), and **hostName** (CloudTable endpoint), and save the modification.

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 3-5**.
2. Obtain the project ID in the project list, as shown in **Figure 3-6**.

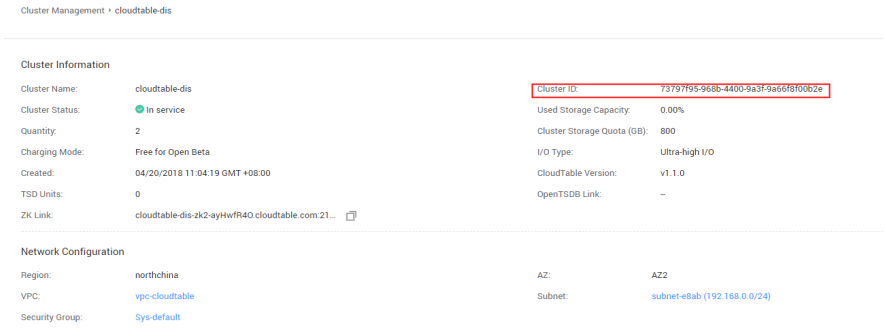
Figure 3-5 My Credentials

Figure 3-6 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 3-7**.

Figure 3-7 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 3-8 Exporting the code

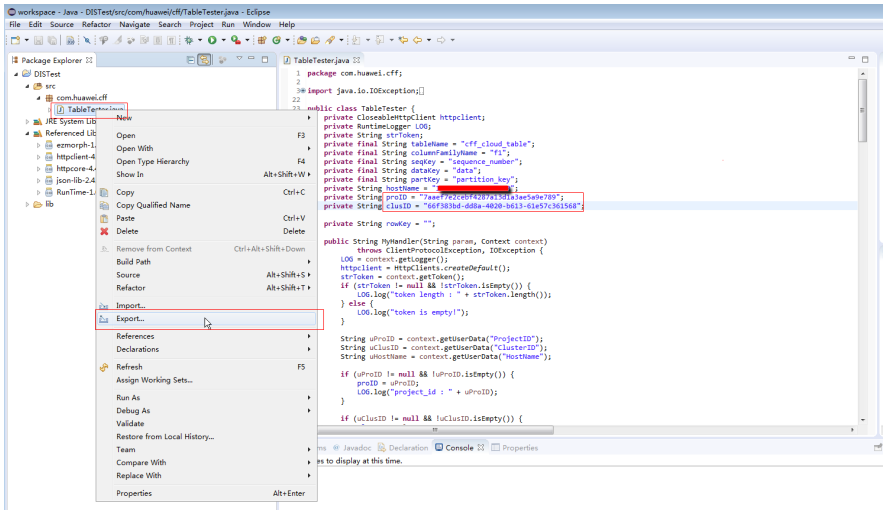


Figure 3-9 Selecting a file type

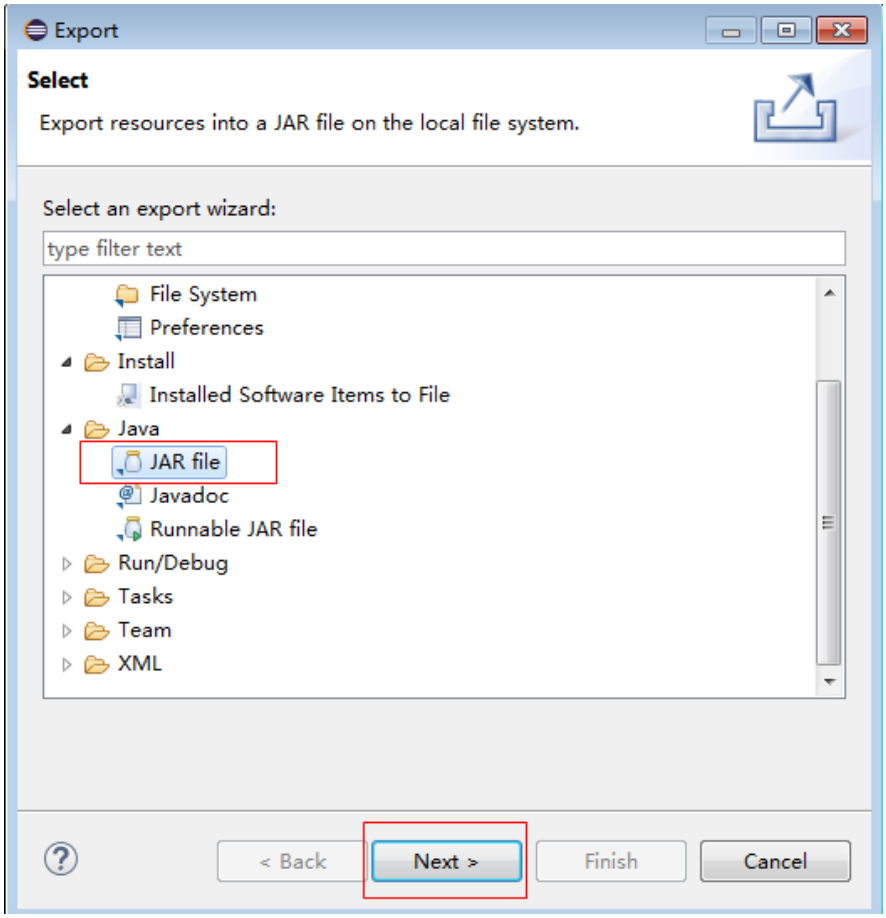
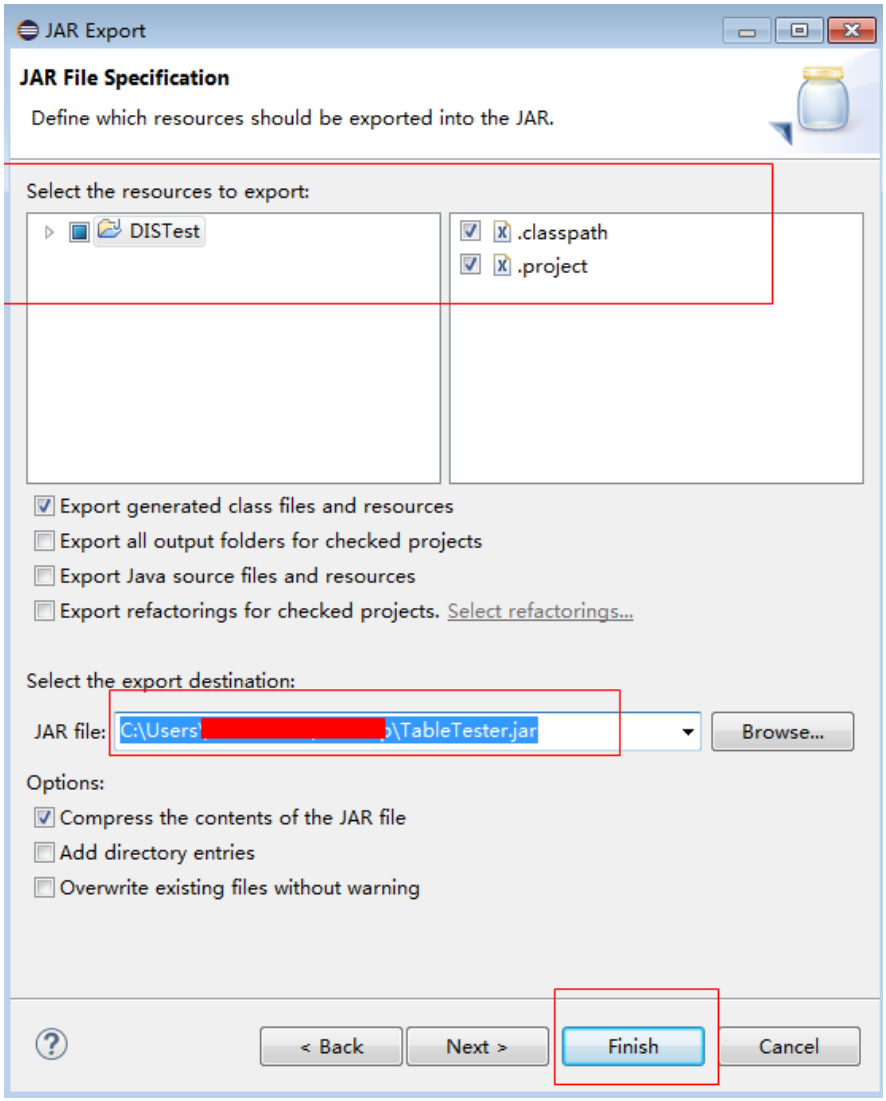


Figure 3-10 Publishing the code file



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 3-11**.
2. Use **Table Tester.jar** to replace **DIS Test.jar**, as shown in **Figure 3-12**.
3. Package all of the files into **disdemo.zip**, as shown in **Figure 3-13**.

Figure 3-11 File directory before replacement

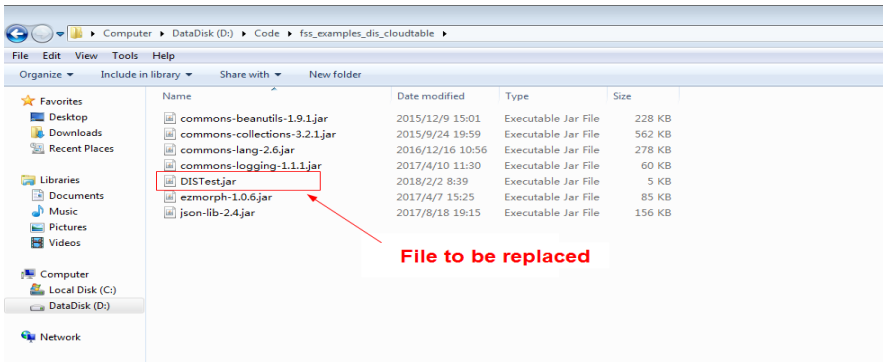


Figure 3-12 File directory after replacement

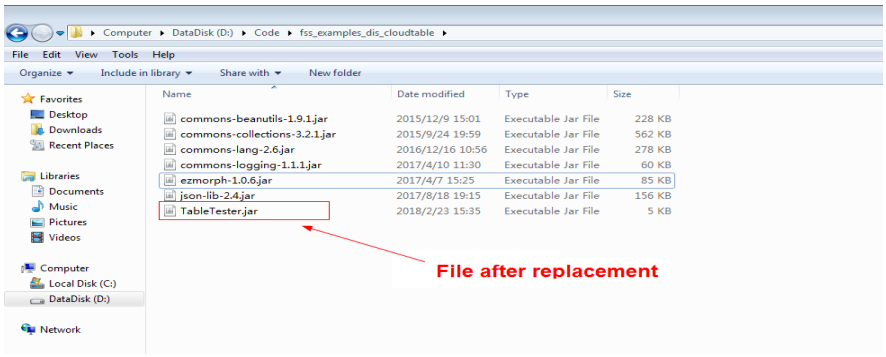
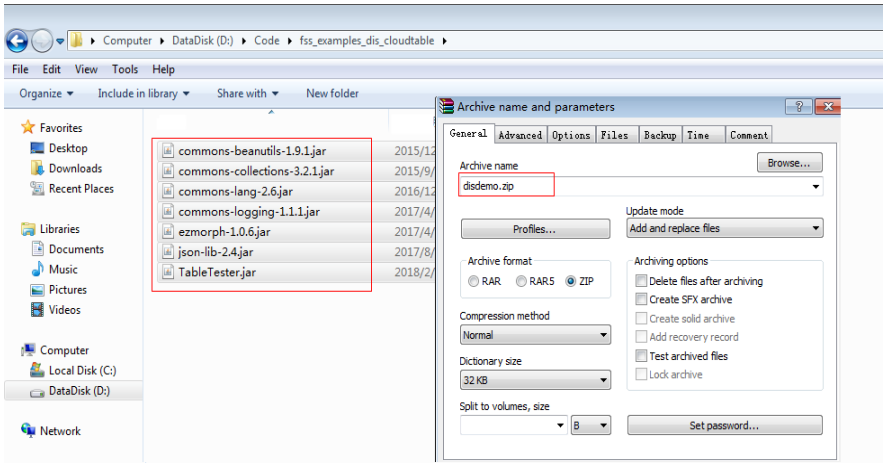


Figure 3-13 Packaging the files in ZIP format



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 basic information, and click **Create Function**.

- For **Function Name**, enter **DISDemo**.
- For **Agency**, select **DISDemo** created in **Preparation**.
- For **Runtime**, select **Java 8**.

Step 4 On the function details page, configure the following information:

- Choose **Configuration** > **Basic Settings**, change the handler to **com.huawei.cff.TableTester.MyHandler**, and click **Save**.
- On the **Code** tab, choose **Upload** > **Local ZIP**, upload the **disdemo.zip** package generated in **Packaging the Code**, and click **OK**.

----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** details page, choose **Configuration** > **Basic Settings**, and modify the following information as required:

- For **Memory**, select **512**.
- For **Execution Timeout**, enter **15**.
- Keep other parameters unchanged.

Step 2 Click **Save**.

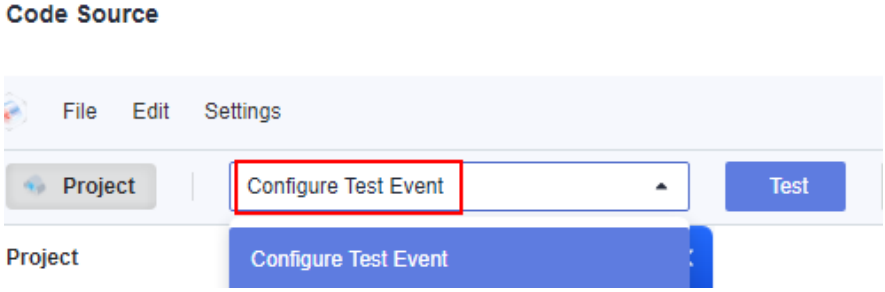
----End

3.4 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** on the **Code** tab, as shown in **Figure 3-14**.

Figure 3-14 Configuring a test event

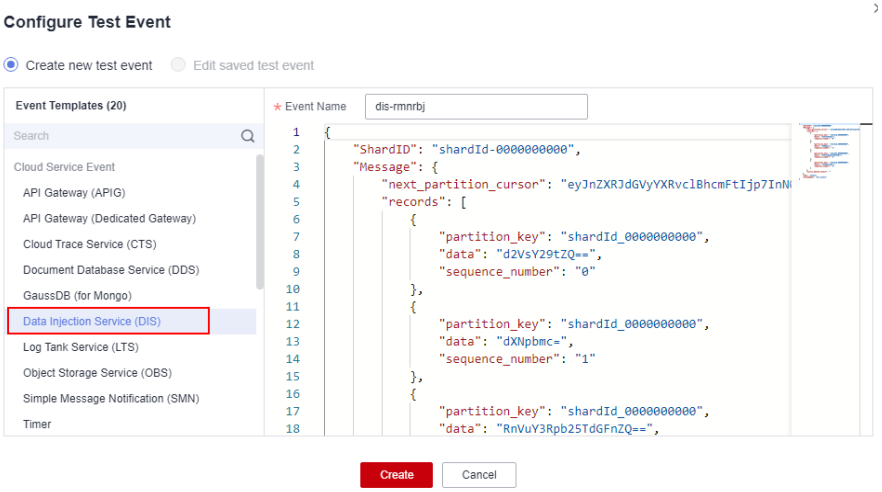


Step 2 In the **Configure Test Event** dialog box, set the test event information, as shown in **Figure 3-15**.

- Select **Create new test event**.

- Event Template: Select **Data Ingestion Service (DIS)**.
- For **Event Name**, enter **dis-test**.

Figure 3-15 Test event



Step 3 Click **Create**.

----End

3.5 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 3-16**.

Figure 3-16 Selecting a test event

Step 2 After the function is executed successfully, check the logs shown in **Figure 3-17**. For all logs of the function, go to the **Logs** tab page.

Figure 3-17 Function execution result

----End

4 Integrating with LTS to Analyze Logs in Real Time

4.1 Introduction

Scenarios

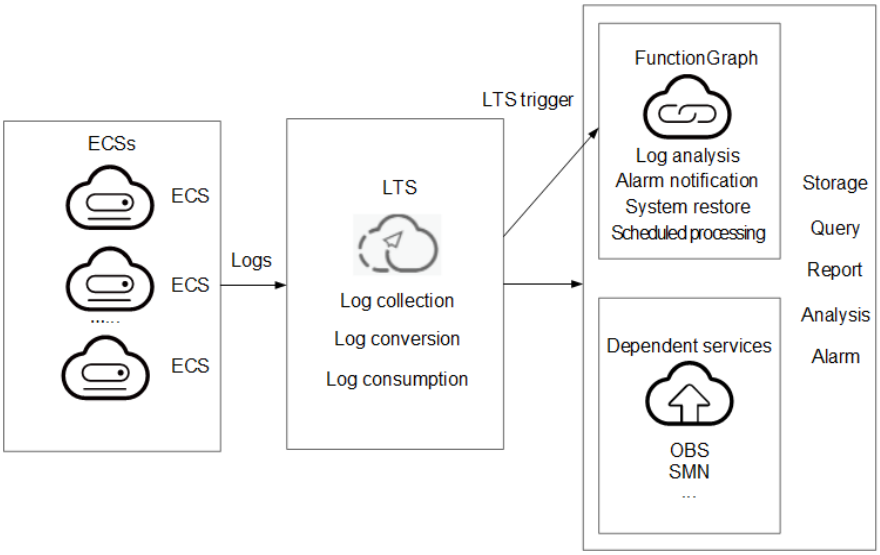
Quickly collect, process, and convert task logs of servers, such as ECSs, through Log Tank Service (LTS).

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 4-1**.

Figure 4-1 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.

4.2 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. For details, see **Installing the ICAgent**.

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. For details, see **Adding a Subscription**.
- 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 4-2**. For details, see **Creating a Bucket**.

   Figure 4-2 Creating a bucket

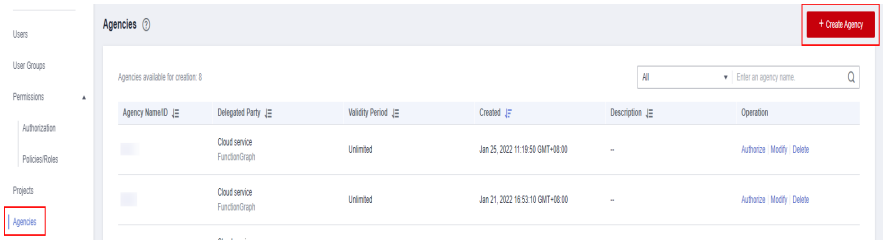
NOTE

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

Creating an Agency

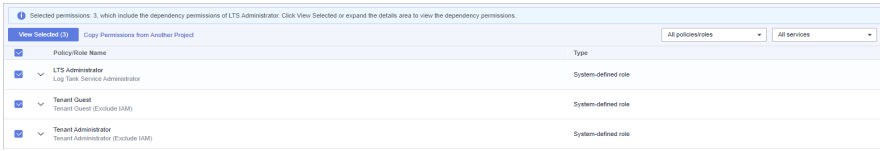
1. Log in to the Identity and Access Management (IAM) console.
2. On the IAM console, choose **Agencies** from the navigation pane, and click **Create Agency** in the upper right corner.

Figure 4-3 Creating an agency



3. Configure the agency.
   - For **Agency Name**, enter **LtsOperation**.
   - For **Agency Type**, select **Cloud service**.
   - For **Cloud Service**, select **FunctionGraph**.
   - For **Validity Period**, select **Unlimited**.
   - **Description**: Enter the description.
4. Click **Next**. On the displayed page, search for **LTS Administrator** and **Tenant Administrator** in the search box on the right and select them.

Figure 4-4 Selecting permissions



NOTE

LTS Administrator depends on **Tenant Guest**. When you select the former, the latter will also be selected.

5. Click **Next** and select the application scope of the permissions based on service requirements.

4.3 Building a Program

Download fss_examples_logstore_warning.zip to create an alarm log extraction function from scratch.

Creating a Function

Create a function by uploading the **sample code package** to extract logs. Select the Python 2.7 runtime and the agency **LtsOperation** created in **Creating an**

Agency. For details about how to create a function, see **Creating an Event 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 4-1**.

