FunctionGraph

Best Practices

Issue	01
Date	2024-12-05

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Contents

1 FunctionGraph Best Practices

This document summarizes practices in common application scenarios of FunctionGraph. Each practice case is given detailed solution description and operation guidance, helping you easily build your services based on FunctionGraph.

Table 1-1 FunctionGraph best practices

Practice	Description
Processing DIS Data	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).
Integrating with LTS to Analyze Logs in Real Time	Analyze and process key log data of servers such as ECS collected and converted by Log Tank Service (LTS), filter out alarm logs, store processed log data in a specified OBS bucket, and use SMN to push alarm messages to service personnel by SMS message or email.
Integrating with CTS to Analyze Login/Logout Security	Obtain operation records of subscribed cloud resources from Cloud Trace Service (CTS), analyze and process the records, report alarms, and use SMN to push alarm messages to service personnel by SMS message or email.
Periodically Starting or Stopping Huawei Cloud ECSs	Use FunctionGraph to call the corresponding ECS APIs to start or stop ECSs at specified time.

Practice	Description
Building an HTTP Function with Spring Boot	Deploy services on FunctionGraph using Spring Boot.
Creating a FunctionGraph Backend API That Uses a Custom Authorizer	APIG supports IAM, app, and custom authentication. Create a FunctionGraph API that uses a custom authorizer.
Uploading Files with FunctionGraph and APIG	Uses Node.js and Python as examples to describe how to develop a backend parsing function for obtaining uploaded files, such as run logs and web application images, from devices to cloud servers based on FunctionGraph and APIG.
Processing IoT Data	Use FunctionGraph functions to process data reported by IoT devices and device status changes through IoTDA triggers.
Workflow + Function: Automatically Processing Data in OBS	Use a function flow to automatically process data in OBS, such as video analysis, image transcoding, and video frame capturing (available only in CN East-Shanghai1 and AP-Singapore).
Filtering Logs in Real Time by Using FunctionGraph and 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.
Building an HTTP Function with Go	Deploy services on FunctionGraph using Go.
Using FunctionGraph HTTP Functions to Process gRPC Requests	Process gRPC requests in FunctionGraph (only supported in LA-Santiago).
Cold Start Optimization Practices	Optimize FunctionGraph cold start.

2 Processing DIS Data

2.1 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.

2.2 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.

In the **Basic Information** area, enter a name, for example, **vpc-cloudtable**. Use the default values for other parameters.

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

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

----End

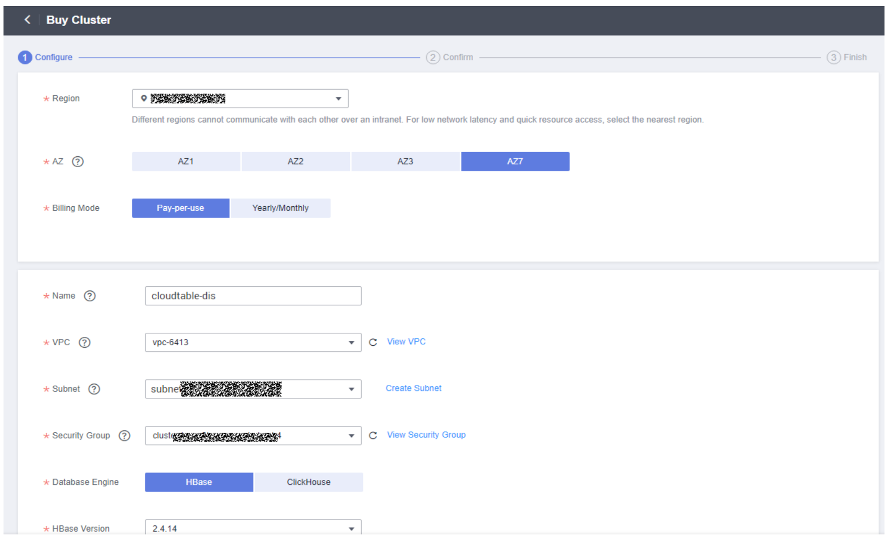
Creating a Cluster

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.

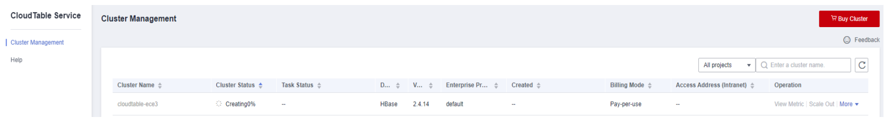
- **Region**: Use the default region.
- **Name**: Enter a name, for example, **cloudtable-dis**.
- **VPC**: Select **vpc-cloudtable** created in **Creating a VPC**.
- Retain the default values for other parameters.

Figure 2-1 Buying a cluster


Buy Cluster
1 Configure — 2 Confirm — 3 Finish
* Region: [redacted]
Different regions cannot communicate with each other over an intranet. For low network latency and quick resource access, select the nearest region.
* AZ: AZ1 | AZ2 | AZ3 | AZ7 (selected)
* Billing Mode: Pay-per-use (selected) | Yearly/Monthly
* Name: cloudtable-dis
* VPC: vpc-6413 — View VPC
* Subnet: subne[redacted] — Create Subnet
* Security Group: cluste[redacted] — View Security Group
* Database Engine: HBase (selected) | ClickHouse
* HBase Version: 2.4.14

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

Figure 2-2 Creating a cluster


CloudTable Service
Cluster Management
Help

Cluster Management — Buy Cluster — Feedback
All projects — Enter a cluster name.

Cluster Name	Cluster Status	Task Status	D...	V...	Enterprise Pr...	Created	Billing Mode	Access Address (Intranet)	Operation
cloudtable-ece3	Creating0%	--	HBase	2.4.14	default	--	Pay-per-use	--	View Metric | Scale Out | More

NOTE

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

----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 an agency name, for example, **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 **DIS Administrator** and **Cloudtable Administrator**.

NOTE

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

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

----End

2.3 Building a Program

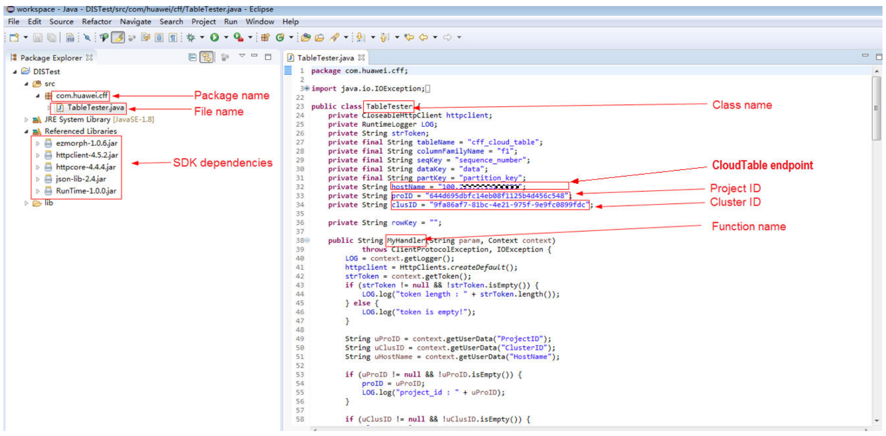
Download the source code and program package (including function dependencies) to create a function from scratch for converting DIS stream data formats.

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 2-3**.

Figure 2-3 Sample code



```
package com.huawei.cff;

import java.io.IOException;

public class TableTester {
    private CloseableHttpClient httpclient;
    private RuntimeLogger LOG;
    private String strToken;
    private final String tableName = "cff_cloud_table";
    private final String columnFamilyName = "f1";
    private final String seqKey = "sequence_number";
    private final String dataKey = "data";
    private final String partKey = "partition_key";
    private String hostName = "100.[redacted]";
    private String proID = "644d695dbfc14eb08f1125b4d456c548";
    private String clusID = "9fa86af7-81bc-4e21-975f-9e9fc0899fdc";

    private String rowKey = "";

    public String MyHandler(String param, Context context)
            throws ClientProtocolException, IOException {
        LOG = context.getLogger();
        httpclient = HttpClients.createDefault();
        strToken = context.getToken();
        if (strToken != null && !strToken.isEmpty()) {
            LOG.log("token length : " + strToken.length());
        } else {
            LOG.log("token is empty!");
        }

        String uProID = context.getUserData("ProjectID");
        String uClusID = context.getUserData("ClusterID");
        String uHostName = context.getUserData("HostName");

        if (uProID != null && !uProID.isEmpty()) {
            proID = uProID;
            LOG.log("project_id : " + uProID);
        }

        if (uClusID != null && !uClusID.isEmpty()) {
```

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 2-4**.
2. Obtain the project ID in the project list, as shown in **Figure 2-5**.