Table 4-1 Environment variables

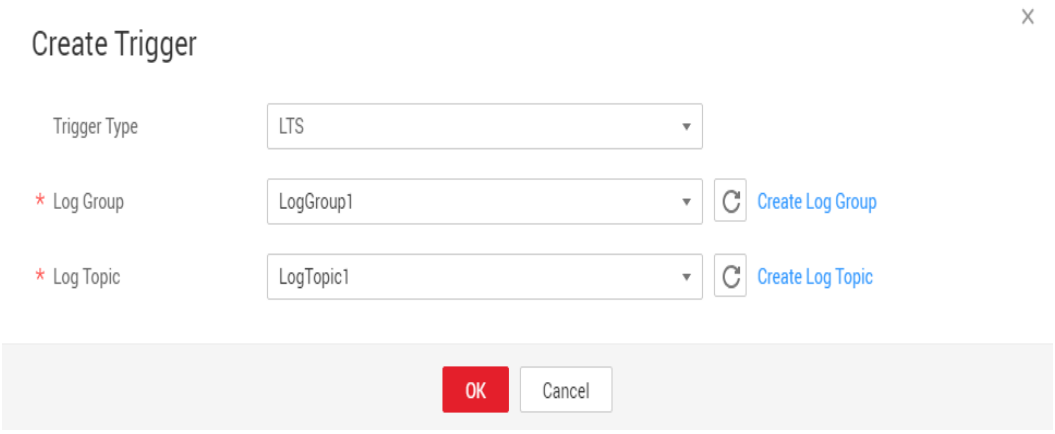
Environment Variable	Description
obs_address	OBS endpoint. To obtain the OBS endpoint, see **Regions and Endpoints**.
obs_store_bucket	Name of the destination bucket for storing logs.
obs_store_objName	Name of the target file for storing logs.
SMN_Topic	SMN topic.
region	Name of your region. To obtain the region name, see **Regions and Endpoints**.

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

4.4 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 4-5**.

Figure 4-5 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.

4.5 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 4-6**. You can also view details of the alarm logs by opening the **log.txt** file in the specified bucket, as shown in **Figure 4-7**.

Figure 4-6 Email notification

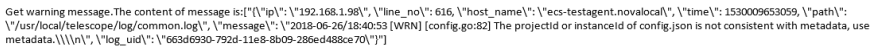
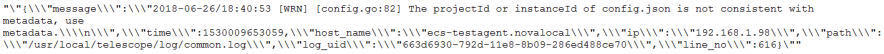
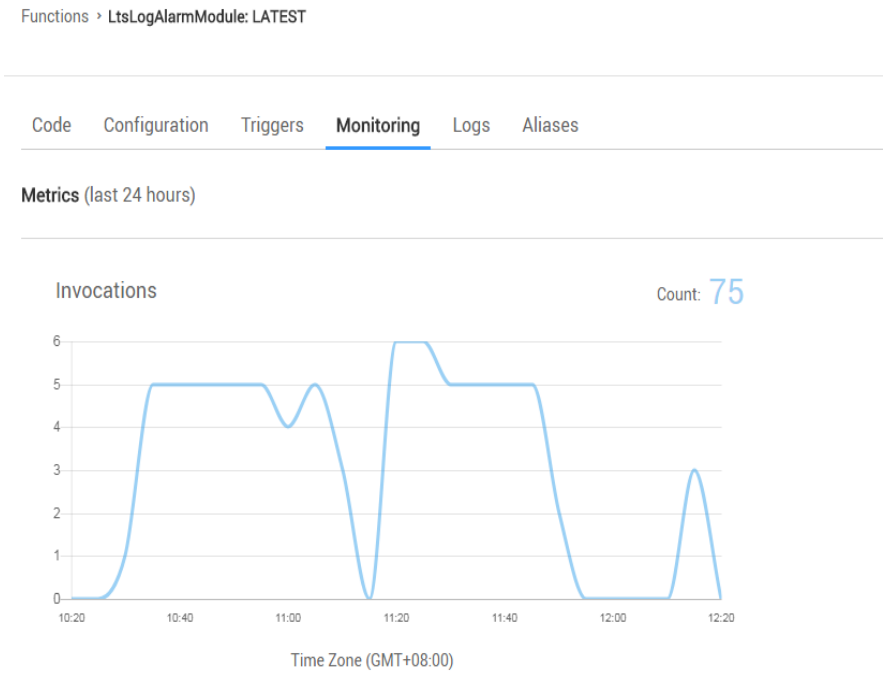


Figure 4-7 Alarm log details



On the **Monitoring** tab page of the function, check the number of invocations, as shown in **Figure 4-8**.

Figure 4-8 Function metrics



4.6 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.

5 Integrating with CTS to Analyze Login/Logout Security

5.1 Introduction

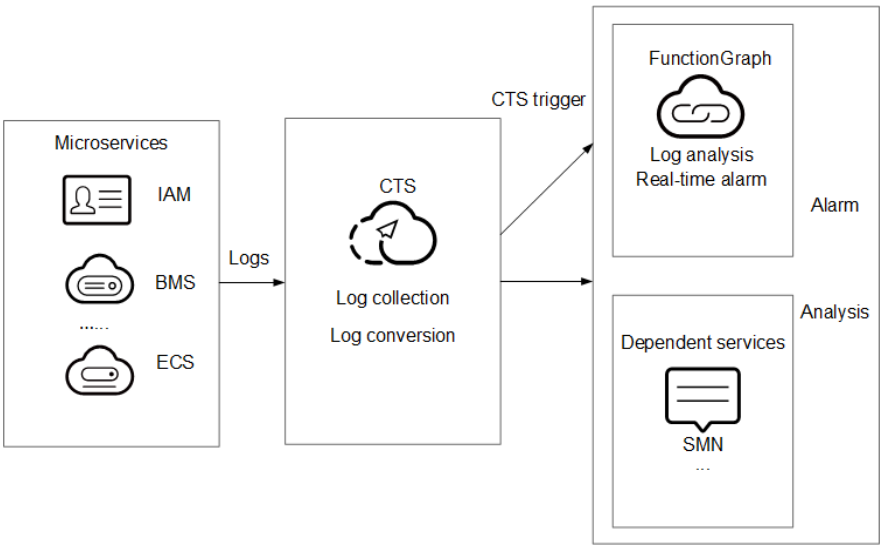
Scenarios

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 5-1**.

Figure 5-1 Processing workflow



Values

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

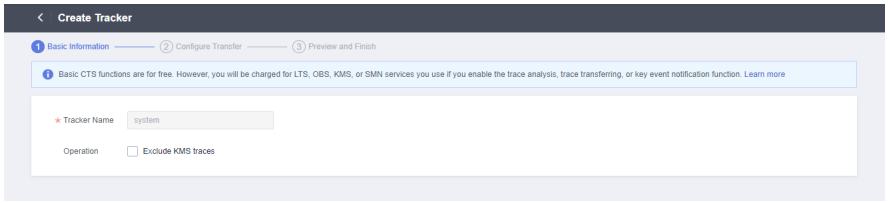
- Processes and analyzes data in response to log events in a serverless architecture, which features automatic scaling, no operation and maintenance, and pay-per-use billing.
- Sends alarm notifications through SMN.

5.2 Preparation

Enabling CTS

Configure a tracker on CTS, as shown in **Figure 5-2**. For details, see **Configuring a Tracker**.

Figure 5-2 Configuring a 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**.
- For **Description**, enter a description.

Step 4 Click **Next**. On the **Select Policy/Role** page, select **Tenant Administrator** and click **Next**.

NOTE

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

Step 5 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. For details, see **Adding a Subscription**.

NOTE

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

In this example, when operation log events trigger the specified function, the function filters operations that are performed by users not in the IP address whitelist, and pushes alarm messages to the subscription endpoints.

5.3 Building a Program

Download index.zip to create an alarm log analysis function from scratch.

Creating a Function

Create a function by uploading the **sample code package** to extract logs. Select the Python 2.7 runtime and the agency **serverless_trust** created in **Creating an Agency**. For details about how to create a function, see **Creating an Event 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 5-1**.

Table 5-1 Environment variables

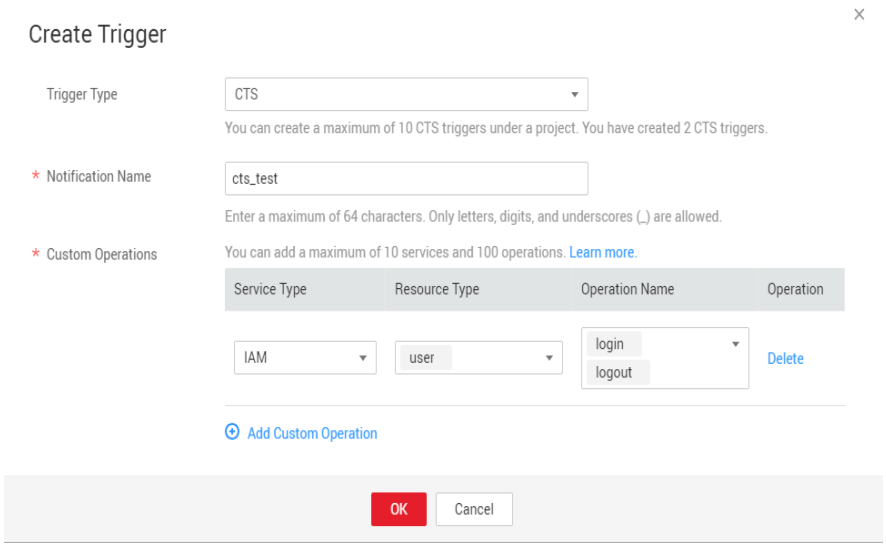
Environment Variable	Description
SMN_Topic	SMN topic.
RegionName	Region name.
IP	IP address whitelist.

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

5.4 Adding an Event Source

Create a CTS trigger, as shown in **Figure 5-3**.

Figure 5-3 Creating a CTS trigger

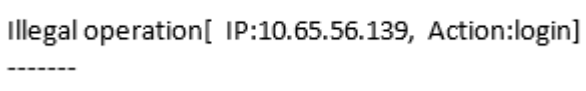


CTS records the logins and logouts of users on IAM.

5.5 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 5-4**.

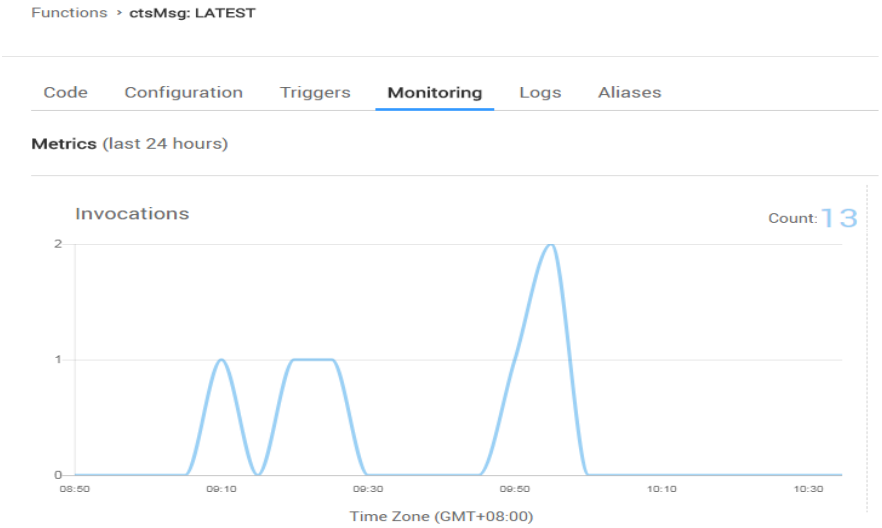
Figure 5-4 Email notification



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 5-5**.

Figure 5-5 Function metrics



6 Periodically Starting or Stopping Huawei Cloud ECSs

Application Scenario

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

- Startup node: VM that needs to be started periodically.
- Shutdown node: VM that needs to be stopped periodically.

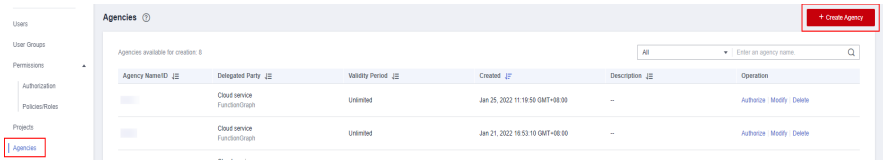
Prerequisites

1. Obtain the program package for periodically **starting** or **stopping** ECSs.
2. Create the **EcsOperation** agency and grant it the **ECS FullAccess** permission. For details, see **Creating an Agency**.

Creating an Agency

1. Log in to the **IAM console**.
2. On the IAM console, choose **Agencies** from the navigation pane, and click **Create Agency** in the upper right corner.

   Figure 6-1 Creating an agency

3. Configure the agency.
   - For **Agency Name**, enter **EcsOperation**.
   - For **Agency Type**, select **Cloud service**.
   - For **Cloud Service**, select **FunctionGraph**.
   - For **Validity Period**, select **Unlimited**.
   - **Description**: Enter the description.

4. Click **Next**. On the displayed page, search for **ECS FullAccess** in the search box on the right and select it.

   Figure 6-2 Selecting the permission

5. Click **Next** and select the application scope of the permission based on service requirements.

Building a Program

Step 1 Create a function.

To create a function for periodically starting or stopping ECSs, upload the program package (for **starting** or **stopping** ECSs) and select the **EcsOperation** agency. For details, see **Creating a Function**.

Select the Python 3.6 runtime and the agency **EcsOperation** created in the previous step.

Step 2 Set environment variables.

On the **Configuration** tab page, set environment variables according to **Table 6-1**.

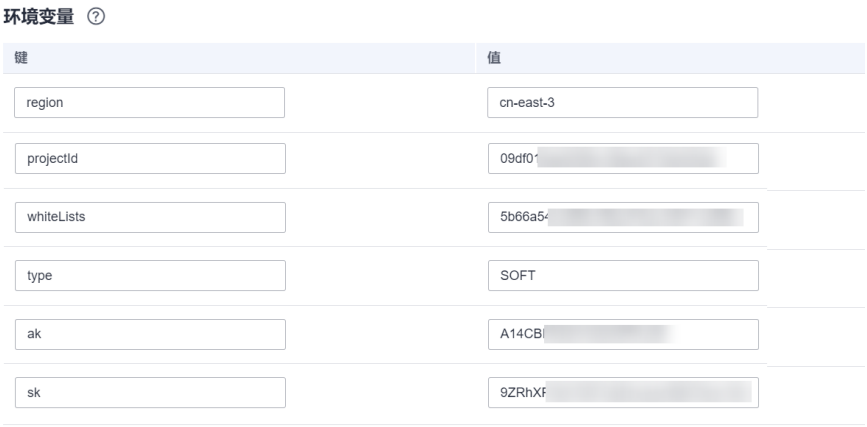
Table 6-1 Environment variables

Environment Variable	Description
region	Region where your ECSs are located, for example, **ap-southeast-1**.
projectId	ID of the project to which the ECS belongs. For details about how to obtain the project ID, see **Obtaining a Project ID**.
whiteLists	• If you want to periodically **start** certain ECSs, specify the IDs of the ECSs that need to be started and **separate them with commas (,)**. • If you want to periodically **stop** certain ECSs, specify the IDs of the ECSs that need to be stopped and **separate them with commas (,)**.
type	Stop type, which needs to be configured when you want to periodically start ECSs. Options: **SOFT**: normal ECS stop (default) **HARD**: forcible ECS stop

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

NOTE

- In the current example, there are no requirements on the region where the function is executed. If the function and the ECS to be started or stopped are in the same region, perform the preceding operations. If they are in different regions, for example, the function is running in CN-Hong Kong and the ECS to be started or stopped is located in AP-Bangkok, change the values of **projectId** and **region** to those of AP-Bangkok, **obtain an AK/SK** and add them to the environment variables, and then delete the configured agency.
  - In AK/SK-based authentication, AK/SK is used to sign requests and the signature is then added to the request headers for authentication.
  - AK: access key ID, which is a unique identifier used in conjunction with a secret access key to sign requests cryptographically.
  - SK: secret access key used together with an AK to sign requests cryptographically. It identifies a request sender and prevents the request from being modified.



- If a large number of ECSs need to be started or stopped, increase the execution timeout for your function.
- Environment variables in **Table 6-1** except **whiteLists** are mandatory. **whiteLists** indicates the comma-separated IDs of the ECSs to be started or stopped.
- Endpoint of the ECS service in the format of "{region}.{domain}", for example, **ap-southeast-1.myhuaweicloud.com**. To obtain the endpoint information, see **Regions and Endpoints**.

Step 3 Add a dependency.

On the **Code** tab, add dependency **huaweicloudsdk_ecs_core_py3.6**.

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

NOTE

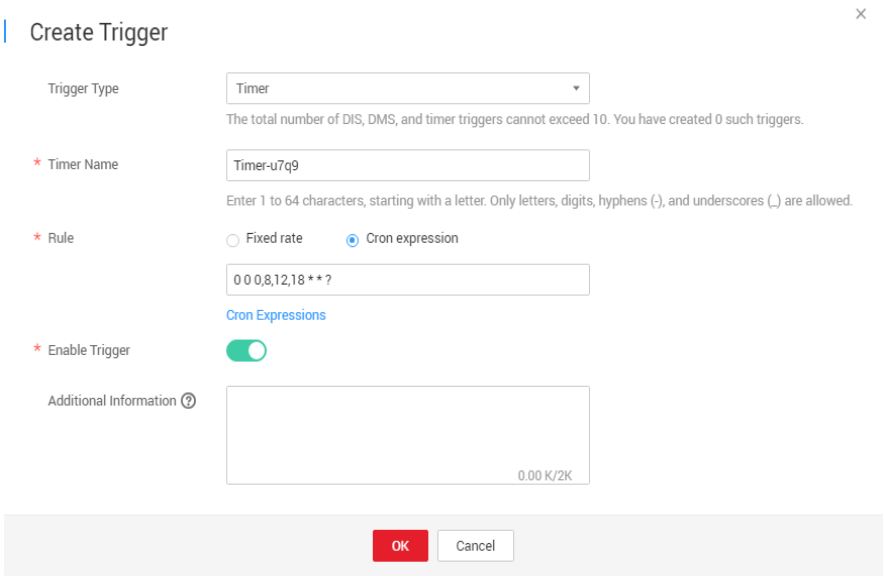
If **huaweicloudsdk_ecs_core_py3.6** is not available in your region, contact customer service.

----End

Adding an Event Source

Create a timer trigger and set the trigger parameters according to **Figure 6-3**.

Figure 6-3 Creating a timer trigger



7 Building an HTTP Function with Spring Boot

Introduction

This chapter describes how to deploy services on FunctionGraph using Spring Boot.

Usually, you may build Spring Boot applications using **SpringInitializr** or IntelliJ IDEA. This chapter uses the Spring.io project in **https://spring.io/guides/gs/rest-service/** as an example to deploy an HTTP function on FunctionGraph.

Procedure

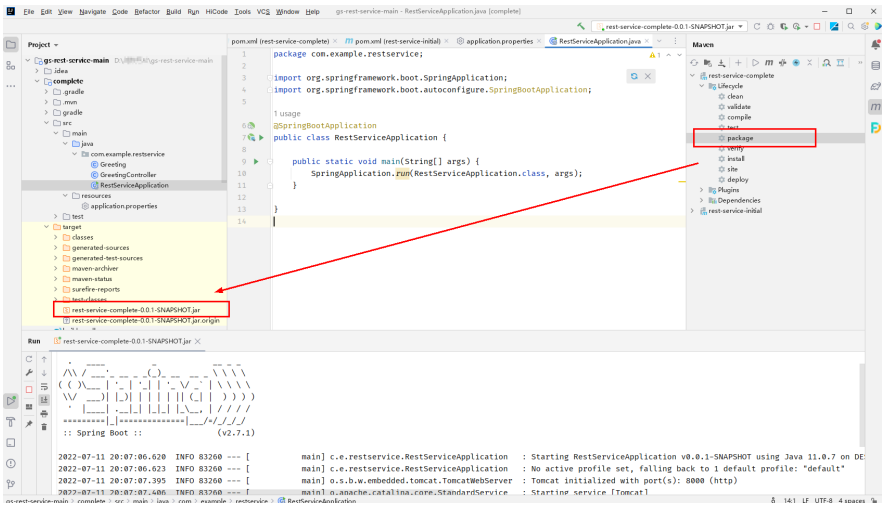
To deploy an existing project to FunctionGraph, change the listening port of the project to **8000**, and create a file named **bootstrap** in the same directory as the JAR file to include the command for executing the JAR file.

In this example, a Maven project created using IntelliJ IDEA is used.

Building a Code Package

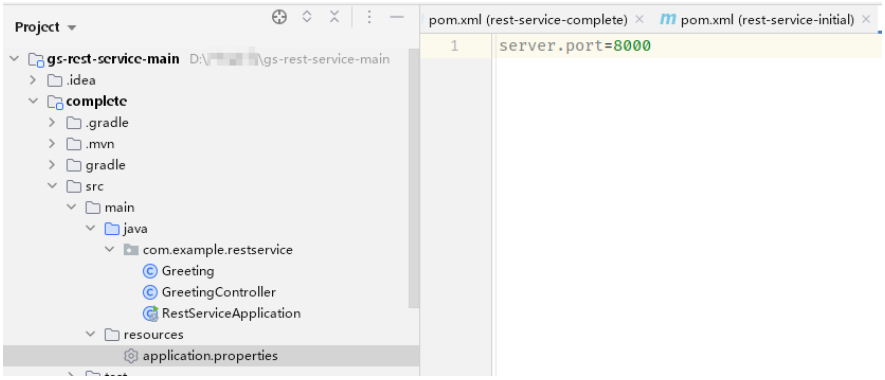
1. Open the Spring Boot project and click **package** in the **Maven** area to generate a JAR file.

Figure 7-1 Generating a JAR file



2. Set the web port to **8000 (do not change this port)** using the **application.properties** file or specify the port during startup. HTTP functions only support this port.

   Figure 7-2 Configuring port 8000

3. Create a file named **bootstrap** in the same directory as the JAR file, and enter the startup parameters.

   ```
   /opt/function/runtime/java11/rtsp/jre/bin/java -jar -Dfile.encoding=utf-8 /opt/function/code/rest-service-complete-0.0.1-SNAPSHOT.jar
   ```

4. Compress the JAR file and **bootstrap** file into a ZIP package.

Creating an HTTP Function and Uploading Code

Create an HTTP function and upload the ZIP file. For details, see **Creating an HTTP Function**.

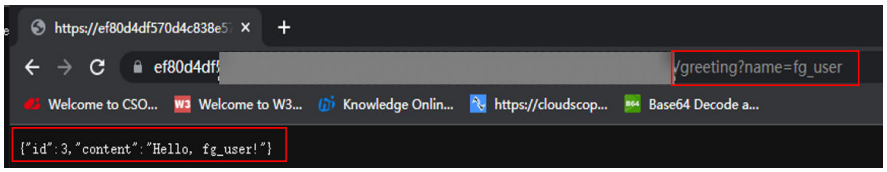
Verifying the Result

- Using a test event
  a. On the function details page, select a version and click **Configure Test Event**.
  b. On the **Configure Test Event** page, select the event template **apig-event-template**, and modify the **path** and **pathParameters** parameters in the template to construct a simple GET request.
  c. Click **Create**.
  d. Click **Test** to obtain the response.

     When debugging a function, increase the memory size and timeout, for example, increase them to 512 MB and 5s.

- Using an APIG trigger
  a. Create an APIG trigger by referring to **Using an APIG Trigger**. Set the authentication mode to **None** for debugging.
  b. Copy the generated URL, add the request parameter **greeting?name=fg_user** to the end of the URL (see **Figure 7-3**), and access the URL using a browser. The response shown in the following figure is displayed.

Figure 7-3 Invoking the function



The default APIG trigger URL is in the format "*Domain name*/*Function name*". In this example, the URL is **https://***your_host.com*/**springboot_demo**, where the function name **springboot_demo** is the first part of the path. If you send a GET request for **https://***your_host.com*/**springboot_demo/greeting**, the request address received by Spring Boot contains **springboot_demo/greeting**. If you have uploaded an existing project, you cannot access your own services because the path contains a function name. To prevent this from happening, use either of the following methods to annotate or remove the function name:

- Method 1: Modify the mapping address in the code. For example, add the first part of the default path to the GetMapping or class annotation.

  Figure 7-4 Modifying the mapping address

  ```
  @RestController
  public class GreetingController {

      1 usage
      private static final String template = "Hello, %s!";
      1 usage
      private final AtomicLong counter = new AtomicLong();

      @GetMapping("/springboot_demo/greeting")
      public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
          return new Greeting(counter.incrementAndGet(), String.format(template, name));
      }
  }
  ```

- Method 2: Click the trigger name to go to the APIG console, and delete the function name in the path.

FAQ

1. What Directories Are Accessible to My Code?

   An uploaded code package is stored in the **/opt/function/code/** directory of the function (runtime environments, compute resources, or containers). However, the directory can only be read and cannot be written. If some data must be written to the function during code running and logged locally, or your dependency is written by default to the directory where the JAR file is located, use the **/tmp** directory.

2. How Are My Logs Collected and Output?

   Function instances that have not received any requests during a specific period of time will be deleted together with their local logs. You will be unable to view the function logs during function running. Therefore, in

addition to writing logs to your local host, output logs to the console by setting the output target of Log4j to **System.out** or by using the **print** function.

Logs output to the console will be collected. If you have enabled LTS, the logs will also be stored in LTS for near real-time analysis.

Suggestion: **Enable LTS**, and click **Go to LTS** to view and analyze logs on the **Real-Time Logs** tab page.

3. What Permissions Does My Code Have?

   Similar to common event functions, code does not have the **root** permission. Code or commands requiring this permission cannot be executed in HTTP functions.

4. How Do I Package Spring Boot Projects of Multiple Modules?

   Configure the following to package these Spring Boot projects.

```
<build>
      <plugins>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
               <mainClass>com.example.YourServiceMainClass</mainClass>
            </configuration>
            <executions>
               <execution>
                  <goals>
                     <goal>repackage</goal>
                  </goals>
               </execution>
            </executions>
         </plugin>
      </plugins>
</build>
```


8 Creating a FunctionGraph Backend API That Uses a Custom Authorizer

8.1 Introduction

In addition to IAM and app authentication, APIG also supports custom authentication with your own system, which can better adapt to your business capabilities.

This chapter guides you through the process of creating a FunctionGraph API that uses a custom authorizer.

Solution

- Log in to the FunctionGraph console, and create a function for custom authentication.
- Create a service function.
- Create an API group on the APIG console.
- Create an API and configure a custom authorizer and a FunctionGraph backend for it.
- Debug the API.

  NOTE

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

  1. An API group storing APIs
  2. A custom authentication function
  3. A service function
  4. An API with a custom authorizer and a FunctionGraph backend

8.2 Resource Planning

Ensure that the following resources are in the same region.

Table 8-1 Resource planning

Resource	Quantity
API group	1
Custom authentication function	1
Service function	1
API	1

8.3 Building a Program

Creating an API group

Before creating a function and adding an event source, create an API group to store and manage APIs.

NOTE

Before enabling APIG functions, buy a gateway by referring to **Buying a Gateway**.

Step 1 Log in to the APIG console, choose **API Management** > **API Groups** in the navigation pane, and click **Create API Group** in the upper right.

Step 2 Select **Create Directly**, set the group information, and click **OK**.

- **Name**: Enter a group name, for example, **APIGroup_test**.
- **Description**: Enter a description about the group.

----End

Creating a Custom Authentication Function

Frontend custom authentication means APIG uses a function to authenticate received API requests. To authenticate API requests by using your own system, create a frontend custom authorizer in APIG. Create a FunctionGraph function with the required authentication information. Then use it to authenticate APIs in APIG.

This section uses the header parameter **event["headers"]** as an example. For the description about request parameters, see **Request Parameter Code Example**.

Step 1 In the left navigation pane of the management console, choose **Compute** > **FunctionGraph** to go to the FunctionGraph console. Then choose **Functions** > **Function List** in the navigation pane.

Step 2 Click **Create Function**.

Step 3 Set the function information, and click **Create Function**.

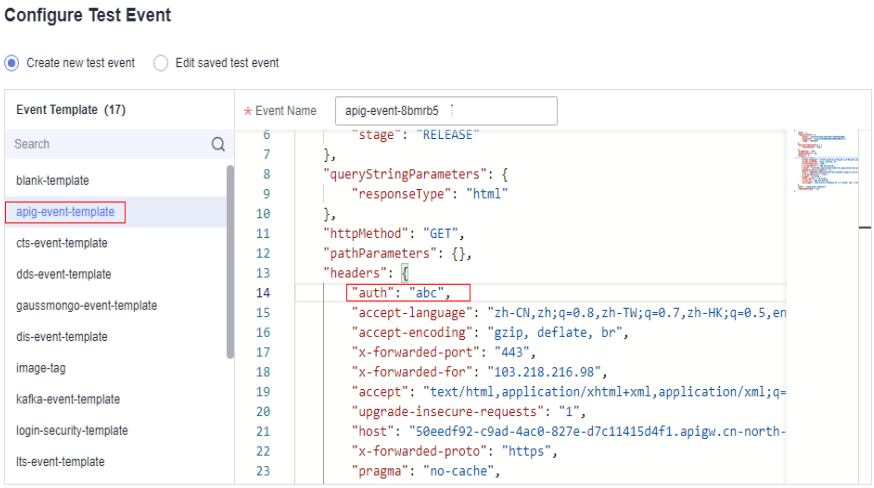
- **Template**: Select **Create from scratch**.

- **Function Name**: Enter a function name, for example, **apig-test**.
- **Agency**: Select **Use no agency**.
- **Runtime**: Select **Python 2.7**.

Step 4 On the function details page that is displayed, click the **Code** tab and copy the **example request parameter code** to the online editor, and click **Deploy**.

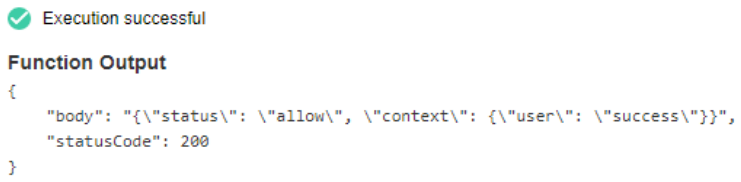
Step 5 Click **Configure Test Event**, and select **apig-event-template**. Modify the template as required, and click **Create**. In this example, add **"auth":"abc"** to **"headers"**.

Figure 8-1 Configuring a test event



Step 6 Click **Test**. If the result is **Execution successful**, the function is successfully created.

Figure 8-2 Viewing the execution result



----End

Creating a Custom Authorizer

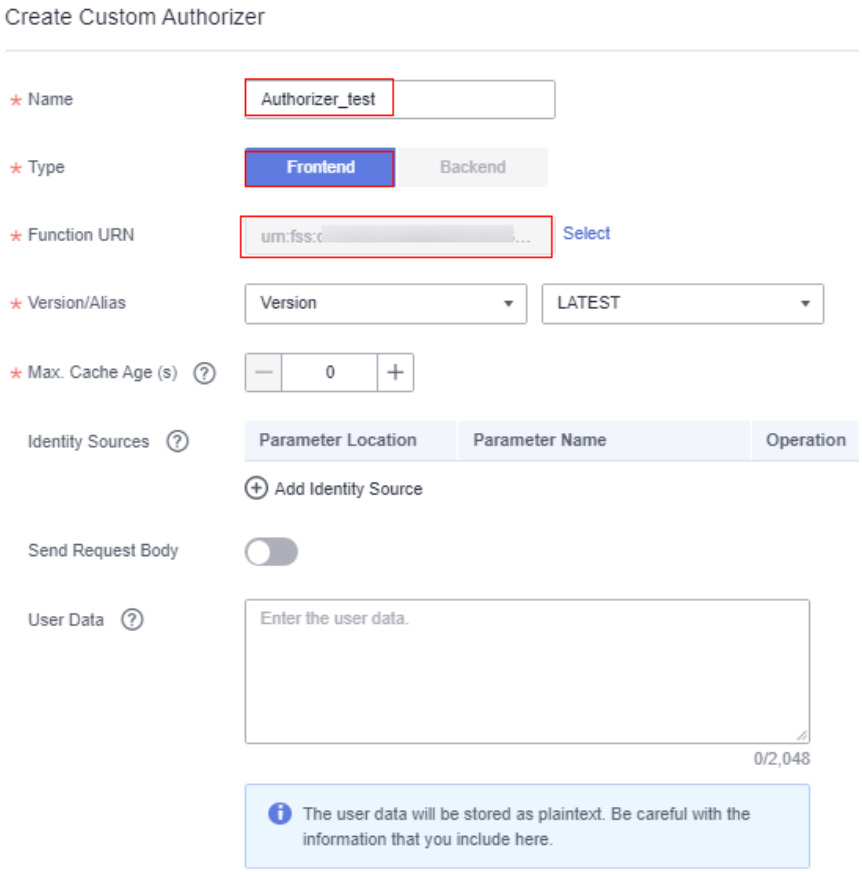
Create a custom authorizer in APIG and connect it to the frontend custom authentication function.

Step 1 In the left navigation pane of the management console, choose **Middleware** > **API Gateway** to go to the APIG console. In the navigation pane, choose **API Management** > **API Policies**. On the **Custom Authorizers** tab, click **Create Custom Authorizer**.

Step 2 Configure basic information about the custom authorizer according to the following figure.

- **Name**: Enter a name, for example, **Authorizer_test**.
- **Type**: Select **Frontend**.
- **Function URN**: Select **apig-test**.

Figure 8-3 Creating a custom authorizer



Step 3 Click **OK**.

----End

Creating a Backend Service Function

APIG supports FunctionGraph backends. After you create a FunctionGraph backend API, APIG will trigger the relevant function, and the function execution result will be returned to APIG.

Step 1 Create a service function by referring to **Creating a Custom Authentication Function**. The function name must be unique.

Step 2 On the **Code** tab of the function details page, copy the following code to the online editor, and click **Deploy**.

```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    body = "<html><title>Functiongraph Demo</title><body><p>Hello, FunctionGraph!</p></body></html>"
    print(body)
```



```
return {
    "statusCode":200,
    "body":body,
    "headers": {
        "Content-Type": "text/html",
    },
    "isBase64Encoded": False
}
```

----End

Request Parameter Code Example

The following are the requirements you must meet when developing FunctionGraph functions. Python 2.7 is used as an example.

The function must have a clear API definition. Example:

def handler (event, context)

- **handler**: name of the entry point function. The name must be consistent with that you define when creating a function.
- **event**: event parameter defined in JSON format for the function.
- **context**: runtime information provided for executing the function. For details, see **SDK APIs**.

event supports three types of request parameters in the following formats:

- Header parameter: **event["headers"]["***Parameter name***"]**
- Query string: **event["queryStringParameters"]["***Parameter name***"]**
- Custom user data: **event["user_data"]**

The three types of request parameters obtained by the function are mapped to the custom authentication parameters defined in APIG.

- Header parameter: Corresponds to the identity source specified in **Header** for custom authentication. The parameter value is transferred when the API that uses custom authentication is called.
- Query string: Corresponds to the identity source specified in **Query** for custom authentication. The parameter value is transferred when the API that uses custom authentication is called.
- Custom user data: Corresponds to the user data for custom authentication. The parameter value is specified when the custom authorizer is created.
- The function response cannot be greater than 1 MB and must be in the following format:

```
{    "statusCode":200,
     "body": "{\"status\": \"allow\", \"context\": {\"user\": \"abc\"}}"
 }
```

The **body** field is a character string, which is JSON-decoded as follows:

```
{
    "status": "allow/deny",
    "context": {
        "user": "abc"
    }
}
```


The **status** field is mandatory and is used to identify the authentication result. The authentication result can only be **allow** or **deny**. **allow** indicates that the authentication is successful, and **deny** indicates that the authentication fails.

The **context** field is optional and can only be key-value pairs. The key value cannot be a JSON object or an array.

The **context** field contains custom user data. After successful authentication, the user data is mapped to the backend parameters. The parameter name in **context** is case-sensitive and must be the same as the system parameter name. The parameter name must start with a letter and can contain 1 to 32 characters, including letters, digits, hyphens (-), and underscores (_).

Example Header Parameter

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event["headers"].get("auth")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"success"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

Example Query String

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event["queryStringParameters"].get("test")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"abcd"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

Example User Data

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event.get("user_data")=='abc':
```



```
    resp = {
        'statusCode': 200,
        'body': json.dumps({
            "status":"allow",
            "context":{
                "user":"abcd"
            }
        })
    }
else:
    resp = {
        'statusCode': 200,
        'body': json.dumps({
            "status":"deny",
        })
    }
return json.dumps(resp)
```

8.4 Adding an Event Source

Creating an API

After creating an API group, custom authentication function, and backend function, create a FunctionGraph backend API that uses a custom authorizer by performing the following steps:

Step 1 Log in to the APIG console, choose **API Management** > **APIs** in the navigation pane, and click **Create API** in the upper right.

Step 2 Configure the basic information according to **Figure 8-4** and **Figure 8-5**.

- **API Name**: Enter a name, for example, **API_test**.
- **Group**: Select API group **APIGroup_test**.
- **URL**: Set **Method** to **ANY**, **Protocol** to **HTTPS**, and **Path** to **/testAPI**.
- **Gateway Response**: Select **default**.
- **Authentication Mode**: Select **Custom**.
- **Custom Authorizer**: Select **Authorizer_test**.

Figure 8-4 Configuring frontend definition

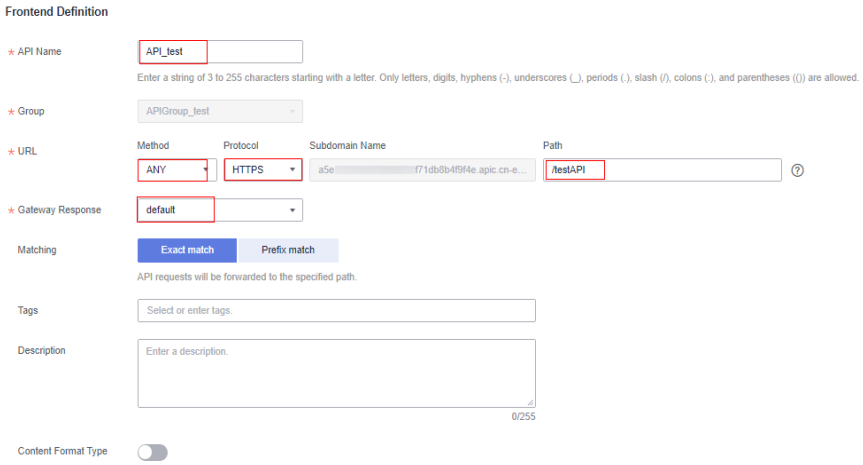
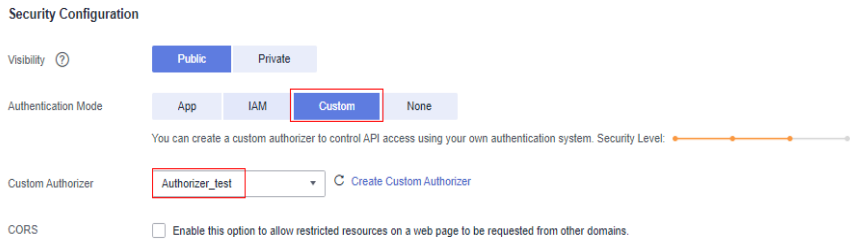


Figure 8-5 Configuring security settings



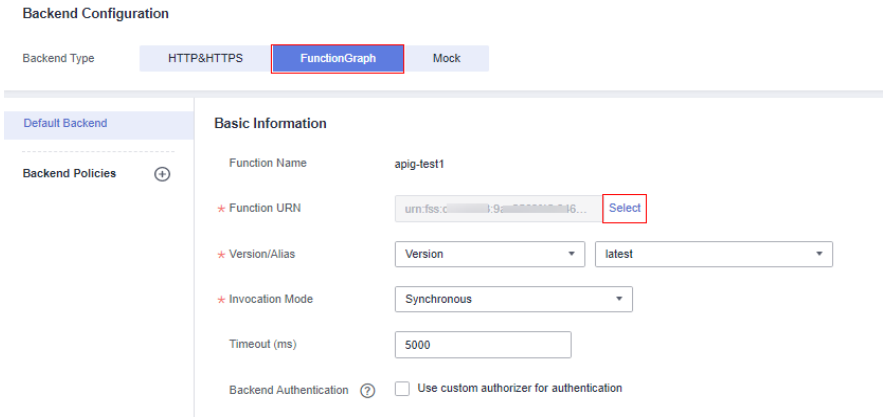
NOTE

For more parameters, see **Creating an API**.

Step 3 Click **Next** to configure the backend service according to **Figure 8-6**.

- **Backend Type**: Select **FunctionGraph**.
- **Function URN**: Select the created service function.
- **Version/Alias**: Select the **latest** version.
- **Invocation Mode**: Select **Synchronous**.

Figure 8-6 Configuring the backend service



Step 4 Click **Finish**.

Step 5 Click **Publish** to publish the API in the RELEASE environment.

Figure 8-7 Publishing an API

----End

8.5 Debugging and Calling the API

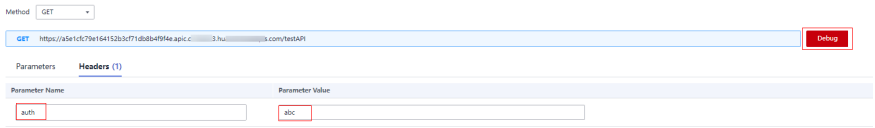
APIG provides online debugging, enabling you to check an API after configuring it.

Step 1 Log in to the APIG console. In the navigation pane, choose **API Management** > **APIs**. Then click **API_test**, and click **Debug**.

Step 2 Add a header parameter and click **Debug**.

- **Parameter Name**: Enter **auth**.
- **Parameter Value**: Enter **abc**.

Figure 8-8 Adding a header



Step 3 Check whether the API response contains the content you have defined in the service function. See **Figure 8-9**.

Figure 8-9 API response