Figure 2-4 My Credentials

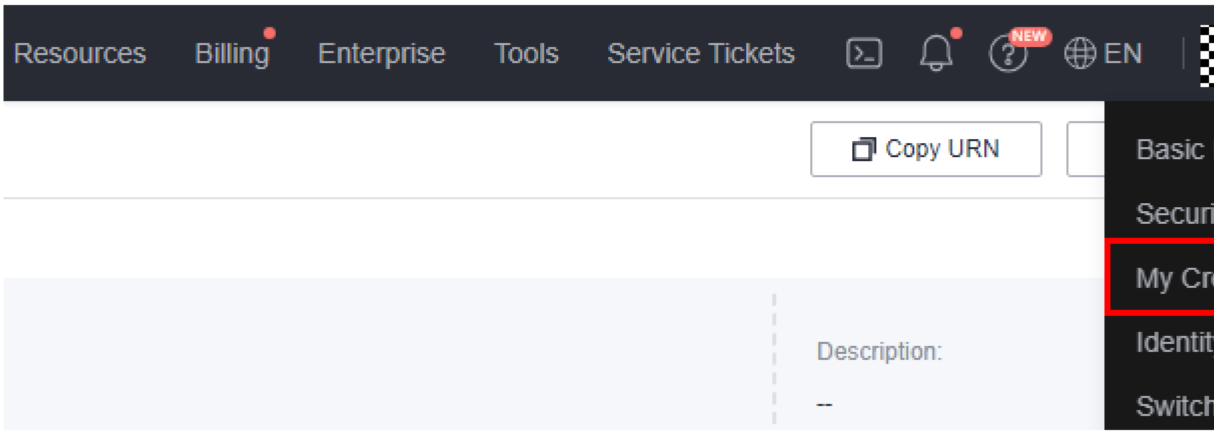
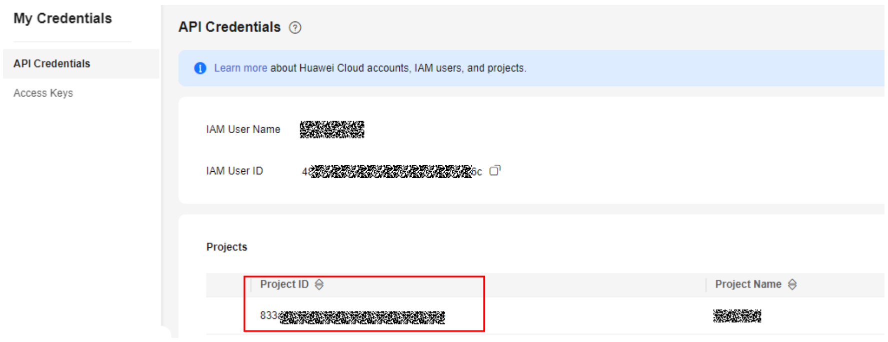


Figure 2-5 Project ID


My Credentials

API Credentials

Access Keys

API Credentials

Learn more about Huawei Cloud accounts, IAM users, and projects.

IAM User Name [redacted]

IAM User ID 4[redacted]6c

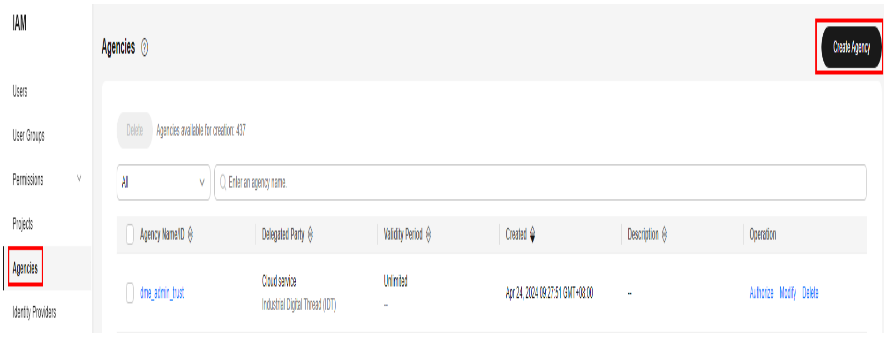
Projects

Project ID	Project Name
833[redacted]	[redacted]

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 2-6**.

Figure 2-6 Cluster ID



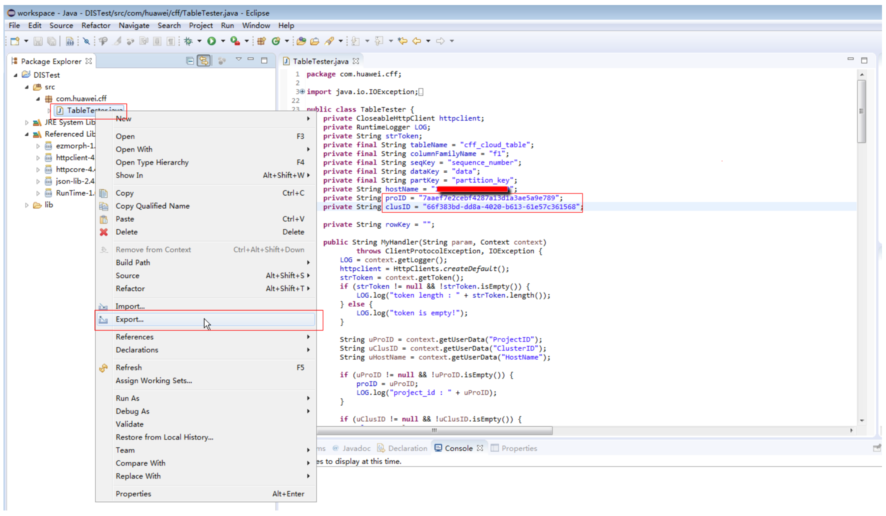
Agency Name/ID	Delegated Party	Validity Period	Created	Description	Operation
dme_admin_trust	Cloud service Industrial Digital Thread (IDT)	Unlimited --	Apr 24, 2024 09:27:51 GMT+08:00	--	Authorize Modify Delete

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 2-7 Exporting the code



```
package com.huawei.cff;

import java.io.IOException;

public class TableTester {
    private CloseableHttpClient httpclient;
    private RuntimeLogger LOG;
    private String strToken;
    private final String tableName = "cff_cloud_table";
    private final String columnFamilyName = "f1";
    private final String seqKey = "sequence_number";
    private final String dataKey = "data";
    private final String partKey = "partition_key";
    private String hostName = "[redacted]";
    private String proID = "7aaef7e2cebf4287a13d1a3ae5a9e789";
    private String clusID = "66f383bd-dd8a-4020-b613-61e57c361568";

    private String rowKey = "";

    public String MyHandler(String param, Context context)
            throws ClientProtocolException, IOException {
        LOG = context.getLogger();
        httpclient = HttpClients.createDefault();
        strToken = context.getToken();
        if (strToken != null && !strToken.isEmpty()) {
            LOG.log("token length : " + strToken.length());
        } else {
            LOG.log("token is empty!");
        }

        String uProID = context.getUserData("ProjectID");
        String uClusID = context.getUserData("ClusterID");
        String uHostName = context.getUserData("HostName");

        if (uProID != null && !uProID.isEmpty()) {
            proID = uProID;
            LOG.log("project_id : " + uProID);
        }

        if (uClusID != null && !uClusID.isEmpty()) {
```

es to display at this time.