```
HTTP/1.1 200 OK
Content-Length: 87
Connection: keep-alive
Content-Type: text/html; charset=UTF-8
Date: Tue, 07 Feb 2023 06:39:18 GMT
Server: api-gateway
Strict-Transport-Security: max-age=31536000; includeSubdomains;
X-Apig-Latency: 2140
X-Apig-Ratelimit-Api: remain:99,limit:100,time:1 minute
X-Apig-Ratelimit-Api: remain:15601,limit:16000,time:1 second
X-Apig-Ratelimit-Api-Allenv: remain:5999,limit:6000,time:1 second
X-Apig-Ratelimit-Api-Allenv: remain:15601,limit:16000,time:1 second
X-Apig-Ratelimit-User: remain:9870,limit:10000,time:1 second
X-Apig-Upstream-Latency: 1539
X-Cff-Billing-Duration: 5
X-Cff-Invoke-Summary: {"funcDigest":"64999f78efbc98714f57b3f190573be","duration":4.952,"billingDuration":5,"memo
rySize":128,"memoryUsed":25.906,"podName":"pool22-300-128-fusion-67fc9b8d95-s6rsv"}
X-Cff-Request-Id: 495bcdf5-d474-4aa5-ba04-c79f84d4367c
X-Content-Type-Options: nosniff
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Request-Id: dfa7d5925751f31f12221f45459a1312
X-Xss-Protection: 1; mode=block;

<html><title>Functiongraph Demo</title><body><p>Hello, FunctionGraph!</p></body></html>
```

----End

9 Uploading Files with FunctionGraph and APIG

9.1 Introduction

Scenario

Uploading files, such as run logs and web application images, from devices to cloud servers is a type of common scenarios for websites and applications. These scenarios can be implemented by using function backends and APIG. This chapter uses Node.js and Python as examples to describe how to develop a backend parsing function for obtaining uploaded files.

Constraints

- The file uploaded in a request cannot exceed 6 MB.
- Function logic processing must be within 15 minutes.

9.2 Resource Planning

Table 9-1 Resource planning

Product	Configuration Example
APIG	• Region: AP-Singapore • Specifications: shared gateway or dedicated gateway
FunctionGraph	• Region: AP-Singapore • Billing mode: pay-per-use

9.3 Procedure

This solution includes the following steps:

1. Create a function to receive and parse uploaded files.
2. Bind an APIG trigger to the function for E2E testing.

9.3.1 Node.js

Prerequisites

- You have a Huawei Cloud account and have completed real-name authentication.
- Your Huawei Cloud account is not in arrears and has sufficient balance for the resources involved in this example.

Procedure

Step 1 Create a function.

1. Log in to the **FunctionGraph console**, choose **Functions** > **Function List** in the navigation pane, and click **Create Function**.
2. Select **Create from Scratch**, set the function information, and click **Create Function**.
   - **Function Type**: Select **Event Function**.
   - **Region**: Select **AP-Singapore**.
   - **Function Name**: Enter **upload-file-1**.
   - **Agency**: Select **Use no agency**.
   - **Runtime**: Select **Node.js 14.18**.
3. On the **Code** tab of the function details page, copy the following code to replace the default code, and click **Deploy**.

```
const stream = require("stream");
const Busboy = require("busboy");

exports.handler = async (event, context) => {
    const logger = context.getLogger()
    logger.info("Function start run.");
    if (!("content-type" in event.headers) ||
        !event.headers["content-type"].includes("multipart/form-data")) {
        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json'
            },
            'body': 'The request is not in multipart/form-data format.',
        };
    }

    const busboy = Busboy({ headers: event.headers });
    let buf = Buffer.alloc(0);
    busboy.on('file', function (fieldname, file, filename, encoding, mimetype) {
        logger.info('filename:' + JSON.stringify(filename))
        file.on('data', function (data) {
            logger.info('Obtains ' + data.length + ' bytes of data.')
            buf = Buffer.concat([buf, data]);
        });
        file.on('end', function () {
            logger.info('End data reception');
        });
    });

    busboy.on('finish', function () {
```



```
    // Data is processed here.
    logger.info(buf.toString());
    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'application/json'
        },
        'body': 'ok',
    };
});

// The APIG trigger encodes data using Base64 by default. The data is decoded here.
const body = Buffer.from(event.body, "base64");
var bodyStream = new stream.PassThrough();
bodyStream.end(body);
bodyStream.pipe(busboy);
}
```

Step 2 Configure a dependency.

1. Make dependency: To parse uploaded files with busboy, generate dependency **busboy.zip** for Node.js 14.18. If you use another Node.js version, create the corresponding dependency by referring to **Creating a Dependency**.
2. Create dependency: In the navigation pane, choose **Functions** > **Dependencies**. Then click **Create Dependency**, configure the dependency information, and click **OK**.
   - **Name**: Enter **busboy**.
   - **Code Entry Mode**: Select **Upload ZIP**.
   - **Runtime**: Select **Node.js 14.18**.
   - **Upload File**: Upload the dependency you made.
3. Add dependency: On the details page of function **upload-file-1**, click **Add** at the bottom of the **Code** tab. On the **Select Dependency** page, set **Type** to **Private**, select the **busboy** dependency, and click **OK**.

Step 3 Create an APIG trigger.

1. On the details page of function **upload-file-1**, choose **Configuration** > **Triggers**.
2. Click **Create Trigger**, and set **Trigger Type** to **API Gateway (APIG)** or **API Gateway (Dedicated Gateway)**. In this example, select **API Gateway (APIG)**.
   - **API Name**: Retain the default name.
   - **API Group**: If no API group is available, click **Create API Group** to create one.
   - **Environment**: Select **RELEASE**.
   - **Security Authentication**: In this example, select **None** for testing. You can select a more secure authentication mode, such as **IAM**, for your own services.
   - **Protocol**: Select **HTTPS**.
   - **Timeout (ms)**: Retain the default value **5000**.

Step 4 Perform E2E testing.

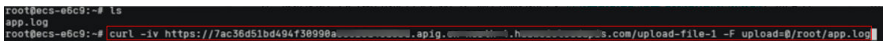
The curl tool is used as an example (**curl -F** is mainly used in Linux). You can also use other tools such as Postman. Create a file named **app.log** with any content on your local host. Example:


```
start something
run
stop all
```

Run the following command:

```
curl -iv {APIG trigger URL} -F upload=@/{Local file path}/app.log
```

Figure 9-1 Example



On the **Monitoring** tab of the **upload-file-1** function details page, view the file content in the logs. If needed, you can modify the code to save data to OBS or LTS or to directly process the data.

----End

9.3.2 Python

Prerequisites

- You have a Huawei Cloud account and have completed real-name authentication.
- Your Huawei Cloud account is not in arrears and has sufficient balance for the resources involved in this example.

Procedure

Step 1 Create a function.

1. Log in to the **FunctionGraph console**, choose **Functions** > **Function List** in the navigation pane, and click **Create Function**.
2. Select **Create from Scratch**, set the function information, and click **Create Function**.
   - **Function Type**: Select **Event Function**.
   - **Region**: Select **AP-Singapore**.
   - **Function Name**: Enter **upload-file-2**.
   - **Agency**: Select **Use no agency**.
   - **Runtime**: Select **Python 3.6**.
3. On the **Code** tab of the function details page, copy the following code to replace the default code, and click **Deploy**.

```
# -*- coding: utf-8 -*-

from requests_toolbelt.multipart import decoder
import base64

def handler(event, context):
    context.getLogger().info("Function start run.")

    content_type = ''
    if "content-type" in event['headers']:
        content_type = event['headers']['content-type']

    if "multipart/form-data" not in content_type:
        return {
```



```
        "statusCode": 200,
        "body": "The request is not in multipart/form-data format.",
        "headers": {
            "Content-Type": "application/json"
        }
    }

body = event['body']
# The APIG trigger encodes data using Base64 by default. The data is decoded here.
raw_data = base64.b64decode(body)
for part in decoder.MultipartDecoder(raw_data, content_type).parts:
    # Data is processed here.
    context.getLogger().info(part.content)

return {
    "statusCode": 200,
    "body": "ok",
    "headers": {
        "Content-Type": "application/json"
    }
}
```

Step 2 Create an APIG trigger.

1. On the details page of function **upload-file-1**, choose **Configuration** > **Triggers**.
2. Click **Create Trigger**, and set **Trigger Type** to **API Gateway (APIG)** or **API Gateway (Dedicated Gateway)**. The shared gateway is used as an example.
   - **API Name**: Retain the default name.
   - **API Group**: If no API group is available, click **Create API Group** to create one.
   - **Environment**: Select **RELEASE**.
   - **Security Authentication**: In this example, select **None** for testing. You can select a more secure authentication mode, such as **IAM**, for your own services.
   - **Protocol**: Select **HTTPS**.
   - **Timeout (ms)**: Retain the default value **5000**.

Step 3 Perform E2E testing.

The curl tool is used as an example (**curl -F** is mainly used in Linux). You can also use other tools such as Postman. Create a file named **app.log** with any content on your local host. Example:

```
start something
run
stop all
```

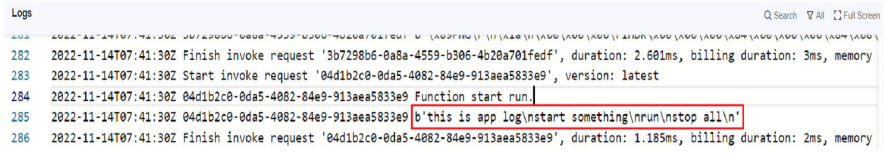
Run the following command:

```
curl -iv {APIG trigger URL} -F upload=@/{Local file path}/app.log
```

Figure 9-2 Example

On the **Monitoring** tab of the **upload-file-2** function details page, view the file content in the logs. If needed, you can modify the code to save data to OBS or LTS or to directly process the data.

Figure 9-3 View logs



----End

10 Processing IoT Data

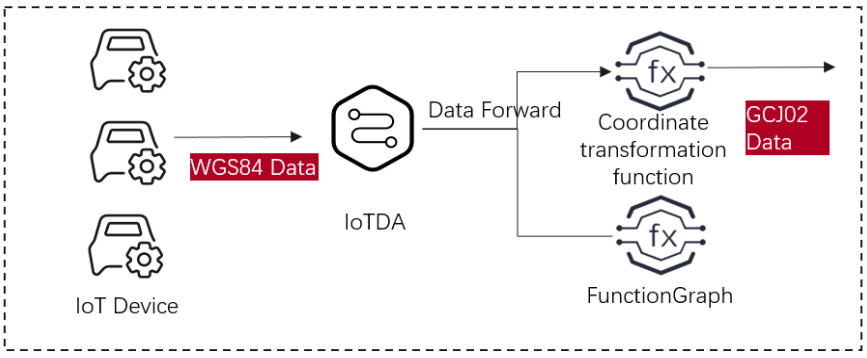
10.1 Introduction

Scenarios

This section demonstrates how to combine FunctionGraph and IoT Device Access (IoTDA) to process status data reported by IoT devices. IoT devices are managed on the IoTDA platform. Data generated by the devices is transferred from IoTDA to trigger the FunctionGraph functions you have compiled for processing.

This combination is suitable for processing device data and storing them to OBS, structuring and cleansing data and storing them to a database, and sending event notifications for device status changes.

This best practice focuses on how to combine IoTDA and FunctionGraph. For details about how to manage devices and report data using IoTDA, see the documentation of IoTDA. In this chapter, we use IoTDA and FunctionGraph to convert WGS84 coordinates to GCJ02.



Procedure

- Create an IoTDA instance in IoTDA. (The standard edition is free of charge. You can use it for testing purposes.)
- Create a function in FunctionGraph.

- Set forwarding rules in IoTDA or create an IoTDA trigger in FunctionGraph.
- Send test messages using forwarding rules.

10.2 Preparation

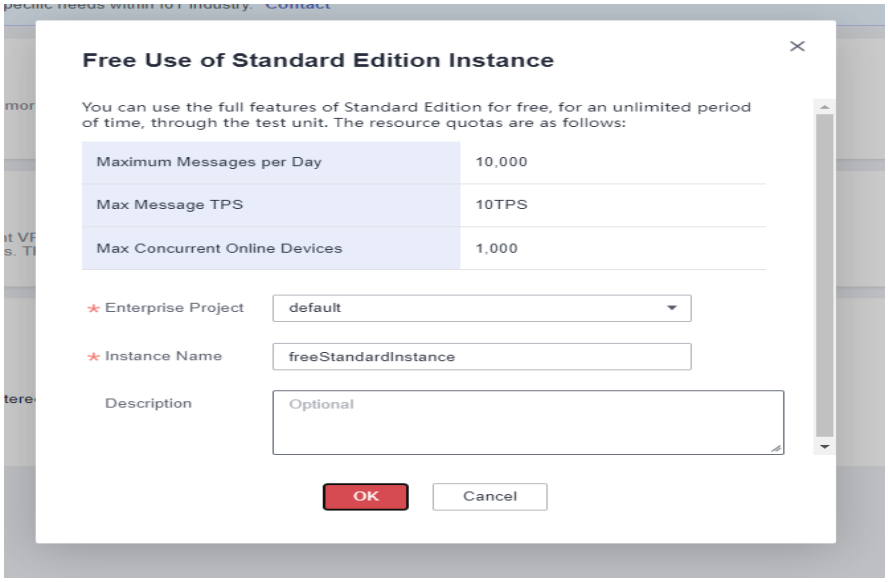
Before creating a forwarding rule, create an IoTDA instance as well as products and devices. In this best practice, we only create an instance for testing.

Creating an IoTDA Instance

Step 1 Log in to the IoTDA console. In the navigation pane, choose **IoTDA Instances**.

Step 2 On the right of the **IoTDA Instances** page, click **Enable Free Standard Edition**. The parameter configuration page is displayed. Set the parameters based on service requirements.

Figure 10-1 Enabling free standard edition



Step 3 Click **Create**.

----End

Creating a Function

Step 1 In the left navigation pane of the management console, choose **Compute** > **FunctionGraph**. On the FunctionGraph console, click **Create Function**.

Step 2 Enter **iotdemo** for **Function Name**, select a runtime (for example, **Python 3.9**), and click **Create**.

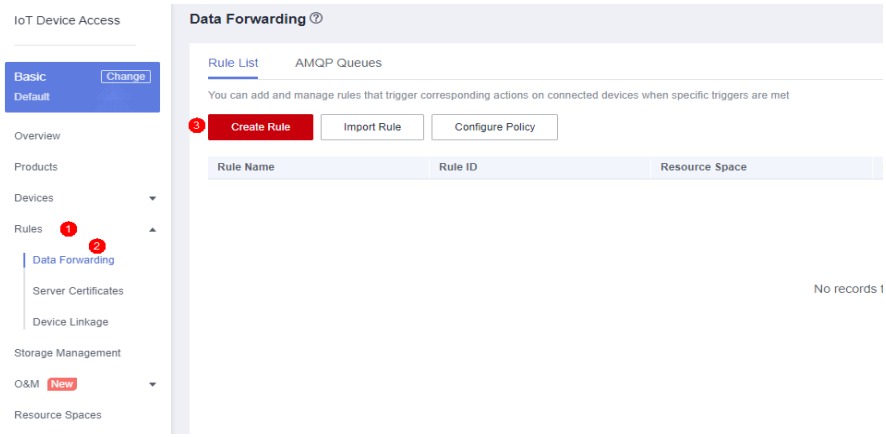
----End

Creating a Forwarding Rule

Forwarding rules are used to transfer data from IoTDA to trigger specified functions. For this purpose, you can create forwarding rules in IoTDA or create an

IoTDA trigger in FunctionGraph. Perform the following procedure to create a forwarding rule:

Step 1 In the left navigation pane of the management console, choose **IoT** > **IoTDA Device Access**. On the IoTDA console, choose **Rules** > **Data Forwarding**, and click **Create Rule**.



Step 2 Enter basic information and click **Create Rule**.

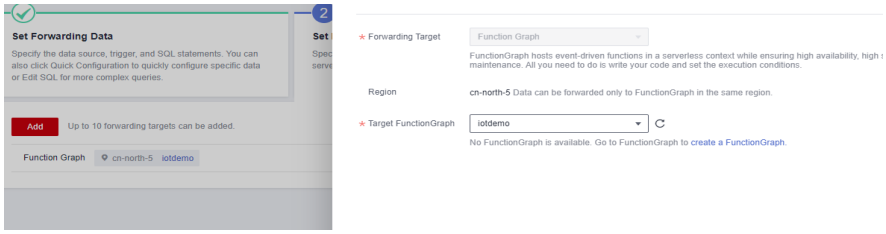
NOTE

- Set **Rule Name** to **iotfg** or another one.
- For **Data Source**, select **Device message**.
- For **Trigger**, select device message reporting.

Step 3 To set the forwarding target, click **Add**, and select **FunctionGraph**.

Step 4 If this is the first time you select **FunctionGraph**, authorize access to IoTDA.

Step 5 Select function **iotdemo**.



Step 6 Start the rule.

----End

10.3 Building a Program

Editing a Function Program

Open function **iotdemo**, copy the following coordinate conversion code to the function. This code is for testing purposes only and can be modified if needed.