Figure 2-8 Selecting a file type

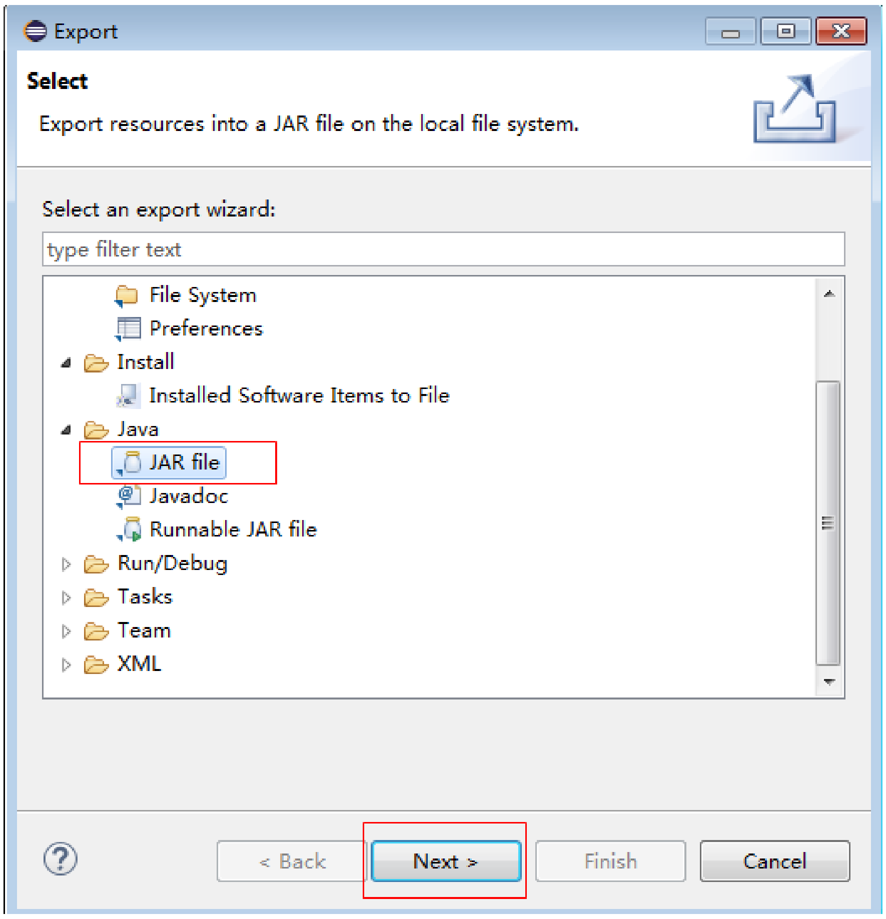
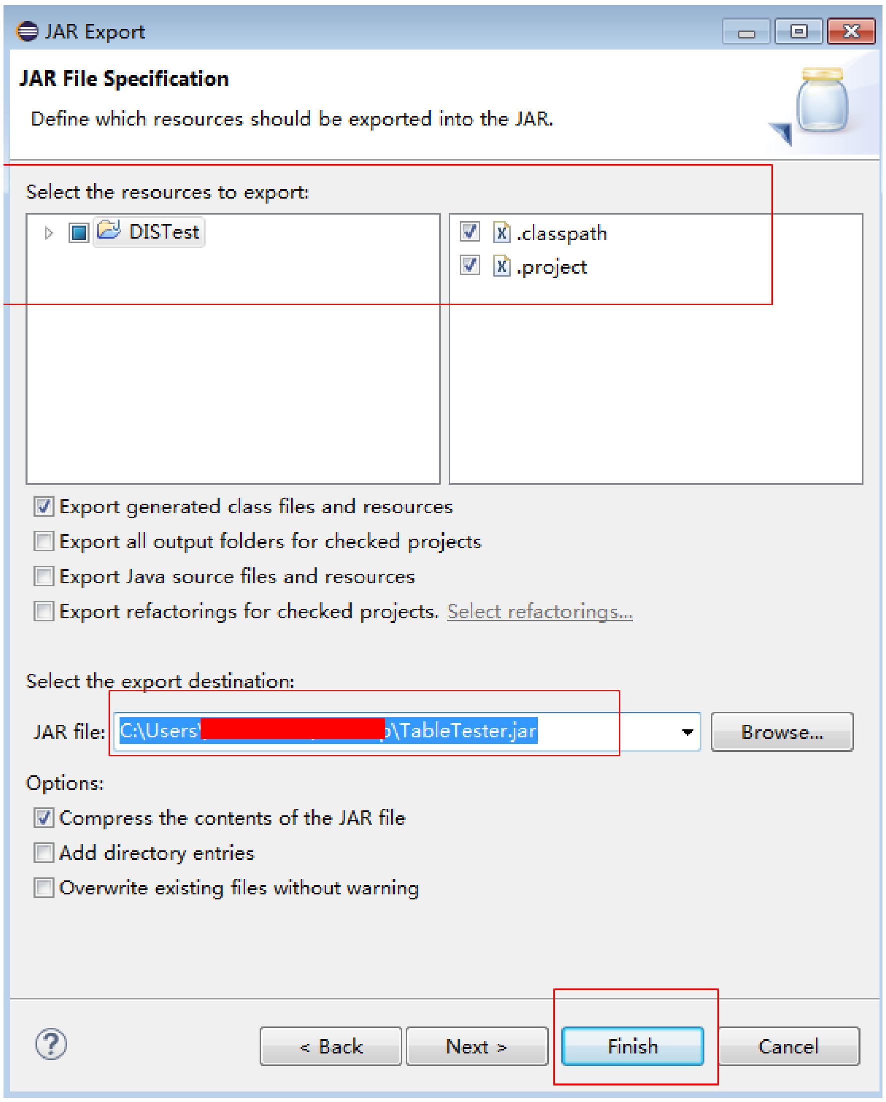


Figure 2-9 Publishing the code file


JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

DISTest

☑ .classpath
☑ .project

☑ Export generated class files and resources
☐ Export all output folders for checked projects
☐ Export Java source files and resources
☐ Export refactorings for checked projects. Select refactorings...

Select the export destination:

JAR file: C:\Users\[redacted]\TableTester.jar

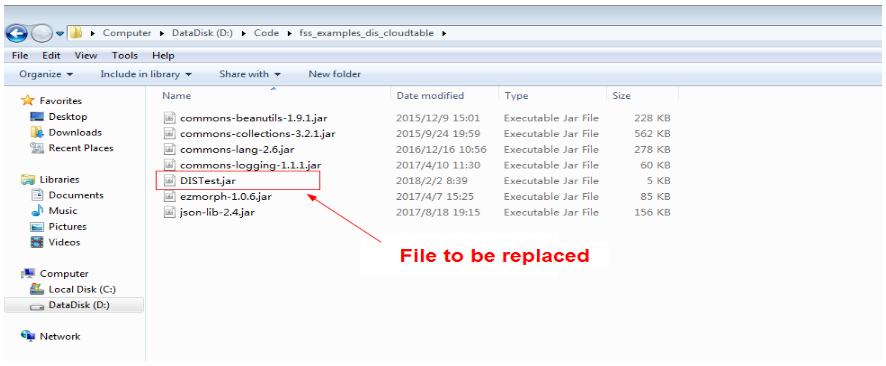
Options:

☑ Compress the contents of the JAR file
☐ Add directory entries
☐ Overwrite existing files without warning

Package the function dependencies by performing the following steps:

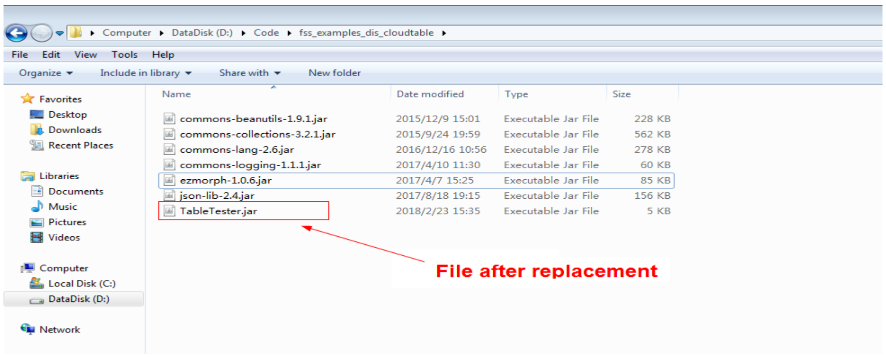
1. **Download program package fss_examples_dis_cloudtable.zip**, and decompress it, as shown in **Figure 2-10**.
2. Use **Table Tester.jar** to replace **DIS Test.jar**, as shown in **Figure 2-11**.
3. Package all of the files into **disdemo.zip**, as shown in **Figure 2-12**.