```
# -*- coding:utf-8 -*-
import json
```



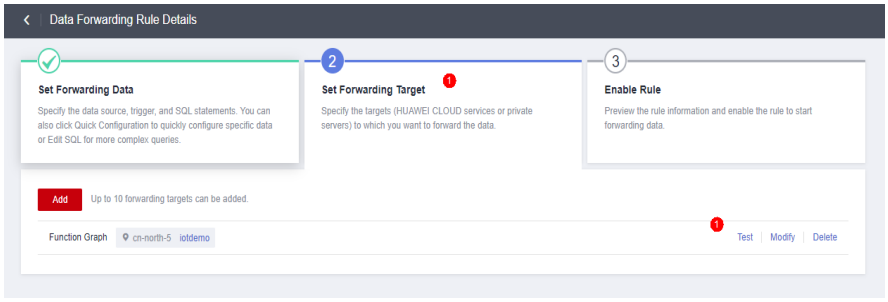
```
import math
from math import pi

def handler(event, context):
    data = event["notify_data"]["body"]
    lat = data["lat"]
    lng = data["lng"]
    print(f" WGS84: ({lng},{lat})")
    gcj_lng, gcj_lat = transform(lng, lat)
    print(f" GCJ02: ({gcj_lng},{gcj_lat})")
    body = {
        "gcj_lng": gcj_lng,
        "gcj_lat": gcj_lat
    }
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(body),
        "headers": {
            "Content-Type": "application/json"
        }
    }

def transform(lon, lat):
    a = 6378245.0
    ee = 0.00669342162296594323

    dlat = transform_lat(lon - 105.0, lat - 35.0)
    dlon = transform_lon(lon - 105.0, lat - 35.0)

    rad_lat = lat / 180.0 * pi
    magic = math.sin(rad_lat)
    magic = 1 - ee * magic * magic
    sqrt_magic = math.sqrt(magic)

    dlat = (dlat * 180.0) / ((a * (1 - ee)) / (magic * sqrt_magic) * pi)
    dlon = (dlon * 180.0) / (a / sqrt_magic * math.cos(rad_lat) * pi)

    mg_lon = lon + dlon
    mg_lat = lat + dlat

    return mg_lon, mg_lat

def transform_lon(x, y):
    ret = 300.0 + x + 2.0 * y + 0.1 * x * x + \
        0.1 * x * y + 0.1 * math.sqrt(math.fabs(x))
    ret += (20.0 * math.sin(6.0 * pi * x) +
            20.0 * math.sin(2.0 * pi * x)) * 2.0 / 3.0
    ret += (20.0 * math.sin(pi * x) +
            40.0 * math.sin(pi / 3.0 * x)) * 2.0 / 3.0
    ret += (150.0 * math.sin(pi / 12.0 * x) +
            300.0 * math.sin(pi / 30.0 * x)) * 2.0 / 3.0
return ret

def transform_lat(x, y):
    ret = -100.0 + 2.0 * x + 3.0 * y + 0.2 * y * y + \
        0.1 * x * y + 0.2 * math.sqrt(math.fabs(x))
    ret += (20.0 * math.sin(6.0 * pi * x) +
            20.0 * math.sin(2.0 * pi * x)) * 2.0 / 3.0
    ret += (20.0 * math.sin(pi * y) +
            40.0 * math.sin(pi / 3.0 * y)) * 2.0 / 3.0
    ret += (160.0 * math.sin(pi / 12.0 * y) +
            320 * math.sin(pi / 30.0 * y)) * 2.0 / 3.0
    return ret
```


Online Joint Commissioning with IoTDA

Step 1 Log in to the IoTDA console. In the navigation pane, choose **Rules** & > **Data Forwarding**. In the **Rule List**, click **View** on the right of the target rule name. The **Data Forwarding Rule Details** page is displayed.

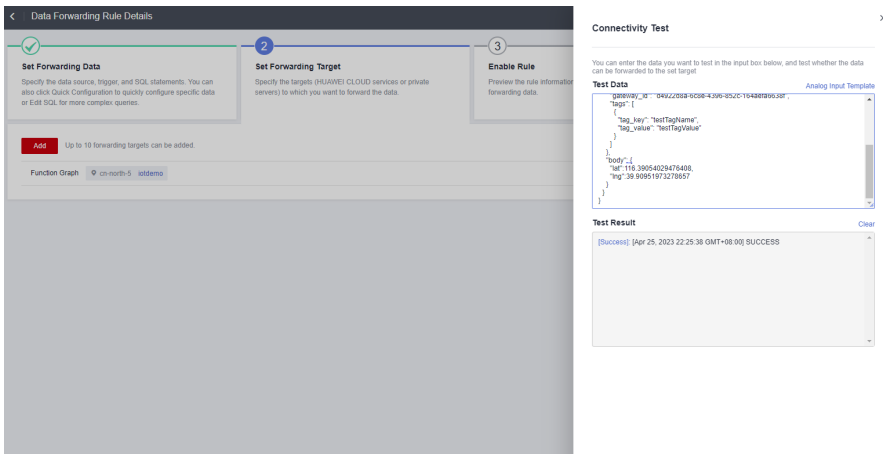
Step 2 Select **Set Forwarding Target** and click **Test** on the right of the forwarding target to edit the test data.

Figure 10-2 Testing the forwarding rule

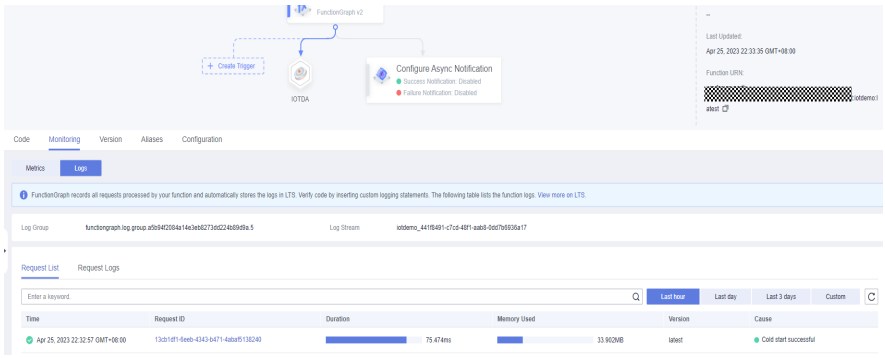


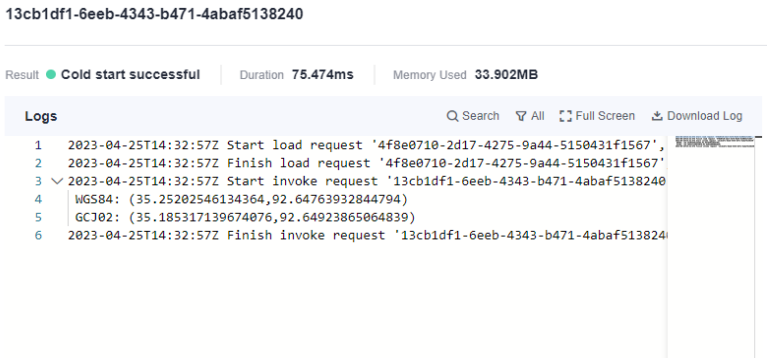
Step 3 Enter the test data and click **Connectivity Test**.

```
{
   "resource": "device.message",
   "event": "report",
   "event_time": "string",
   "notify_data": {
      "header": {
         "app_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
         "device_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
         "node_id": "ABC123456789",
         "product_id": "ABC123456789",
         "gateway_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
         "tags": [
            {
               "tag_key": "testTagName",
               "tag_value": "testTagValue"
            }
         ]
      },
      "body": {
         "lat": 92.64763932844794,
         "lng": 35.25202546134364
      }
   }
}
```



Step 4 Go to the FunctionGraph console, choose **Monitoring** > **Logs**, and click the request ID in blue to view logs.





To invoke other systems, persist data in OBS, or achieve other purposes, modify the program.

----End

11 Function + DEW: Encrypting/Decrypting Files

11.1 Introduction

Huawei Cloud Data Encryption Worksop (DEW) uses the hardware security module (HSM) to protect your keys. All of your keys are protected by the root key in HSM. DEW provides access control and log tracing for all operations on keys, and records key uses to meet audit and compliance requirements. You can buy a dedicated HSM instance to encrypt your service systems (including sensitive data, financial payments, and electronic bills). It encrypts the sensitive data of your enterprise (contracts, transactions, and records) and of users (IDs and mobile numbers). This prevents data breaches and unauthorized access or data tampering by internal users caused by network attacks and data reduction. This chapter describes how to use FunctionGraph and DEW to encrypt and decrypt files.

Scenarios

- Upload files to a specified OBS bucket.
- Encrypt and decrypt each uploaded file.
- Upload the processed files to another OBS bucket.

  NOTE

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

Procedure

- Create two buckets on the OBS console.
- Create a function with an OBS trigger.
- Upload files to one of the buckets.
- Trigger the function to encrypt and decrypt the files.
- The function uploads the processed files to the other bucket.

NOTE

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

1. Two OBS buckets (for storing uploaded and processed files respectively)
2. A file encryption/decryption function
3. An OBS trigger for associating the function with the OBS buckets

11.2 Preparation

Create two OBS buckets to store uploaded and encrypted/decrypted files, respectively.

Create an agency to delegate FunctionGraph to access OBS resources.

Creating OBS Buckets

CAUTION

- The function and the source and destination buckets for storing files must be in the same region.
- Use two different OBS buckets. If only one bucket is used, the function will be executed infinitely. (When a file is uploaded to the bucket, the function is triggered to process the file and store the processed file 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.

- For **Region**, select a region.
- For **Data Redundancy Policy**, select **Single-AZ storage**.
- For **Bucket Name**, enter **dew-bucket-input**.
- For **Default Storage Class**, select **Standard**.
- For **Bucket Policies**, select **Private**.
- For **Direct Reading**, select **Disable**.

Click **Create Now**.

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

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

Step 4 View **dew-bucket-input** and **dew-bucket-output** in the bucket list.

----End

Creating a DEW Key

CAUTION

- The DEW key and function must be in the same region.

Procedure

Step 1 In the left navigation pane of the management console, choose **Security & Compliance** > **Data Encryption Workshop** to go to the DEW console. Then click **Create Key**.

Step 2 On the **Create Key** page, click **OK**.

Step 3 Record the master key ID.

----End

Creating an Agency

Step 1 In the left navigation pane of the management console, choose **Management & Governance** > **Identity and Access Management** to go to the IAM console. Then 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**.
- For **Description**, enter a description.

Step 4 Click **Next**. On the **Select Policy/Role** page, select **Tenant Administrator** and click **Next**.

NOTE

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

Step 5 Click **OK**.

----End

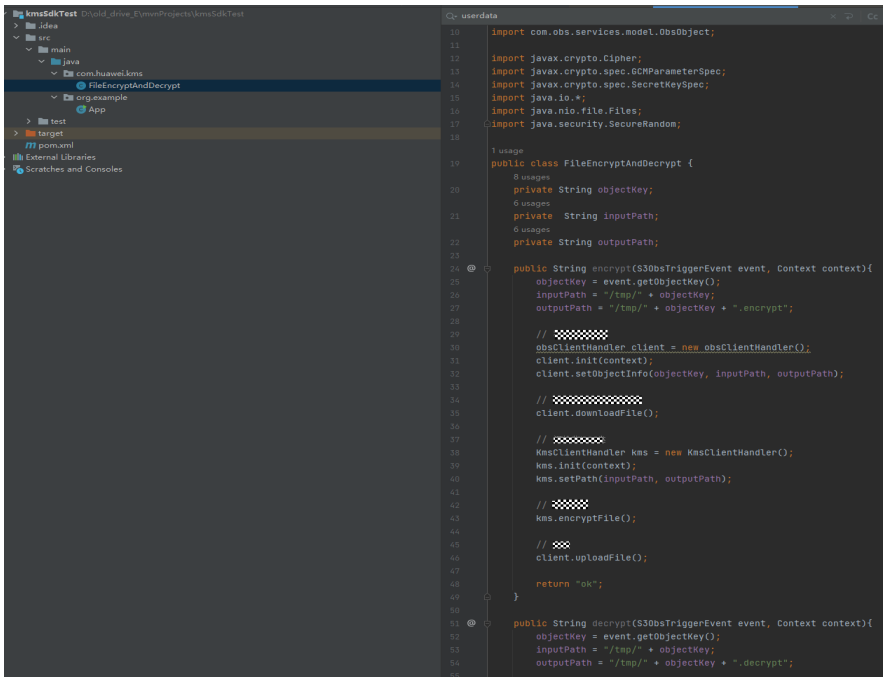
11.3 Building a Program

This section provides a file encryption/decryption package. You can create a function with the sample code in this package.

Creating a Deployment Package

This example uses a Java 8 function to encrypt/decrypt files. For details about function development, see **Developing Functions in Java**. **Figure 11-1** shows the sample code directory. The service code is not described.

Figure 11-1 Sample code directory



FileEncryptAndDecrypt is the function execution entry point. The entry function in **FileEncryptAndDecrypt** contains the following code:

```
package com.huawei.kms;
import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.s3obs.S3ObsTriggerEvent;
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.kms.v1.KmsClient;
import com.huaweicloud.sdk.kms.v1.model.*;
import com.obs.services.ObsClient;
import com.obs.services.exception.ObsException;
import com.obs.services.model.ObsObject;
import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.nio.file.Files;
import java.security.SecureRandom;
public class FileEncryptAndDecrypt {
    private String objectKey;
    private  String inputPath;
    private String outputPath;
    public String encrypt(S3ObsTriggerEvent event, Context context){
        objectKey = event.getObjectKey();
        inputPath = "/tmp/" + objectKey;
        outputPath = "/tmp/" + objectKey + ".encrypt";
        // Initialize OBS class.
        obsClientHandler client = new obsClientHandler();
        client.init(context);
        client.setObjectInfo(objectKey, inputPath, outputPath);
        // Download files from the specified OBS bucket.
```



```
        client.downloadFile();
        // Initialize KMS class.
        KmsClientHandler kms = new KmsClientHandler();
        kms.init(context);
        kms.setPath(inputPath, outputPath);
        // Encrypt files.
        kms.encryptFile();
        // Upload files.
        client.uploadFile();
        return "ok";
    }
    public String decrypt(S3ObsTriggerEvent event, Context context){
        objectKey = event.getObjectKey();
        inputPath = "/tmp/" + objectKey;
        outputPath = "/tmp/" + objectKey + ".decrypt";
        // Initialize OBS class.
        obsClientHandler client = new obsClientHandler();
        client.init(context);
        client.setObjectInfo(objectKey, inputPath, outputPath);
        // Download files from the specified OBS bucket.
        client.downloadFile();
        // Initialize KMS class.
        KmsClientHandler kms = new KmsClientHandler();
        kms.init(context);
        kms.setPath(inputPath, outputPath);
        // Encrypt files.
        kms.decryptFile();
        // Upload files.
        client.uploadFile();
        return "ok";
    }
    static class KmsClientHandler {
        // DEW API version. Currently fixed to v1.0.
        private static final String KMS_INTERFACE_VERSION = "v1.0";
        private static final String AES_KEY_BIT_LENGTH = "256";
        private static final String AES_KEY_BYTE_LENGTH = "32";
        private static final String AES_ALG = "AES/GCM/PKCS5Padding";
        private static final String AES_FLAG = "AES";
        private static final int GCM_TAG_LENGTH = 16;
        private static final int GCM_IV_LENGTH = 12;
        private String ACCESS_KEY;
        private String SECRET_ACCESS_KEY;
        private String PROJECT_ID;
        private String KMS_ENDPOINT;
        private String keyId;
        private String cipherText;
        private String inputPath;
        private String outputPath;
        private Context context;
        private KmsClient kmsClient = null;
        void init(Context context) {
            this.context = context;
        }
        void initKmsClient() {
            if (kmsClient == null) {
                ACCESS_KEY = context.getAccessKey();
                SECRET_ACCESS_KEY = context.getSecretKey();
                PROJECT_ID = context.getProjectID();
                KMS_ENDPOINT = context.getUserData("kms_endpoint");
                keyId = context.getUserData("kms_key_id");
                cipherText = context.getUserData("cipher_text");
                final BasicCredentials auth = new
BasicCredentials().withAk(ACCESS_KEY).withSk(SECRET_ACCESS_KEY).withProjectId(PROJECT_ID);
                kmsClient = kmsClient.newBuilder().withCredential(auth).withEndpoint(KMS_ENDPOINT).build();
            }
        }
        byte[] getEncryptPlainKey() {
            final CreateDatakeyRequest createDatakeyRequest = new
CreateDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
```



```
                .withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));
        final CreateDatakeyResponse createDatakeyResponse =
kmsClient.createDatakey(createDatakeyRequest);
        final String cipherText = createDatakeyResponse.getCipherText();
        return hexToBytes(createDatakeyResponse.getPlainText());
    }
    byte[] hexToBytes(String hexString) {
        final int stringLength = hexString.length();
        assert stringLength > 0;
        final byte[] result = new byte[stringLength / 2];
        int j = 0;
        for (int i = 0; i < stringLength; i += 2) {
            result[j++] = (byte) Integer.parseInt(hexString.substring(i, i + 2), 16);
        }
        return result;
    }
    public void setPath(String inputPath, String outputPath) {
        this.inputPath = inputPath;
        this.outputPath = outputPath;
    }
    public void  encryptFile() {
        final File outEncryptFile = new File(outputPath);
        final File inFile = new File(inputPath);
        final byte[] iv = new byte[GCM_IV_LENGTH];
        final SecureRandom secureRandom = new SecureRandom();
        secureRandom.nextBytes(iv);
        doFileFinal(Cipher.ENCRYPT_MODE, inFile, outEncryptFile, getEncryptPlainKey(), iv);
    }
    byte[] getDecryptPlainKey() {
final CreateDatakeyRequest createDatakeyRequest = new
CreateDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
    .withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));
// Create a data key.
final CreateDatakeyResponse createDatakeyResponse = kmsClient.createDatakey(createDatakeyRequest);
        final DecryptDatakeyRequest decryptDatakeyRequest  = new
DecryptDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
                .withBody(new
DecryptDatakeyRequestBody().withKeyId(keyId).withCipherText(createDatakeyResponse.getCipherText()
).withDatakeyCipherLength(AES_KEY_BYTE_LENGTH));
        return hexToBytes(kmsClient.decryptDatakey(decryptDatakeyRequest).getDataKey());
    }
    public void  decryptFile() {
        final File outEncryptFile = new File(outputPath);
        final File inFile = new File(inputPath);
        final byte[] iv = new byte[GCM_IV_LENGTH];
        final SecureRandom secureRandom = new SecureRandom();
        secureRandom.nextBytes(iv);
        doFileFinal(Cipher.DECRYPT_MODE, inFile, outEncryptFile, getDecryptPlainKey(), iv);
    }
    /**         //     * Encrypt/Decrypt files.         //     *         //     * @param cipherMode Encryption mode.
Options: Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.         //     * @param infile     Files before
encryption/decryption.         //     * @param outFile    Files after encryption/decryption.         //     *
@param keyPlain   Plaintext key         //     * @param iv         Initialize vector.         //     */        void
doFileFinal(int cipherMode, File infile, File outFile, byte[] keyPlain, byte[] iv) {
        try (BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(infile.toPath()));
             BufferedOutputStream bos = new
BufferedOutputStream(Files.newOutputStream(outFile.toPath()))) {
            final byte[] bytIn = new byte[(int) infile.length()];
            final int fileLength = bis.read(bytIn);
            assert fileLength > 0;
            final SecretKeySpec secretKeySpec = new SecretKeySpec(keyPlain, AES_FLAG);
            final Cipher cipher = Cipher.getInstance(AES_ALG);
            final GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(GCM_TAG_LENGTH *
Byte.SIZE, iv);
            cipher.init(cipherMode, secretKeySpec, gcmParameterSpec);
            final byte[] bytOut = cipher.doFinal(bytIn);
            bos.write(bytOut);
```



```
            } catch (Exception e) {
                throw new RuntimeException(e.getMessage());
            }
        }
    }
    static class obsClientHandler {
        private ObsClient obsClient = null;
        private String inputBucketName;
        private String outputBucketName;
        private String objectKey;
        private Context context;
        private String localInPath;
        private String localOutPath;
        public void init(Context context) {
            this.context = context;
        }
        void initObsclient() {
            if (obsClient == null) {
                inputBucketName = context.getUserData("input_bucket");
                outputBucketName = context.getUserData("output_bucket");
                String ACCESS_KEY = context.getAccessKey();
                String SECRET_ACCESS_KEY = context.getSecretKey();
                String OBS_ENDPOINT = context.getUserData("obs_endpoint");
                obsClient = new ObsClient(ACCESS_KEY, SECRET_ACCESS_KEY, OBS_ENDPOINT);
            }
        }
        public  void setObjectInfo(String objectKey, String inPath, String outPath) {
            this.objectKey = objectKey;
            localInPath = inPath;
            localOutPath = outPath;
        }
        public void downloadFile() {
            initObsclient();
            try {
                ObsObject obsObject = obsClient.getObject(inputBucketName, objectKey);
                InputStream inputStream = obsObject.getObjectContent();
                byte[] b = new byte[1024];
                int len;
                FileOutputStream fileOutputStream = new FileOutputStream("/tmp/" + objectKey);
                while ((len = inputStream.read(b)) != -1) {
                    fileOutputStream.write(b);
                }
                inputStream.close();
                fileOutputStream.close();
            } catch (ObsException ex) {
                ex.printStackTrace();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        public void uploadFile() {
            try {
                // Local path of the files to upload. File names must be specified.
                FileInputStream fis = new FileInputStream(new File("/tmp/" + objectKey + ".encrypt"));
                obsClient.putObject(outputBucketName, objectKey, fis);
                fis.close();
            } catch (FileNotFoundException e) {
                throw new RuntimeException(e);
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```


Creating a Function

When creating a function, specify an agency with OBS and DEW access permissions so that FunctionGraph can invoke these two services.

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.

After setting the basic information, click **Create**.

- For **Function Name**, enter **fss_examples_dew**.
- For **Agency**, select **serverless_trust**.
- For **Runtime**, select **Java 8**.