Figure 2-10 File directory before replacement



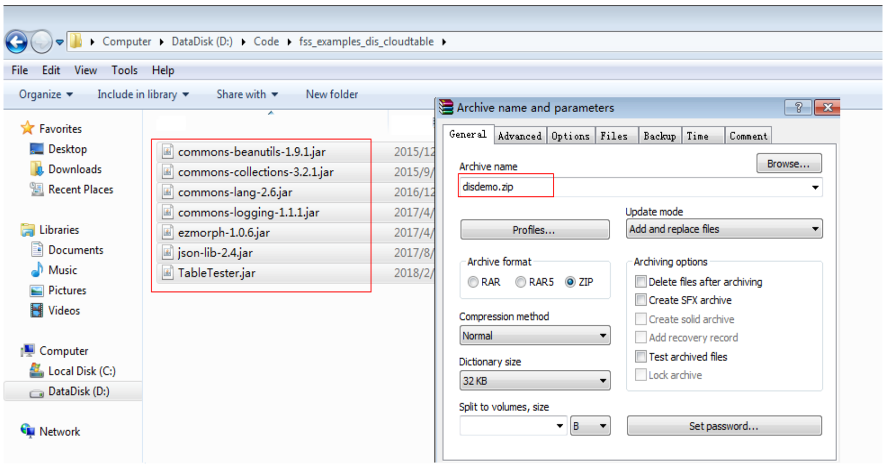
Name	Date modified	Type	Size
commons-beanutils-1.9.1.jar	2015/12/9 15:01	Executable Jar File	228 KB
commons-collections-3.2.1.jar	2015/9/24 19:59	Executable Jar File	562 KB
commons-lang-2.6.jar	2016/12/16 10:56	Executable Jar File	278 KB
commons-logging-1.1.1.jar	2017/4/10 11:30	Executable Jar File	60 KB
DISTest.jar	2018/2/2 8:39	Executable Jar File	5 KB
ezmorph-1.0.6.jar	2017/4/7 15:25	Executable Jar File	85 KB
json-lib-2.4.jar	2017/8/18 19:15	Executable Jar File	156 KB

Figure 2-11 File directory after replacement



Name	Date modified	Type	Size
commons-beanutils-1.9.1.jar	2015/12/9 15:01	Executable Jar File	228 KB
commons-collections-3.2.1.jar	2015/9/24 19:59	Executable Jar File	562 KB
commons-lang-2.6.jar	2016/12/16 10:56	Executable Jar File	278 KB
commons-logging-1.1.1.jar	2017/4/10 11:30	Executable Jar File	60 KB
ezmorph-1.0.6.jar	2017/4/7 15:25	Executable Jar File	85 KB
json-lib-2.4.jar	2017/8/18 19:15	Executable Jar File	156 KB
TableTester.jar	2018/2/23 15:35	Executable Jar File	5 KB

Figure 2-12 Packaging the files in ZIP format


Computer ▸ DataDisk (D:) ▸ Code ▸ fss_examples_dis_cloudtable
File Edit View Tools Help
Organize Include in library Share with New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
DataDisk (D:)
Network
commons-beanutils-1.9.1.jar
commons-collections-3.2.1.jar
commons-lang-2.6.jar
commons-logging-1.1.1.jar
ezmorph-1.0.6.jar
json-lib-2.4.jar
TableTester.jar
Archive name and parameters
General Advanced Options Files Backup Time Comment
Browse...
Profiles...
Set password...

Archive name: disdemo.zip
Update mode: Add and replace files
Archive format: RAR RAR5 [✓ ZIP]
Compression method: Normal
Dictionary size: 32 KB
Split to volumes, size: B
Archiving options: Delete files after archiving, Create SFX archive, Create solid archive, Add recovery record, Test archived files, Lock archive

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 Select **Create from scratch**, set the function information, and click **Create Function**.

- For **Function Type**, select **Event Function**.
- For **Function Name**: Enter a function name, for example, **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**.

----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**.

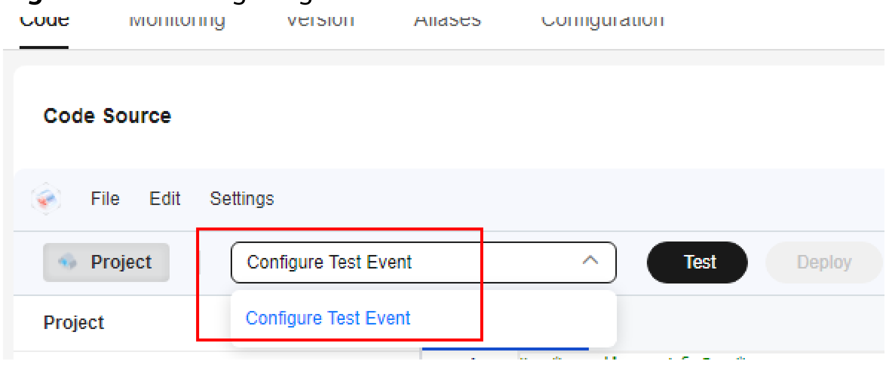
----End

2.4 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 2-13**.

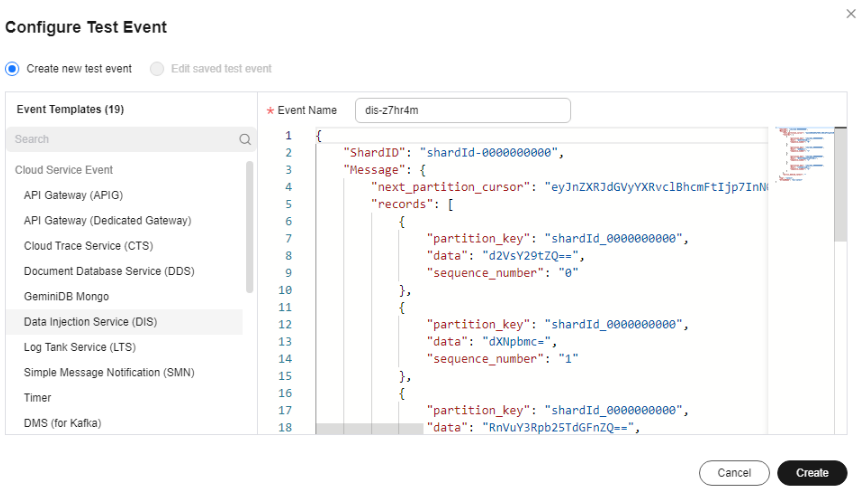
Figure 2-13 Configuring a test event



Step 2 In the **Configure Test Event** dialog box, set the test event information, as shown in **Figure 2-14**.

- Select **Create new test event**.
- Event Template: Select **Data Ingestion Service (DIS)**.
- **Event Name**: Enter an event name, for example, **dis-test**.

Figure 2-14 Test event



Event Name: dis-z7hr4m

```
{
    "ShardID": "shardId-0000000000",
    "Message": {
        "next_partition_cursor": "eyJnZXRJdGVyYXRvclBhcmFtIjp7InN
        "records": [
            {
                "partition_key": "shardId_0000000000",
                "data": "d2VsY29tZQ==",
                "sequence_number": "0"
            },
            {
                "partition_key": "shardId_0000000000",
                "data": "dXNpbmc=",
                "sequence_number": "1"
            },
            {
                "partition_key": "shardId_0000000000",
                "data": "RnVuY3Rpb25TdGFnZQ==",
```

Step 3 Click **Create**.

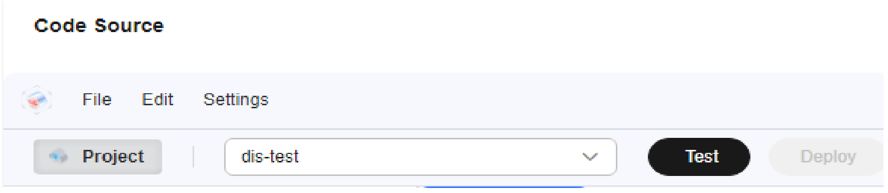
----End

2.5 Processing Data

Perform the following procedure to process simulated stream data:

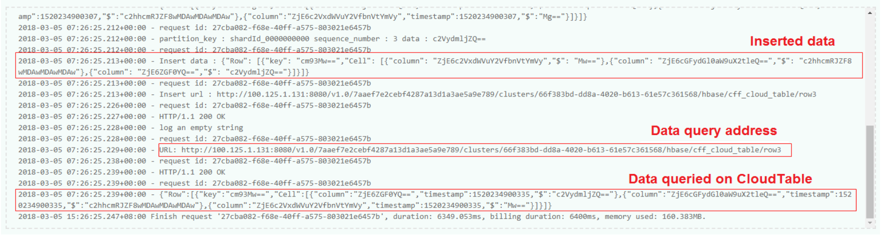
Step 1 On the **DISDemo** page, select test event **dis-test**, and click **Test** to test the function, as shown in **Figure 2-15**.

Figure 2-15 Configuring a test event



Step 2 After the function is executed successfully, check the logs shown in **Figure 2-16**. For all logs of the function, go to the **Logs** tab page.

Figure 2-16 Function execution result



```
amp":1520234900307,"$":"c2hhcmRJZF8wMDAwMDAwMDAw"},{"column":"ZjE6c2VxdWVuY2VfbnVtYmVy","timestamp":1520234900307,"$":"Mg=="}]}]}
2018-03-05 07:26:25.212+00:00 - request id: 27cba082-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.212+00:00 - partition_key : shardId_0000000000 sequence_number : 3 data : c2VydmljZQ==
2018-03-05 07:26:25.212+00:00 - request id: 27cba082-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.213+00:00 - Insert data : {"Row": [{"key": "cm93Mw==","Cell": [{"column": "ZjE6c2VxdWVuY2VfbnVtYmVy","$": "Mw=="},{"column": "ZjE6cGFydGl0aW9uX2tleQ==","$": "c2hhcmRJZF8
wMDAwMDAwMDAw"},{"column": "ZjE6ZGF0YQ==","$": "c2VydmljZQ=="}]}]}
2018-03-05 07:26:25.213+00:00 - request id: 27cba082-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.213+00:00 - Insert url : http://100.125.1.131:8080/v1.0/7aaef7e2cebf4287a13d1a3ae5a9e789/clusters/66f383bd-dd8a-4020-b613-61e57c361568/hbase/cff_cloud_table/row3
2018-03-05 07:26:25.226+00:00 - request id: 27cba082-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.227+00:00 - HTTP/1.1 200 OK
2018-03-05 07:26:25.228+00:00 - log an empty string
2018-03-05 07:26:25.228+00:00 - request id: 27cba082-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.229+00:00 - URL: http://100.125.1.131:8080/v1.0/7aaef7e2cebf4287a13d1a3ae5a9e789/clusters/66f383bd-dd8a-4020-b613-61e57c361568/hbase/cff_cloud_table/row3
2018-03-05 07:26:25.238+00:00 - request id: 27cba082-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.239+00:00 - HTTP/1.1 200 OK
2018-03-05 07:26:25.239+00:00 - request id: 27cba082-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.239+00:00 - {"Row":[{"key":"cm93Mw==","Cell":[{"column":"ZjE6ZGF0YQ==","timestamp":1520234900335,"$":"c2VydmljZQ=="},{"column":"ZjE6cGFydGl0aW9uX2tleQ==","timestamp":152
0234900335,"$":"c2hhcmRJZF8wMDAwMDAwMDAw"},{"column":"ZjE6c2VxdWVuY2VfbnVtYmVy","timestamp":1520234900335,"$":"Mw=="}]}]}
2018-03-05 15:26:25.247+08:00 Finish request '27cba082-f68e-40ff-a575-803021e6457b', duration: 6349.053ms, billing duration: 6400ms, memory used: 160.383MB.
```

----End

3 Integrating with LTS to Analyze Logs in Real Time

3.1 Introduction

Scenarios

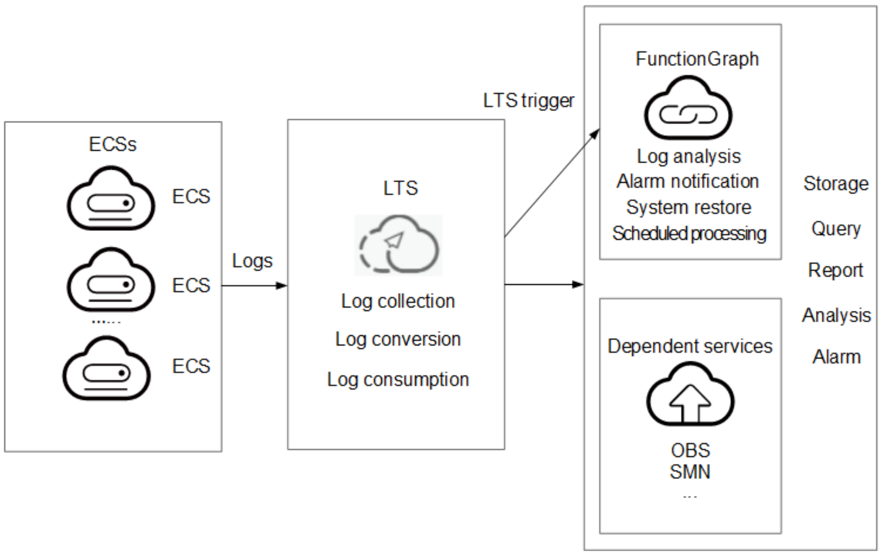
Quickly collect, process, and convert task logs of servers, such as ECSs, through Log Tank Service (LTS).

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 3-1**.

Figure 3-1 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.

3.2 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.

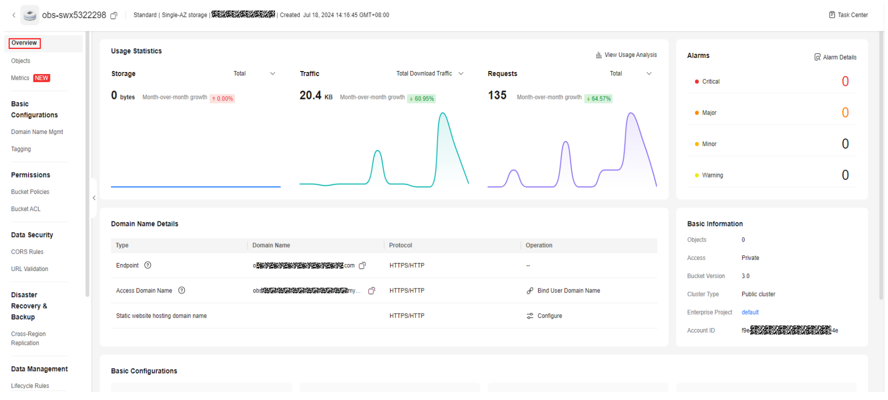
  In this example, when log events trigger the specified function through an LTS trigger, the function filters alarm logs and pushes alarm message to the subscription endpoints.

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 3-2**. For details, see **Creating a Bucket**.