Step 4 On the details page of function **fss_examples_dew**, configure the following information:

1. On the **Code** tab, choose **Upload** > **Local JAR**, upload the compiled sample code JAR package, and click **OK**.
2. On the **Code** tab page, click **Deploy** and wait until the deployment is complete.
3. Choose **Configuration** > **Basic Settings**, set the following parameters, and click **Save**.
   - For **Memory**, select **128**.
   - For **Execution Timeout**, enter **3**.
   - For **Handler**, enter **com.huawei.kms.FileEncryptAndDecrypt.encrypt**.
   - For **App**, retain the default value **default**.
   - **Description**: Enter **File encryption and decryption**.
4. Choose **Configuration** > **Environment Variables**, set environment variables, and click **Save**.

   dew_endpoint: DEW endpoint

   dew_key_id: Master key ID

   input_bucket: OBS bucket for storing uploaded files

   output_bucket: OBS bucket for storing encrypted/decrypted files

   obs_endpoint: OBS endpoint

Table 11-1 Environment variables

Environment Variable	Description
dew_endpoint	DEW endpoint. To obtain the DEW endpoint, see **Regions and Endpoints**.
dew_key_id	User master key ID.
input_bucket	OBS bucket for storing input files.

Environment Variable	Description
output_bucket	OBS bucket for storing encrypted and uploaded files.
obs_endpoint	OBS endpoint. To obtain the OBS endpoint, see **Regions and Endpoints**.

----End

11.4 Adding an Event Source

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

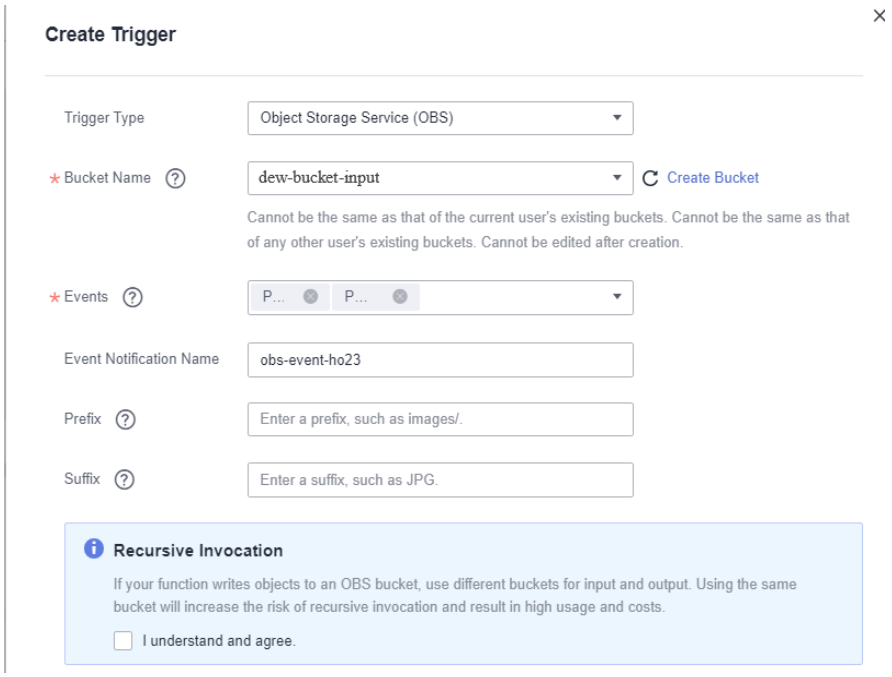
Step 1 On the **fss_examples_dew** page, choose **Configuration** > **Triggers** and click **Create Trigger**.

Step 2 Select **Object Storage Service (OBS)** for **Trigger Type**, and set the trigger information, as shown in **Figure 11-2**.

Select bucket **input_bucket**.

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

Figure 11-2 Creating an OBS trigger



Step 3 Click **OK**.

NOTE

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

----End

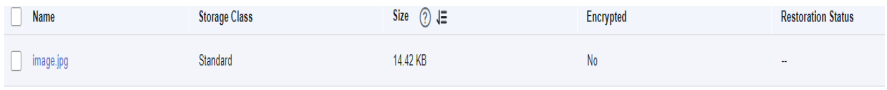
11.5 Processing Files

When a file is uploaded and updated to bucket **dew-bucket-input**, an event is generated to trigger the function. The function encrypts and decrypts the file and stores the processed one into bucket **dew-bucket-output**.

Uploading a File to Generate an Event

Log in to the **OBS console**, go to the object page of the **dew-bucket-input** bucket, and upload the **image.jpg** file, as shown in **Figure 11-3**.

Figure 11-3 Uploading a file

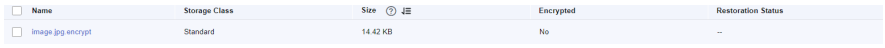


Triggering the Function

After the file is uploaded to bucket **dew-bucket-input**, OBS generates an event to trigger the file encryption/decryption function. The function encrypts/decrypts the file and stores the processed one into bucket **dew-bucket-output**. View the run logs of **fss_examples_dew** on the **Logs** tab page.

The **Objects** page of the bucket **dew-bucket-output** displays the processed file **image.jpg.encrypt**, as shown in **Figure 11-4**. In the **Operation** column, click **Download** to download the file.

Figure 11-4 Output file



12 Workflow + Function: Automatically Processing Data in OBS

12.1 Introduction

This best practice guides you through OBS data processing by using FunctionGraph. (The function flow feature is available in CN East-Shanghai1 and AP-Singapore.)

Scenarios

Use a function flow to automatically process data in OBS, such as video analysis, image transcoding, and video frame capturing.

- Upload images to a specified OBS bucket.
- Orchestrate functions to download images from OBS for transcoding and return the transcoded images to the client using a stream.

  NOTE

  The function you create must be in the same region (default region recommended) as the OBS bucket.

Procedure

- Create a bucket on the OBS console.
- Upload images to the bucket.
- Create a function.
- Create a function flow and orchestrate functions.
- Trigger the function to transcode images.

NOTE

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

1. One OBS bucket (for storing uploaded images)
2. One image processing function (**test-rotate**)
3. One function flow (**test-rotate-workflow**)

12.2 Preparation

Create an OBS bucket to store uploaded images.

Then create an agency to delegate FunctionGraph to access OBS resources.

Creating an OBS bucket

CAUTION

The bucket and function must be in the same region.

Procedure

Step 1 In the left navigation pane of the management console, choose **Storage** > **Object Storage Service** to go to the **OBS console**, and click **Create Bucket**.

On the **Create Bucket** page, set the bucket information.

- For **Region**, select a region.
- For **Bucket Name**, enter **your-bucket-input**.
- For **Data Redundancy Policy**, select **Single-AZ storage**.
- For **Default Storage Class**, select **Standard**.
- For **Bucket Policies**, select **Private**.
- For **Default Encryption**, select **Disable**.
- For **Direct Reading**, select **Disable**.

Retain the default values for other parameters and click **Create Now**.

View **your-bucket-input** in the bucket list.

----End

Creating an Agency

Step 1 In the left navigation pane of the management console, choose **Management & Governance** > **Identity and Access Management** to go to the IAM console. Then choose **Agencies** in the navigation pane.

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

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**.
- For **Description**, enter a description.

  Click **Next**. On the **Select Policy/Role** page, select **Tenant Administrator** and click **Next**.

  NOTE

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

Click **OK**.

----End

12.3 Building a Program

This section provides the sample code for image rotation.

Creating a Deployment Package

This example uses a Go function to rotate images. For details about function development, see the *FunctionGraph Developer Guide*. **Figure 12-1** shows the sample code directory. The service code is not described.

Figure 12-1 Sample code directory

```
func Test(b []byte, ctx context.RuntimeContext) (interface{}, error) {
    ak := ctx.GetAccessKey()
    sk := ctx.GetSecretKey()
    obsAddress := os.Getenv( key: "obsAddress")
    bucket := os.Getenv( key: "bucket")
    object := os.Getenv( key: "object")
    client, err := obs.New(ak, sk, obsAddress)
    if err != nil {
        log.Printf( format: "err:%v", err)
        return nil, err
    }
    output, err := client.GetObject(&obs.GetObjectInput{
        GetObjectMetadataInput: obs.GetObjectMetadataInput{
            Bucket: bucket,
            Key:    object,
        },
    })
    if err != nil {
        log.Printf( format: "err:%v", err)
        return nil, err
    }
    defer output.Body.Close()
    img, err := jpeg.Decode(output.Body)
    if err != nil {
        log.Printf( format: "err:%v", err)
        os.Exit( code: -1)
    }
    res := rotate180(img)
    buffer := bytes.NewBuffer( buf: nil)
    err = jpeg.Encode(buffer, res, &jpeg.Options{ Quality: 100})
    if err != nil {
        fmt.Println(err)
        return nil, err
    }
    err = ctx.Write(bytes.NewReader(buffer.Bytes()))
    return struct {}{}, err
}

func rotate180(m image.Image) image.Image {
    rotate180 := image.NewRGBA(image.Rect( x0: 0,  y0: 0, m.Bounds().Dx(), m.Bounds().Dy()))
    for x := m.Bounds().Min.X; x < m.Bounds().Max.X; x++ {
        for y := m.Bounds().Min.Y; y < m.Bounds().Max.Y; y++ {
            rotate180.Set(m.Bounds().Max.X-x, m.Bounds().Max.Y-y, m.At(x, y))
        }
    }
    return rotate180
}
```

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.

Click **Create Function**.

Set the function information.

After setting the basic information, click **Create**.

- For **Function Name**, enter **test-rotate**.
- For **Agency**, select **serverless_trust**.
- For **Runtime**, select **Go 1.x**.

On the details page of function **test-rotate**, configure the following information:

a. On the **Code** tab, choose **Upload** > **Local ZIP**, upload the binary file **go-test.zip** of the sample code, and click **OK**.

b. Choose **Configuration** > **Basic Settings**, set the following parameters, and click **Save**.

- For **Memory**, select **256**.
- For **Execution Timeout**, enter **40**.
- For **Handler**, retain the default value **index.handler**.
- For **App**, retain the default value **default**.
- For **Description**, enter **Image rotation**.

c. Choose **Configuration** > **Environment Variables**, set environment variables, and click **Save**.

bucket: the bucket parameter defined in **handler.go** for pulling images. Set the value to **your-bucket-output**, the bucket created for storing images.

object: the image name parameter defined in **handler.go**. Set the value to **your-picture-name**.

obsAddress: the bucket address parameter defined in **handler.go** for pulling images. Set the value to **obs.region.myhuaweicloud.com**.

----End

Table 12-1 Environment variable description

Environment Variable	Description
bucket	OBS bucket parameters defined in the **handler.go** file for pulling images.
object	The image name parameter defined in **handler.go**.
obsAddress	The bucket address parameter defined in **handler.go** for pulling images. The value of **obsAddress** is in the format of **obs.{region}.myhuaweicloud.com**. For details about the value of **region**, see **Regions and Endpoints**.

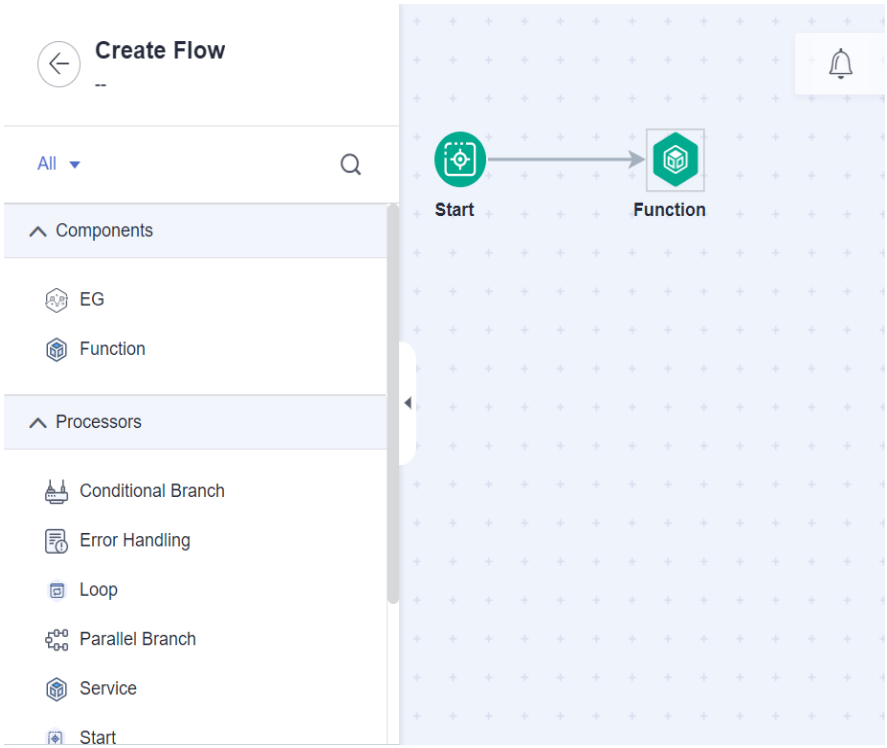
---- End

Creating a Flow

Step 1 Return to the FunctionGraph console. Then choose **Flows** in the navigation pane.

Click **Create** next to **Express Flow**.

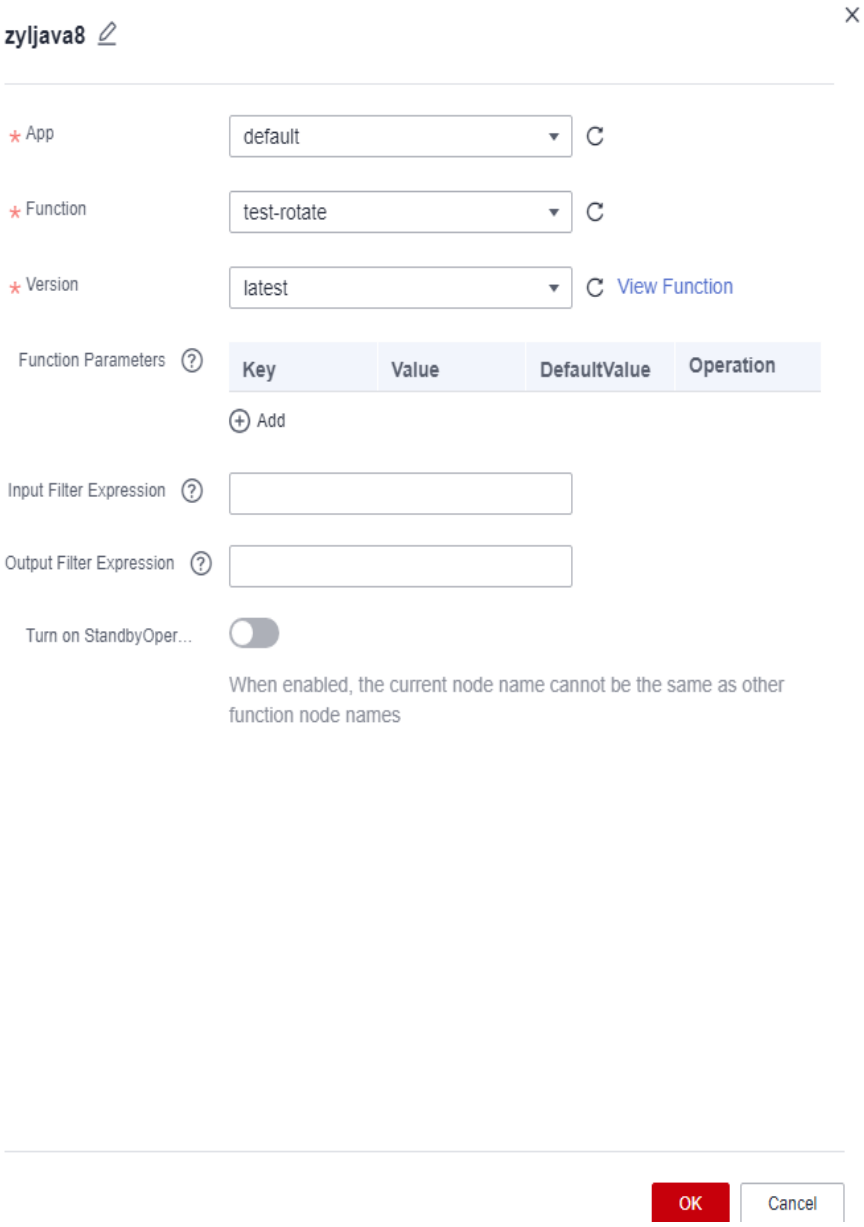
Figure 12-2 Creating an express flow



Step 2 Drag a function node and click it to configure parameters.

- **App**: Retain the default value **default**.
- **Function**: Select the **test-rotate** function created in the previous step.
- **Version**: Retain the default value **latest**.
- Retain the default values for other parameters.

Figure 12-3 Configuring function node



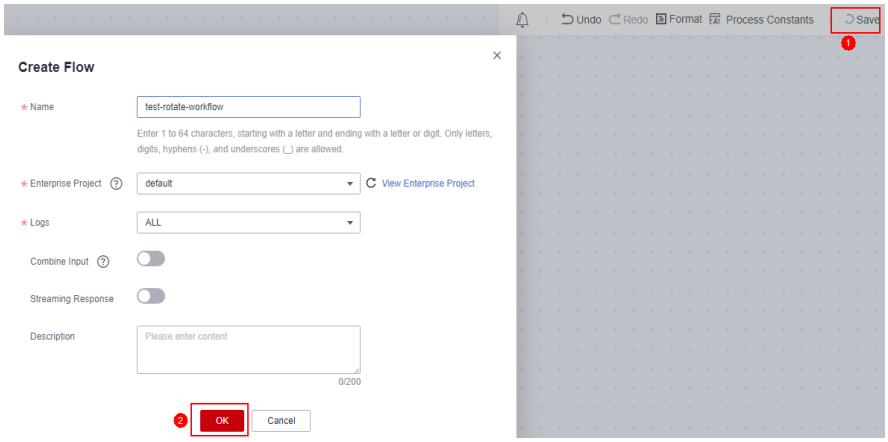
Click **OK** after the parameters are configured.

Step 3 After the function flow node is created, click **Save** in the upper right corner, configure the following basic information, and click **OK**.

- **Name**: **test-rotate-workflow**.
- **Enterprise Project**: Retain the default value **default**.
- **Logs**: Retain the default value **ALL**.

Retain the default values for other parameters.

Figure 12-4 Saving a flow



----End

12.4 Processing Images

Upload an image to bucket **your-bucket-input**, and use a tool to simulate a client and trigger the function flow. The image is rotated by 180°, and then returned to the client as stream data.

Uploading an Image

Log in to the **OBS console**, go to the object page of the **your-bucket-input** bucket, and upload the **image.jpeg** image, as shown in **Figure 12-5**. **Figure 12-6** shows the uploaded image.

Figure 12-5 Example

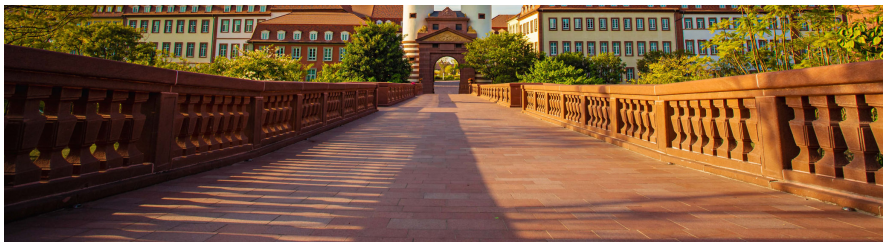
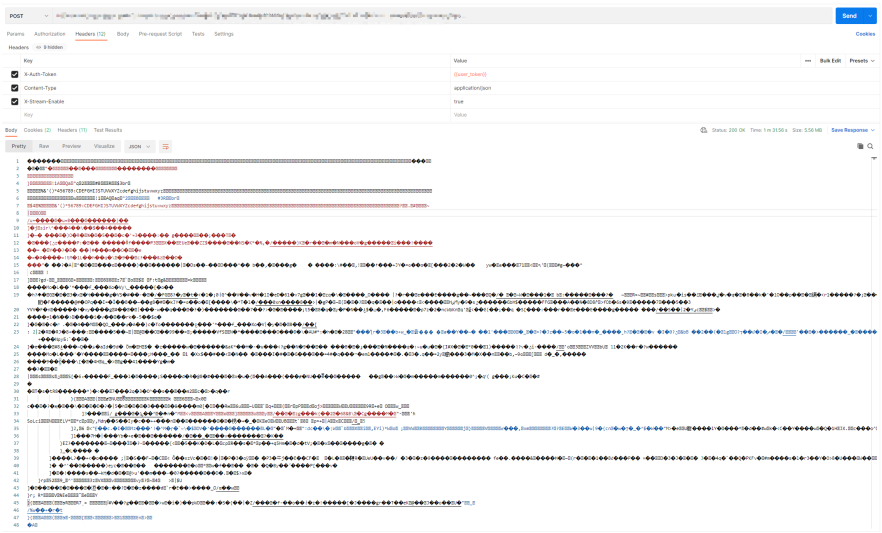


Figure 12-6 Uploaded image

Name	Storage Class	Size	Encrypted
image.jpeg	Standard	1.24 MB	No

Using Postman to Trigger the Function Flow



The following figure shows the image saved from the byte stream.