Figure 3-2 Creating a bucket


obs-swx5322298 | Standard | Single-AZ storage | [redacted] | Created Jul 18, 2024 14:16:45 GMT+08:00 | Task Center

Overview, Objects, Metrics NEW, Basic Configurations, Domain Name Mgmt, Tagging, Permissions, Bucket Policies, Bucket ACL, Data Security, CORS Rules, URL Validation, Disaster Recovery & Backup, Cross-Region Replication, Data Management, Lifecycle Rules

Usage Statistics (View Usage Analysis)

Storage (Total): 0 bytes, Month-over-month growth ↑ 0.00%

Traffic (Total Download Traffic): 20.4 KB, Month-over-month growth ↓ 60.95%

Requests (Total): 135, Month-over-month growth ↓ 64.57%

Alarms (Alarm Details): Critical 0, Major 0, Minor 0, Warning 0

Domain Name Details

Type	Domain Name	Protocol	Operation
Endpoint	o[redacted].com	HTTPS/HTTP	--
Access Domain Name	obs[redacted]my...	HTTPS/HTTP	Bind User Domain Name
Static website hosting domain name		HTTPS/HTTP	Configure

Basic Information

Objects	0
Access	Private
Bucket Version	3.0
Cluster Type	Public cluster
Enterprise Project	default
Account ID	f9e[redacted]4e

Basic Configurations

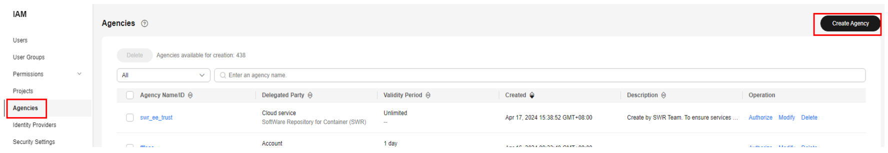
NOTE

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 3-3 Creating an agency


IAM: Users, User Groups, Permissions, Projects, Agencies, Identity Providers, Security Settings

Agencies | Create Agency

Delete | Agencies available for creation: 438

All | Enter an agency name.

Agency Name/ID	Delegated Party	Validity Period	Created	Description	Operation
swr_ee_trust	Cloud service SoftWare Repository for Container (SWR)	Unlimited --	Apr 17, 2024 15:38:52 GMT+08:00	Create by SWR Team. To ensure services ...	Authorize Modify Delete
	Account	1 day			

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

   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.

3.3 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 3-1**.

Table 3-1 Environment variables

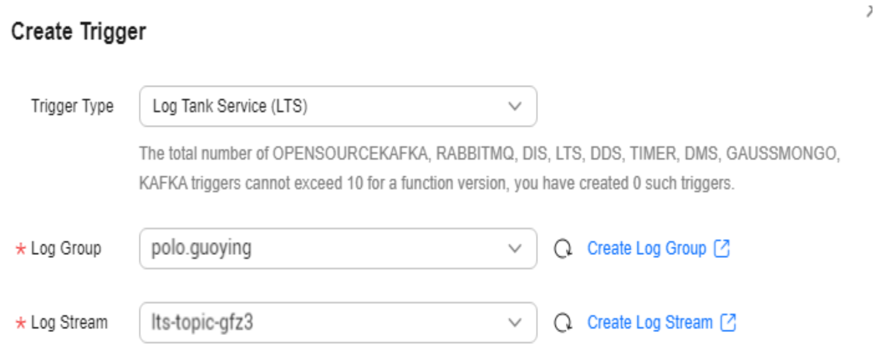
Environment Variable	Description
obs_address	OBS endpoint. To obtain the OBS endpoint, see **Regions and Endpoints**.
obs_store_bucket	Name of the destination bucket for storing logs.
obs_store_objName	Name of the target file for storing logs.
SMN_Topic	SMN topic.
region	Name of your region. To obtain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

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

3.4 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 3-4**.

Figure 3-4 Creating an LTS trigger

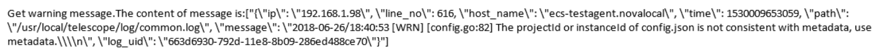

Create Trigger
Trigger Type
Log Tank Service (LTS)
The total number of OPENSOURCEKAFKA, RABBITMQ, DIS, LTS, DDS, TIMER, DMS, GAUSSMONGO, KAFKA triggers cannot exceed 10 for a function version, you have created 0 such triggers.
* Log Group
polo.guoying
Create Log Group
* Log Stream
lts-topic-gfz3
Create Log Stream

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.

3.5 Processing Log Data

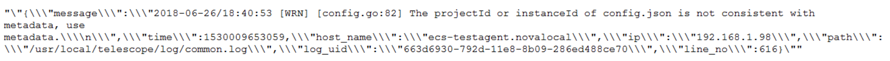
Email notifications will be received from SMN if alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR** are generated, as shown in **Figure 3-5**. You can also view details of the alarm logs by opening the **log.txt** file in the specified bucket, as shown in **Figure 3-6**.

Figure 3-5 Email notification



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

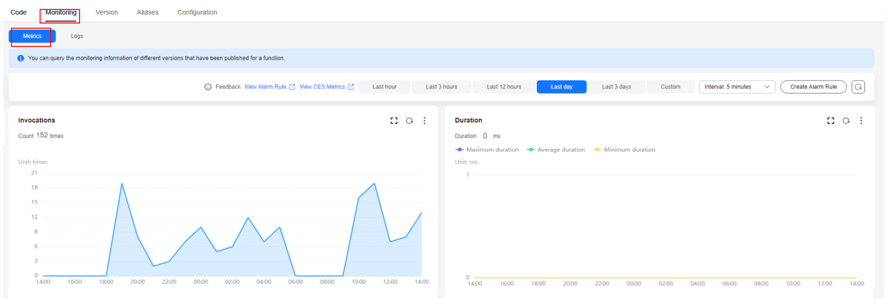
Figure 3-6 Alarm log details



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

On the **Monitoring** tab page of the function, check the number of invocations, as shown in **Figure 3-7**.

Figure 3-7 Function metrics



3.6 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.

4 Integrating with CTS to Analyze Login/Logout Security

4.1 Introduction

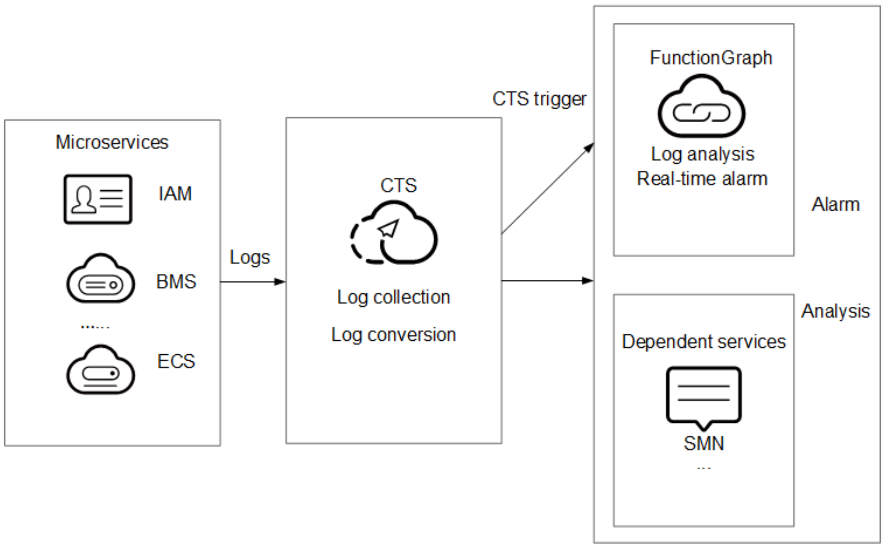
Scenarios

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 4-1**.

Figure 4-1 Processing workflow



Values

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

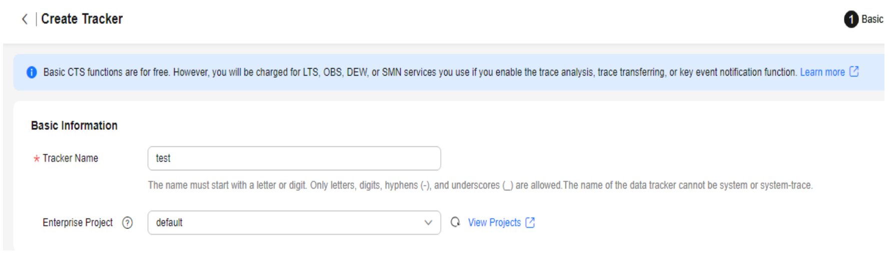
- Processes and analyzes data in response to log events in a serverless architecture, which features automatic scaling, no operation and maintenance, and pay-per-use billing.
- Sends alarm notifications through SMN.