13 Filtering Logs in Real Time by Using FunctionGraph and LTS

13.1 Introduction

This chapter elaborates the following aspects of the practice:

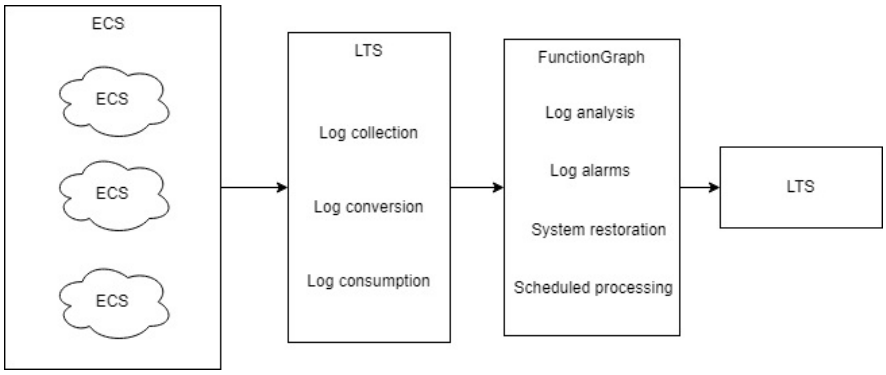
- Scenario and benefits
- Preparation
- Building a program
- Event source
- Processing results
- Extended applications

Scenario

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

Obtain log data using an LTS trigger created on FunctionGraph, analyze and process key information in the logs by using a customized function, and then transfer the filtered logs to another log stream. **Figure 13-1** shows this process.

Figure 13-1 Processing workflow



Benefits

- Quickly collect and convert logs with LTS.
- Process and analyze data by using the event triggering and auto scaling features of serverless function computing. No O&M is involved, and resources are pay-per-use.
- Transfer filtered logs to another log stream. The original log stream is automatically deleted at the expiration time you set, reducing log storage costs.

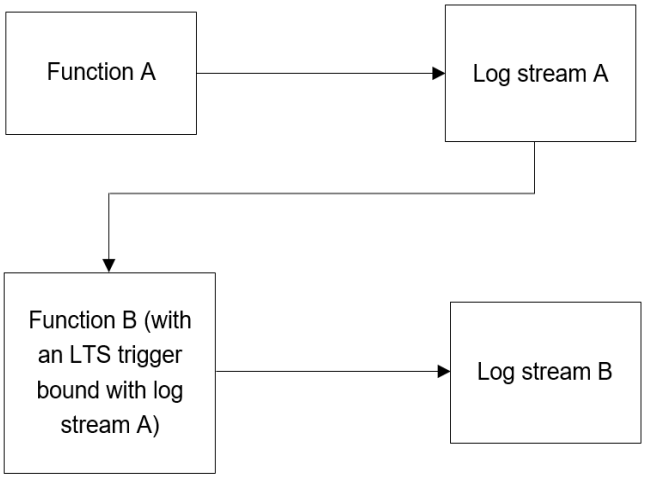
13.2 Preparation

Download **lts_cleanse.zip** (including code file **write_log.py** of function A, code file **lts_cleanse.py** of function B, and dependency **huaweicloudsdklts**) and **lts_cleanse.zip.sha256** to filter logs in real time.

Collecting and Storing Logs

- Create two log groups, for example, **test1206** and **test-1121**, on the LTS console. For details, see **Creating a Log Group**.
- Create two log streams, for example, **test-206** and **test-1121**, on the LTS console. For details, see **Creating a Log Stream**.
- Create function **A** to write logs to **test-206**. For the sample code of this function, see the **write_log.py** file.
- Create function **B** with an LTS trigger to receive logs from **test-206**, process the logs, and write the result to **test-1121**. For the sample code of this function, see the **lts_cleanse.py** file.
- Configure an agent to collect logs from servers, such as ECSs, to a specified log group. For details, see **Installing the ICAgent**.

Figure 13-2 Flowchart

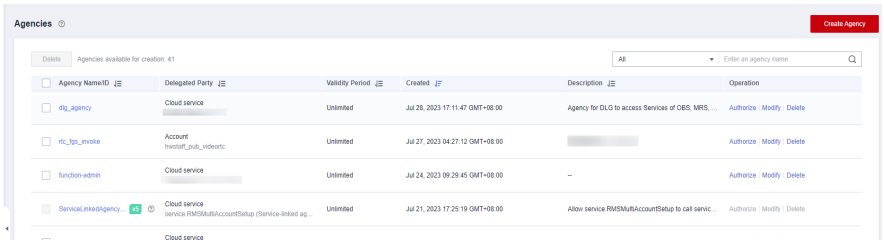


Creating an Agency

Step 1 Log in to the IAM console.

Step 2 Choose **Agencies** from the navigation pane, and click **Create Agency** in the upper right corner, as shown in **Figure 13-3**.

Figure 13-3 Creating an agency

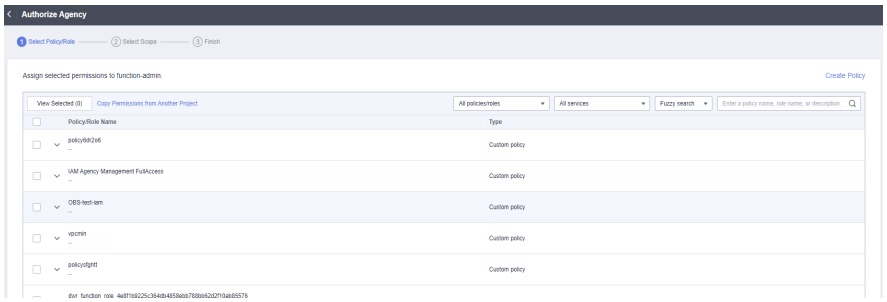


Step 3 Configure the agency.

- **Agency Name**: Enter **LtsOperation**.
- **Agency Type**: Select **Cloud service**.
- **Cloud Service**: Select **FunctionGraph**.
- **Validity Period**: Select **Unlimited**.
- **Description**: Describe the agency.

Step 4 Click **Next**. On the displayed page, search for **LTS Administrator** and **Tenant Administrator** in the search box on the right and select them, as shown in **Figure 13-4**.

Figure 13-4 Selecting a permission



NOTE

LTS Administrator depends on **Tenant Guest**. When you select the former, the latter will also be selected.

Step 5 Click **Next** and select the application scope of the permission based on service requirements.

----End

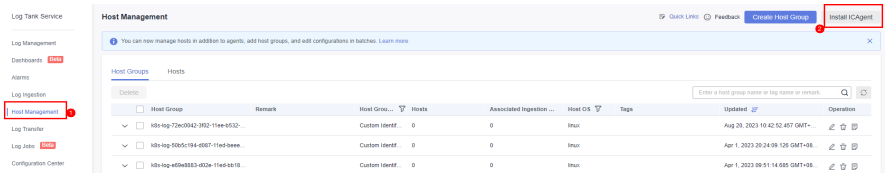
13.3 Building a Program

Prerequisites

(1) The IP address in the two functions is an access point of LTS. To obtain this IP address, perform the following steps:

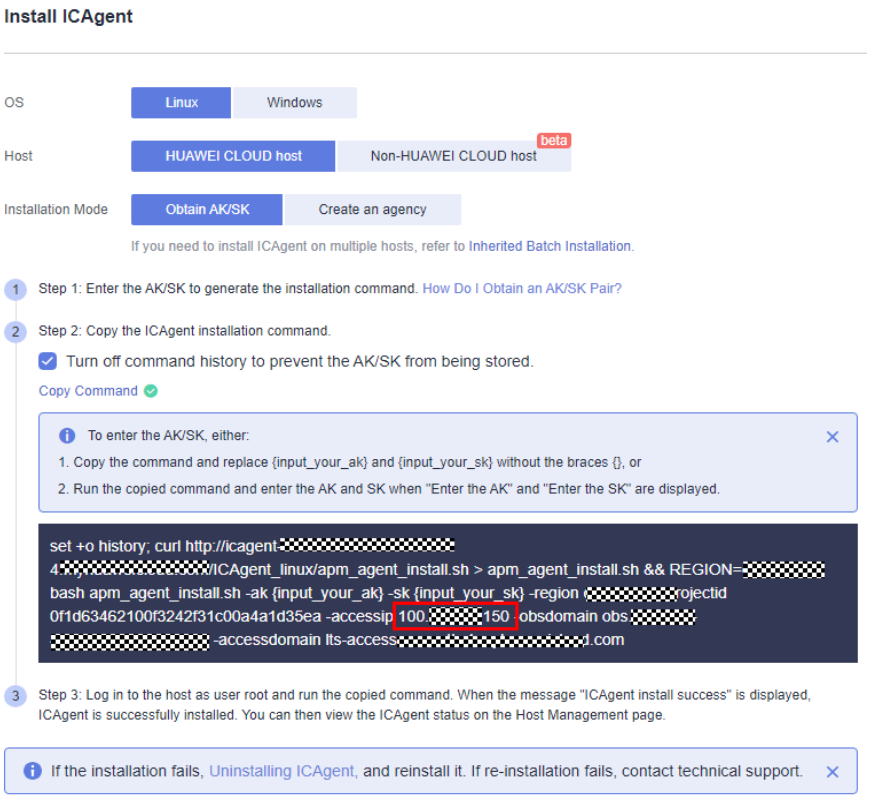
1. Log in to the LTS console. In the navigation pane, choose **Host Management**.
2. Click **Install ICAgent** in the upper right.

Figure 13-5 Installing an ICAgent



3. Obtain the access point IP address in the **Install ICAgent** window.

Figure 13-6 Access point IP address



2. Obtain the values of **log_group_id** and **log_stream_id** in the functions. For details, see **Obtaining the Account ID, Project ID, Log Group ID, and Log Stream ID**.

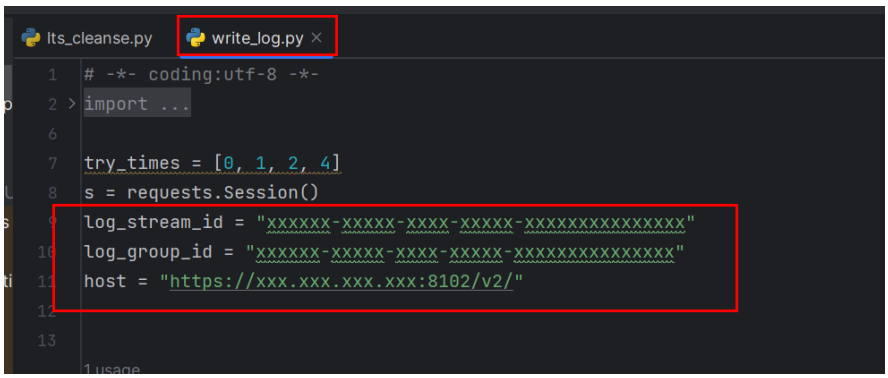
3. Create the LTS dependency required by function **B**. For details, see **How Do I Create a Dependency on the FunctionGraph Console?** and **How Do I Add a Dependency to a Function?** You can run the **pip install huaweicloudsdklts** command to create the dependency. The sample code contains the **huaweicloudsdklts** dependency for Python 3.9.

Creating a Function

Create a log extraction function by uploading the sample code package. Select the Python 3.9 runtime and the agency **LtsOperation** created in **Creating an Agency**. For details about how to create a function, see **Creating an Event Function**.

Create function **A**. For the sample code of this function, see the **write_log.py** file. In the code of function **A**, replace **host**, **log_group_id**, and **log_stream_id** with the access point and the IDs of log group **test-1206** and log stream **test-206**, as shown in **Figure 13-7**.

Figure 13-7 write_log.py



Create function **B**. For the sample code of this function, see the **lts_cleanse.py** file. In the code of function **B**, replace **host**, **log_group_id**, and **log_stream_id** with the access point and the IDs of log group **test-1121** and log stream **test-1121**, and add the **huaweicloudsdklts** dependency to this function, as shown in **Figure 13-8** and **Figure 13-9**.

Figure 13-8 lts_cleanse.py

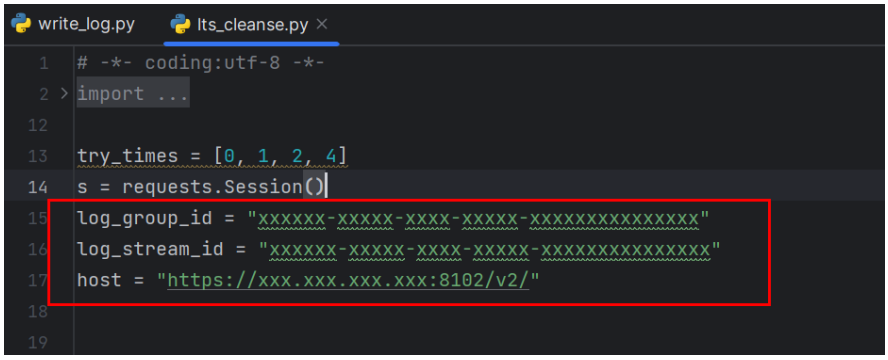
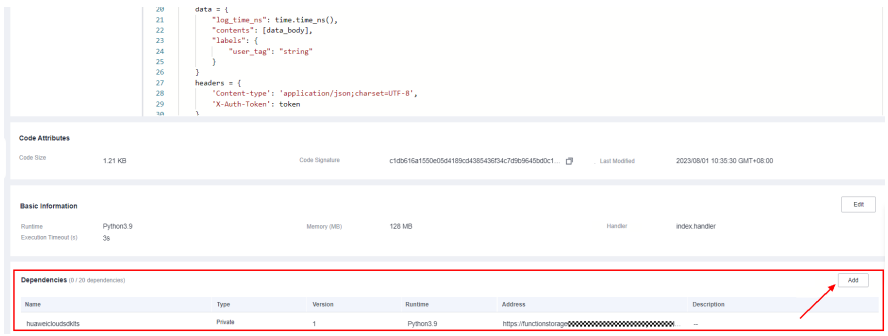


Figure 13-9 Adding a dependency for function B



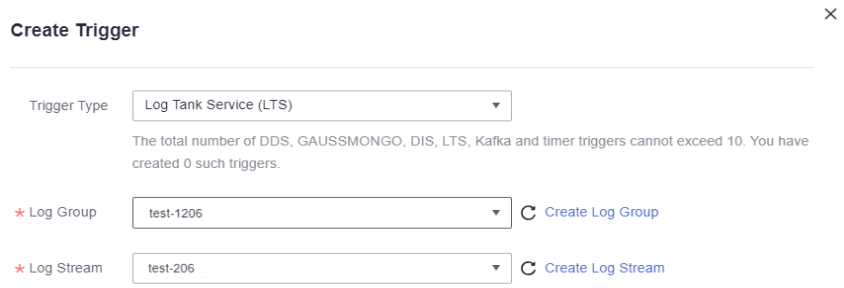
This function performs Base64 decoding on received log event data, extracts alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR**, and then stores the

extracted logs to a specified LTS log stream. Set log extraction conditions based on the content of your service logs.

13.4 Adding an Event Source

Create an LTS trigger by using the log group and log stream created in **Preparation**, and configure the trigger information according to **Figure 13-10**.

Figure 13-10 Creating an LTS trigger



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

13.5 Processing Results

Filter alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR**, and transfer them to a specified log stream. **Figure 13-11** and **Figure 13-12** show the real-time logs before and after filtering, respectively.

Figure 13-11 Logs before filtering

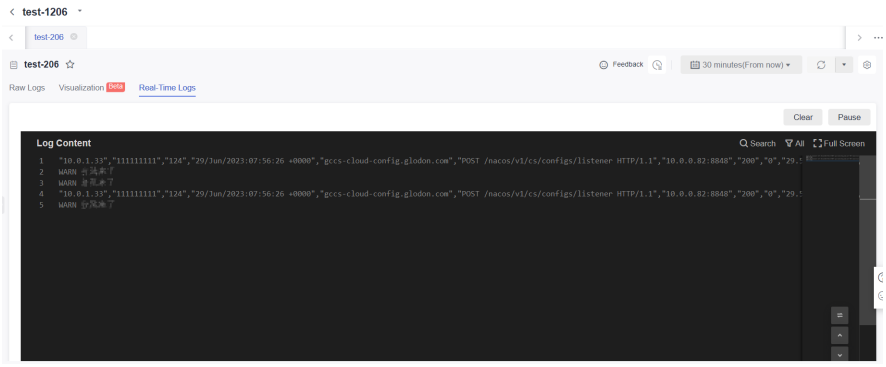
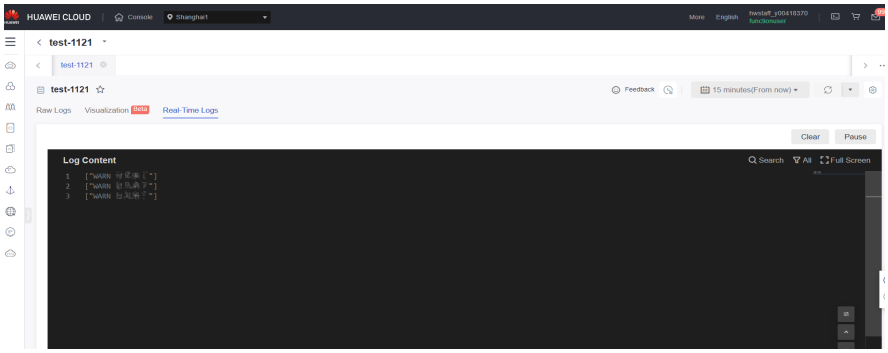


Figure 13-12 Logs after filtering



Check the function invocation by viewing the metrics, as shown in the following figures.

Figure 13-13 Function metrics (1)

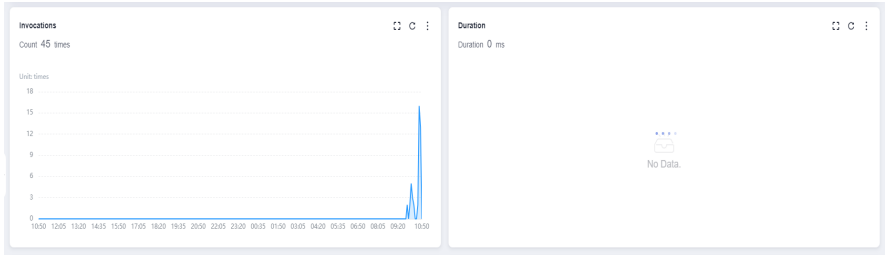


Figure 13-14 Function metrics (2)

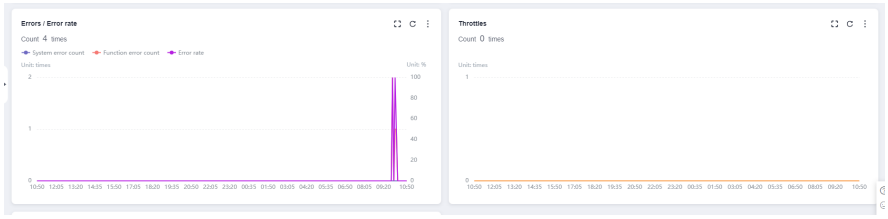
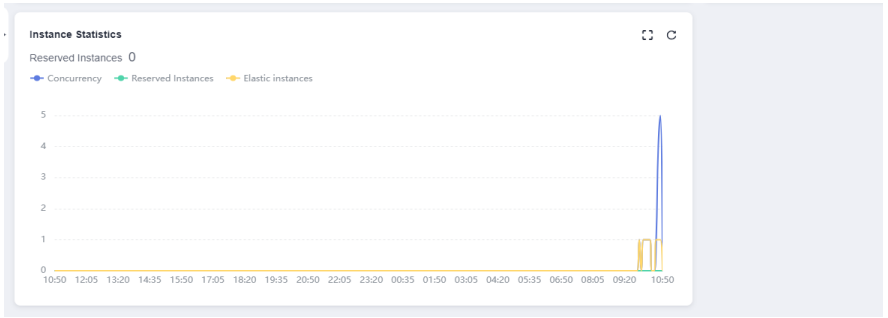


Figure 13-15 Function metrics (3)



13.6 Extended Applications

In addition to log processing and transfer to LTS, the combination of FunctionGraph and LTS can apply to more scenarios, for example, analyzing and

processing log data with a timer trigger to delete redundant logs and save space and costs.