4.2 Preparation

Enabling CTS

Configure a tracker on CTS, as shown in **Figure 4-2**. For details, see **Configuring a Tracker**.

Figure 4-2 Configuring a tracker



Basic CTS functions are for free. However, you will be charged for LTS, OBS, DEW, or SMN services you use if you enable the trace analysis, trace transferring, or key event notification function. Learn more

Basic Information

* Tracker Name: test

The name must start with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.The name of the data tracker cannot be system or system-trace.

Enterprise Project: default

View Projects

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 an agency name, for example, **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 **CTS Administrator** and **SMN Administrator**.

NOTE

- **SMN Administrator**: Users with this permission can perform any operation on SMN resources.
- **CTS Administrator** depends on **Tenant Guest**. When you select the former, the latter will also be selected.

Step 5 Click **Next**, select an authorization scope that meets your service requirements, and 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.

4.3 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 4-1**.

Table 4-1 Environment variables

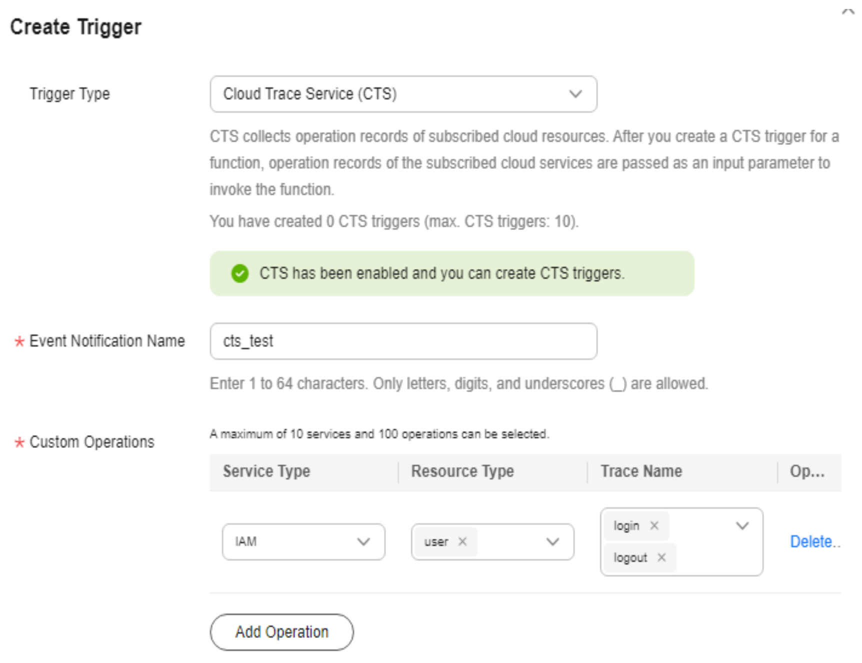
Environment Variable	Description
SMN_Topic	SMN topic.
RegionName	Region name.
IP	IP address whitelist.

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

4.4 Adding an Event Source

Create a CTS trigger, as shown in **Figure 4-3**.

Figure 4-3 Creating a CTS trigger



Create Trigger

Trigger Type: Cloud Trace Service (CTS)

CTS collects operation records of subscribed cloud resources. After you create a CTS trigger for a function, operation records of the subscribed cloud services are passed as an input parameter to invoke the function.

You have created 0 CTS triggers (max. CTS triggers: 10).

CTS has been enabled and you can create CTS triggers.

* Event Notification Name: cts_test

Enter 1 to 64 characters. Only letters, digits, and underscores (_) are allowed.

* Custom Operations

A maximum of 10 services and 100 operations can be selected.

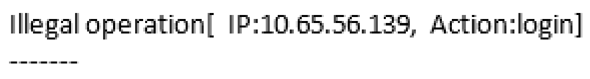
Service Type	Resource Type	Trace Name	Op...
IAM	user	login, logout	Delete..

CTS records the logins and logouts of users on IAM.

4.5 Processing Operation Records

The function runs in response to account logins and logouts to filter those not from the IP address whitelist, and sends a message or email through SMN, as shown in **Figure 4-4**.

Figure 4-4 Email notification

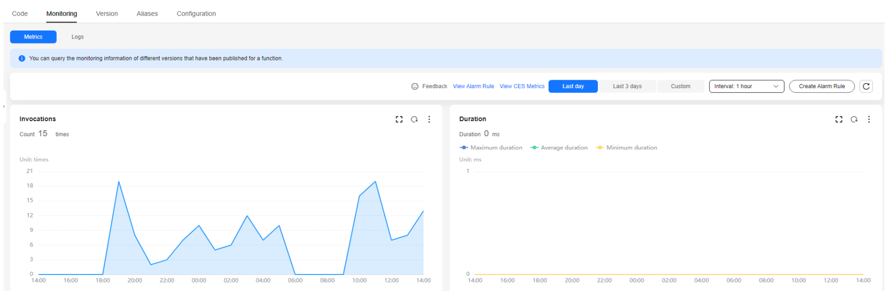


Illegal operation[IP:10.65.56.139, Action:login]

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

On the **Monitoring** tab page of the function, check the number of invocations, as shown in **Figure 4-5**.

Figure 4-5 Function metrics



You can query the monitoring information of different versions that have been published for a function.

5 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.

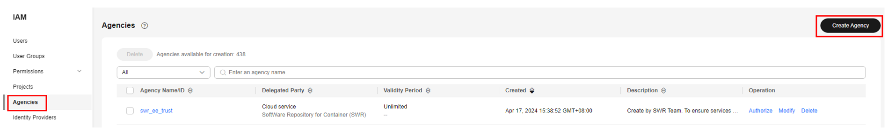
Prerequisites

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 5-1 Creating an agency

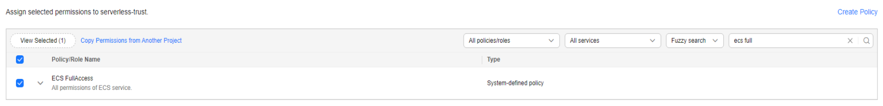


Agencies available for creation: 438

Agency Name/ID	Delegated Party	Validity Period	Created	Description	Operation
swr_ee_trust	Cloud service SoftWare Repository for Container (SWR)	Unlimited --	Apr 17, 2024 15:38:52 GMT+08:00	Create by SWR Team. To ensure services ...	Authorize Modify Delete

3. Configure the agency.
   - For **Agency Name**: Enter an agency name, for example, **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 5-2 Selecting the permission


Assign selected permissions to serverless-trust.

	Policy/Role Name	Type
✓	ECS FullAccess All permissions of ECS service.	System-defined policy

5. Click **Next**, select an authorization scope that meets your service requirements, and click **OK**.

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 5-1**.

Table 5-1 Environment variables

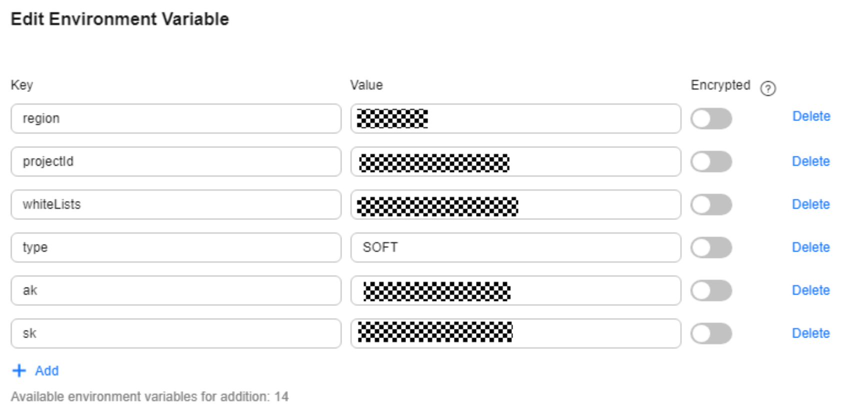
Environment Variable	Description
region	Region where your ECSs are located, for example, **ap-southeast-1**.
projectId	ID of the project to which the ECS belongs. For details about how to obtain the project ID, see **Obtaining a Project ID**.
whiteLists	● If you want to periodically **start** certain 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.