14 Gracefully Shutting Down ECSs Using FunctionGraph and AS

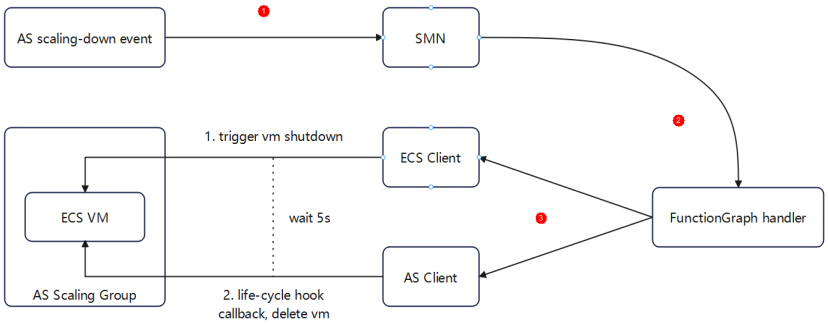
14.1 Introduction

In Auto Scaling (AS), instance scale-in means that instances in an AS group are first shut down, and then removed and deleted after you clear the data in these instances. This is also called "graceful shutdown".

Scenario

1. Configure a message notification to forward AS scale-in events to SMN.
2. Use FunctionGraph to receive scale-in messages forwarded by SMN and obtain instance information through a customized function.
3. Call the ECS service to shut down scaled-in instances. Five seconds (or a custom period) after the shutdown, call the AS service to delete the instances.

Figure 14-1 Graceful shutdown in AS



14.2 Preparation

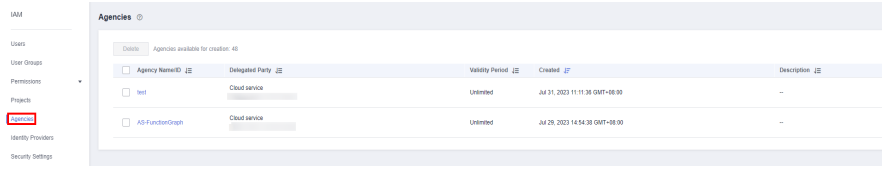
1. Obtain the program package of graceful shutdown with AS.

2. Create an agency named **ASOperation**, and grant it the **ECS FullAccess** and **AutoScaling FullAccess** permissions. For details, see **Configuring Agency Permissions**.

Creating an Agency

1. Log in to the **IAM console**.
2. Choose **Agencies** from the navigation pane, and click **Create Agency** in the upper right corner.

   Figure 14-2 Creating an agency

3. Configure the agency.
   - **Agency Name**: Enter **ASOperation**.
   - **Agency Type**: Select **Cloud service**.
   - **Cloud Service**: Select **FunctionGraph**.
   - **Validity Period**: Select **Unlimited**.
   - **Description**: Describe the agency.
4. Click **Next**. On the displayed page, search for **ECS FullAccess** and **AutoScaling FullAccess** in the search box on the right and select them.
5. Click **Next** and select the application scope of the permissions based on service requirements.
6. Click **OK**.

14.3 Building a Program

Download the graceful shutdown program package **as_graceful_shutdown.zip** and create a function from scratch.

Creating a Function

1. Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane.
2. Click **Create Function**.
3. Set the function information. After setting the basic information, click **Create Function**.
   - **Function Name**: Enter **as_graceful_shutdown**.
   - **Agency**: Select **ASOperation** created in **Creating an Agency**.
   - **Runtime**: Select **Python 3.6**.
4. On the **Code** tab of the **as_graceful_shutdown** function details page, choose **Upload** > **Local ZIP**, upload the sample code **as_graceful_shutdown.zip**, and click **Deploy**. Choose **Configuration** > **Basic Settings**, set the following parameters, and click **Save**.

- **Memory**: Select **128**.
- **Execution Timeout**: Enter **10**.
- **Handler**: Retain the default value **index.handler**.
- **App**: Retain the default value **default**.
- **Description**: Enter **Graceful shutdown with AS**.

Setting Environment Variables

On the details page of function **as_graceful_shutdown**, choose **Configuration** > **Environment Variables**, set environment variables according to **Table 14-1**, and click **Save**.

Table 14-1 Environment variable description

Environment Variable	Description
region	Region of your AS group.
projectId	Project ID of your AS group.
ak	Access key ID, which is a unique identifier used in conjunction with a secret access key to sign requests cryptographically. For details about how to obtain an AK, see **Obtaining an AK/SK**.
sk	Secret access key used together with an AK to sign requests cryptographically. It identifies a request sender and prevents the request from being modified. For details about how to obtain an SK, see **Obtaining an AK/SK**.

Adding Dependencies

1. Create dependencies **huaweicloudsdk_ecs_core_py3.6** and **huaweicloudsdk_as_core_py3.6**. For more information, see **Creating a Dependency**.

   NOTE

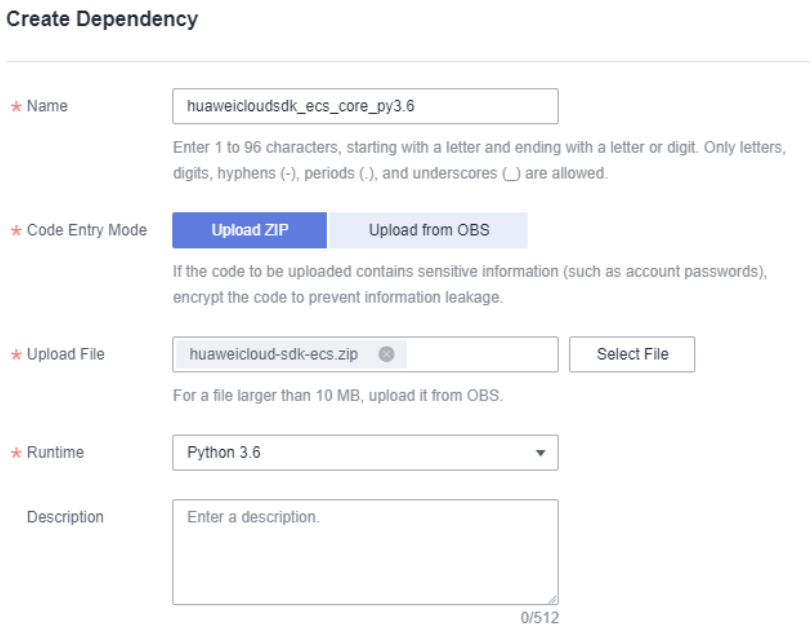
   You can also download the dependencies provided by FunctionGraph if needed:

   - **huaweicloudsdk_ecs_core_py3.6**
   - huaweicloudsdk_as_core_py3.6 (coming soon)

2. Return to the FunctionGraph console, and choose **Functions** > **Dependencies** in the navigation pane. Click **Create Dependency**, and create dependencies **huaweicloudsdk_ecs_core_py3.6** and **huaweicloudsdk_as_core_py3.6** successively.
3. Set the dependency information and click **OK**.

- **Name**: Enter **huaweicloudsdk_ecs_core_py3.6** or **huaweicloudsdk_as_core_py3.6**.
- **Code Entry Mode**: Select **Upload ZIP**.
- **Upload File**: Select the target ZIP file.
- **Runtime**: Select **Python 3.6**.

Figure 14-3 Creating a dependency

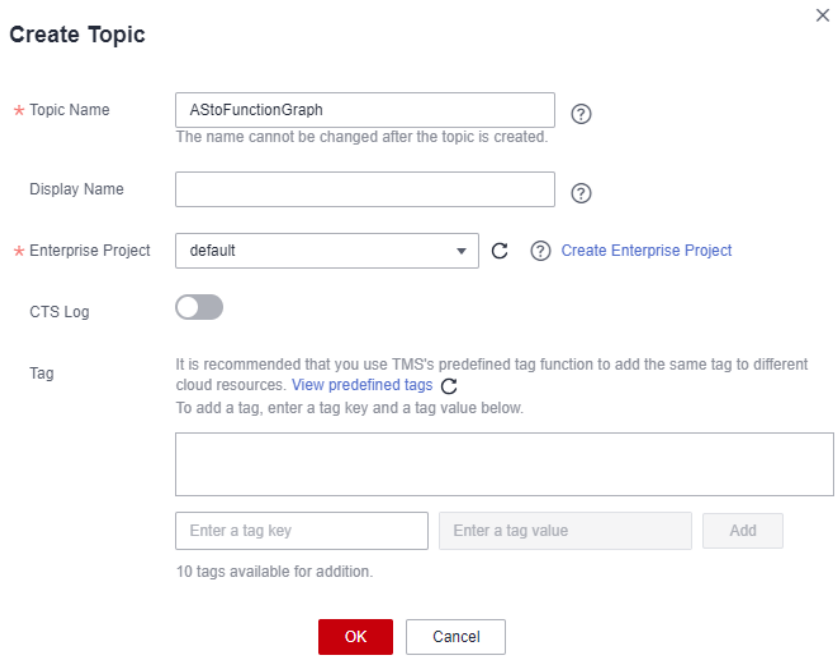


4. Go to the details page of function **as_graceful_shutdown**, click the **Code** tab, and click **Add** in the **Dependencies** area at the bottom.
5. Add dependencies **huaweicloudsdk_ecs_core_py3.6** and **huaweicloudsdk_as_core_py3.6**.

14.4 Configuring a Message Notification

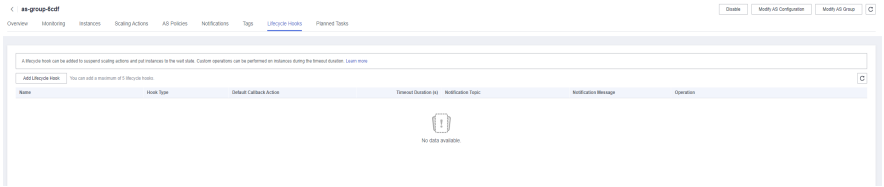
1. Log in to the SMN console, choose **Topic Management** > **Topics**, and click **Create Topic**.
2. Set the topic information, and click **OK**.
   - **Topic Name**: Enter **AStoFunctionGraph**.
   - **Enterprise Project**: Select **default** or another one.

Figure 14-4 Creating a topic



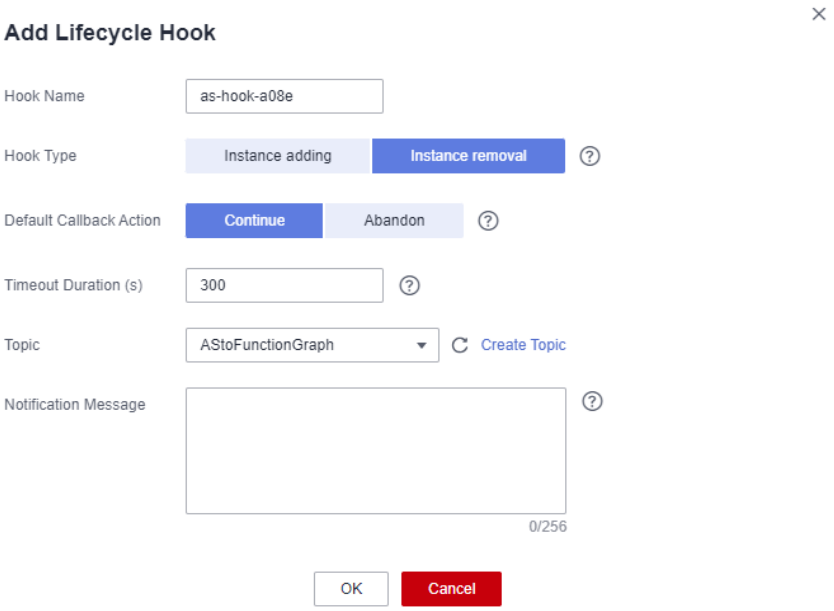
3. Log in to the AS console, go to the details page of the target AS group, click the **Lifecycle Hooks** tab, and then click **Add Lifecycle Hook**.

Figure 14-5 Lifecycle hooks



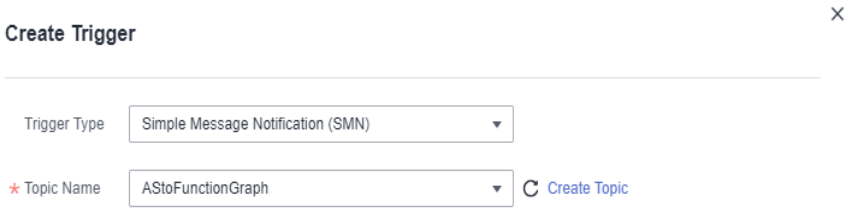
4. Configure the lifecycle hook information, and click **OK**. This hook is used to suspend the scaled-in instances and send a message to the specified SMN topic.
   - **Hook Type**: Select **Instance removal**.
   - **Default Callback Action**: Select **Continue**.
   - **Timeout Duration**: Enter **300**.
   - **Notification Topic**: Select **AStoFunctionGraph**.

Figure 14-6 Creating a lifecycle hook



5. Go to the FunctionGraph console. On the details page of function **as_graceful_shutdown**, choose **Configuration** > **Triggers**, and click **Create Trigger**.
6. Select SMN for the trigger type and **AStoFunctionGraph** for the topic. Then the messages received by this topic will trigger the handler function.

Figure 14-7 Creating a trigger



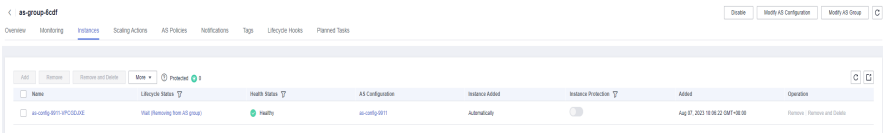
14.5 Processing Results

Triggering an AS Action

On the AS group details page, change the number of expected instances to a value less than the number of existing instances to trigger the scale-in action.

If an instance is moved out of the AS group, it is successfully suspended.

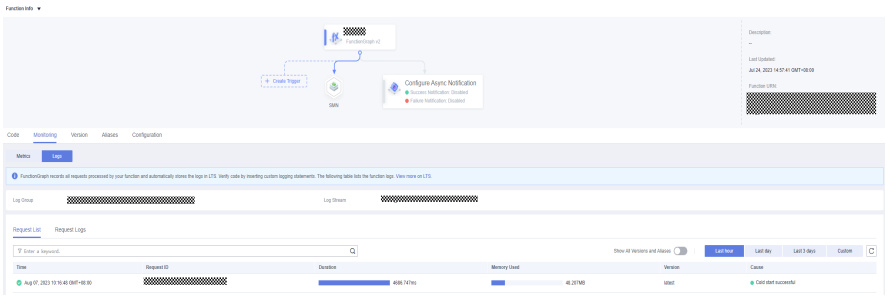
Figure 14-8 Instance status



Triggering the Function

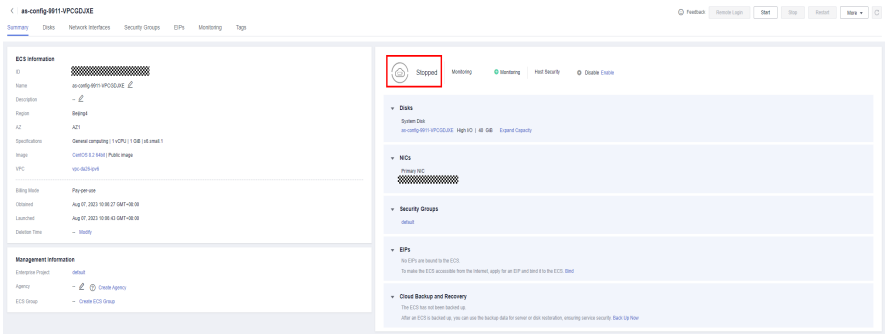
1. On the function details page, choose **Monitoring** > **Logs**, and check the notification and function execution result.

   Figure 14-9 Function logs

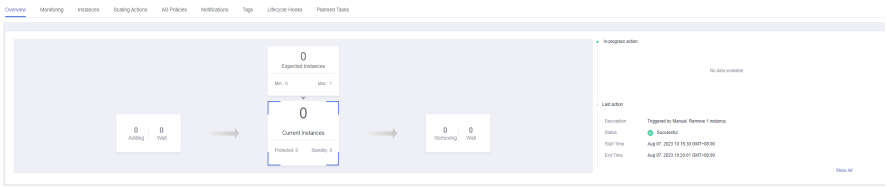
2. On the AS group details page, select the preceding instance, and check that it is shut down.

   Figure 14-10 Instance status

3. Wait until the instance is automatically moved out and deleted.

   Figure 14-11 Instance moved out and deleted

15 Building an HTTP Function with Go

Introduction

This chapter describes how to deploy services on FunctionGraph using Go.

HTTP functions do not support direct code deployment using Go. This section uses binary conversion as an example to describe how to deploy Go programs on FuntionGraph.

Procedure

Building a code package

Create the source file **main.go**. The code is as follows:

```
// main.go
package main

import (
    "fmt"
    "net/http"

    "github.com/emicklei/go-restful"
)

func registerServer() {
    fmt.Println("Running a Go Http server at localhost:8000/")

    ws := new(restful.WebService)
    ws.Path("/")

    ws.Route(ws.GET("/hello").To(Hello))
    c := restful.DefaultContainer
    c.Add(ws)
    fmt.Println(http.ListenAndServe(":8000", c))
}

func Hello(req *restful.Request, resp *restful.Response) {
    resp.Write([]byte("nice to meet you"))
}

func main() {
    registerServer()
}
# bootstrap
/opt/function/code/go-http-demo
```


In **main.go**, an HTTP server is started using port **8000**, and an API whose path is **/hello** is registered. When the API is invoked, "nice to meet you" is returned.

Compiling and packaging

1. On the Linux server, compile the preceding code using the **go build -o go-http-demo main.go** command. Then, compress **go-http-demo** and **bootstrap** into a ZIP package named **xxx.zip**.
2. To use the Golang compiler to complete packaging on a **Windows** host, perform the following steps:

```
# Switch the compilation environment
# Check the previous Golang compilation environment
go env
# Set the following parameters to the corresponding value of Linux
set GOARCH=amd64
go env -w GOARCH=amd64
set GOOS=linux
go env -w GOOS=linux

# go build -o [target executable program] [source program]
# Example
go build -o go-http-demo main.go

# Restore the compilation environment
set GOARCH=amd64
go env -w GOARCH=amd64
set GOOS=windows
go env -w GOOS=windows
```

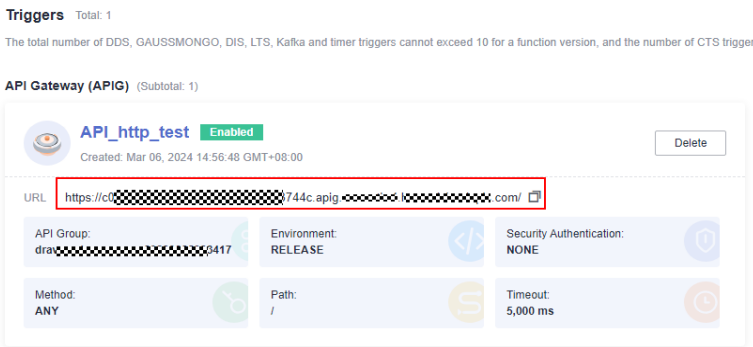
Creating an HTTP function and uploading code

Create an HTTP function and upload the **xxx.zip** package. For details, see **Creating an HTTP Function**.

Creating an APIG trigger

Create an APIG trigger by referring to **Using an APIG Trigger**. Set the authentication mode to **None** for debugging.

Figure 15-1 APIG trigger



Invocation test

Copy the URL of the APIG trigger and the **/hello** path registered in the code to the address box of the browser. The following information is displayed.

Figure 15-2 Request result