Figure 5-3 Setting environment variables



Edit Environment Variable

Key	Value	Encrypted	
region	[redacted]		Delete
projectId	[redacted]		Delete
whiteLists	[redacted]		Delete
type	SOFT		Delete
ak	[redacted]		Delete
sk	[redacted]		Delete

Available environment variables for addition: 14

- If a large number of ECSs need to be started or stopped, increase the execution timeout for your function.
- Environment variables in **Table 5-1** 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

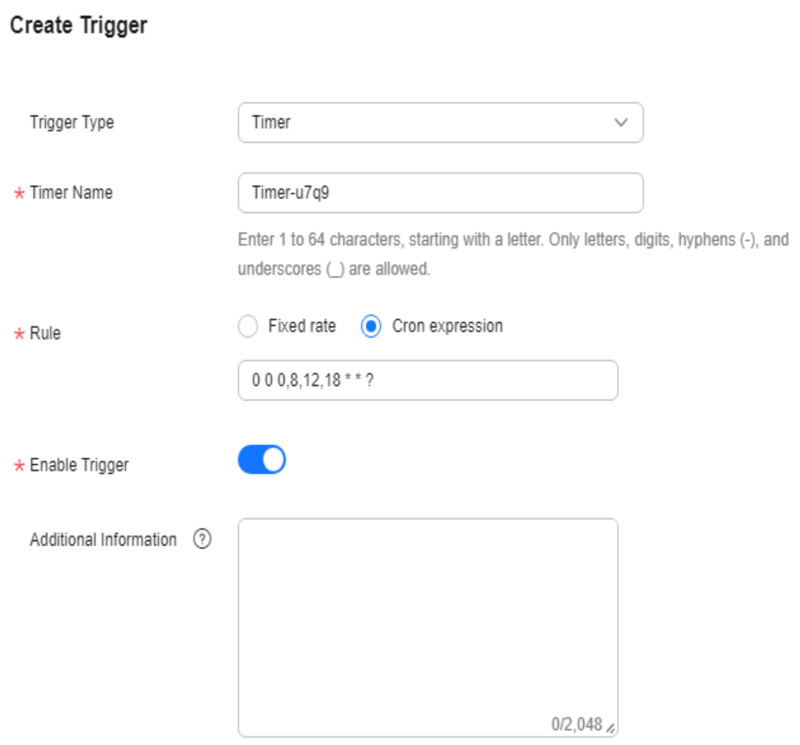
If **huaweicloudsdk_ecs_core_py3.6** is not available in your region, contact customer service.

----End

Adding an Event Source

Create a timer trigger and set the trigger parameters according to **Figure 5-4**.

Figure 5-4 Creating a timer trigger



Create Trigger

Trigger Type	Timer
* Timer Name	Timer-u7q9
	Enter 1 to 64 characters, starting with a letter. Only letters, digits, hyphens (-), and underscores (_) are allowed.
* Rule	Fixed rate [✓ Cron expression]
	0 0 0,8,12,18 * * ?
* Enable Trigger	●
Additional Information	0/2,048

6 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

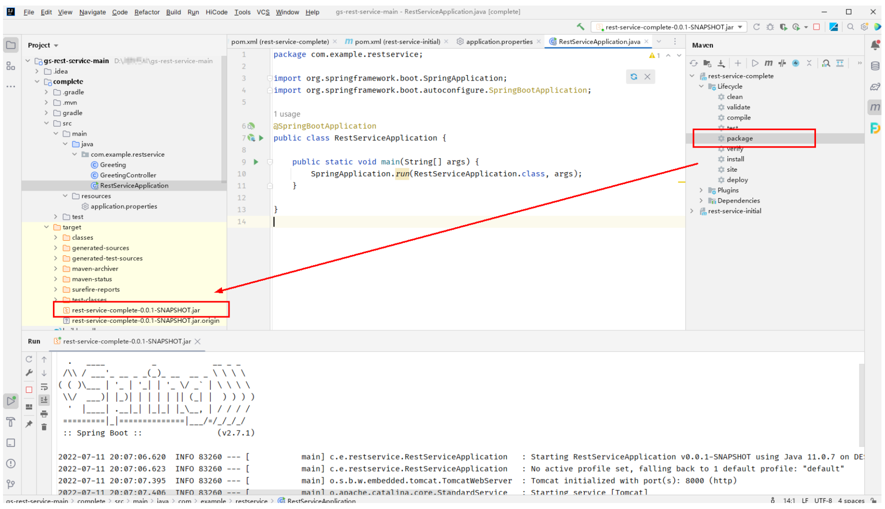
To deploy an existing project to FunctionGraph, change the listening port of the project to **8000**, and create a file named **bootstrap** in the same directory as the JAR file to include the command for executing the JAR file.

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 6-1 Generating a JAR file

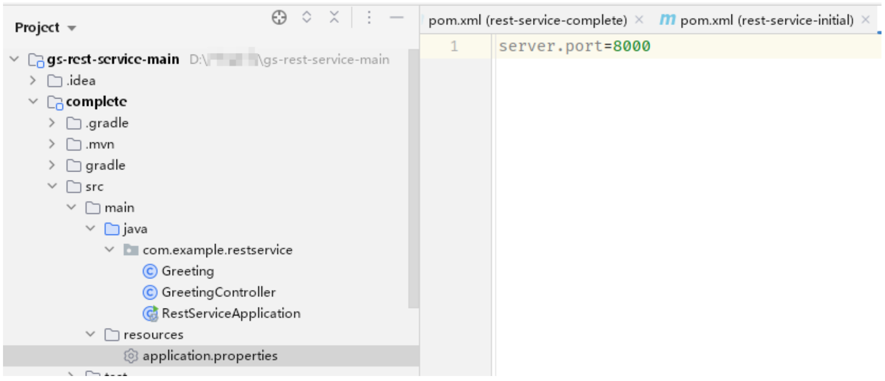


```java
package com.example.restservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RestServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestServiceApplication.class, args);
    }

}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

2022-07-11 20:07:06.620  INFO 83260 --- [           main] c.e.restservice.RestServiceApplication   : Starting RestServiceApplication v0.0.1-SNAPSHOT using Java 11.0.7 on DE
2022-07-11 20:07:06.623  INFO 83260 --- [           main] c.e.restservice.RestServiceApplication   : No active profile set, falling back to 1 default profile: "default"
2022-07-11 20:07:07.395  INFO 83260 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8000 (http)
2022-07-11 20:07:07.406  INFO 83260 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
```

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 6-2 Configuring port 8000



```
server.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
```

NOTE

The Java runtime environment can be directly invoked in the function, and no additional installation is required.

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

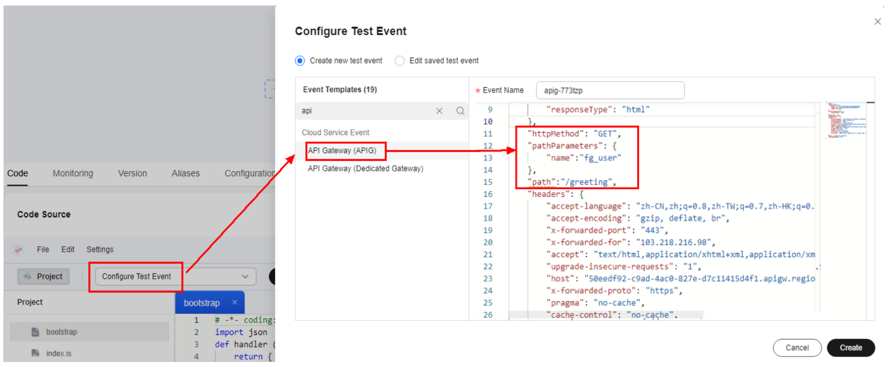
Creating an HTTP Function and Uploading Code

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, and modify the **path** and **pathParameters** parameters in the template to construct a simple GET request.

Figure 6-3 Configuring a test event



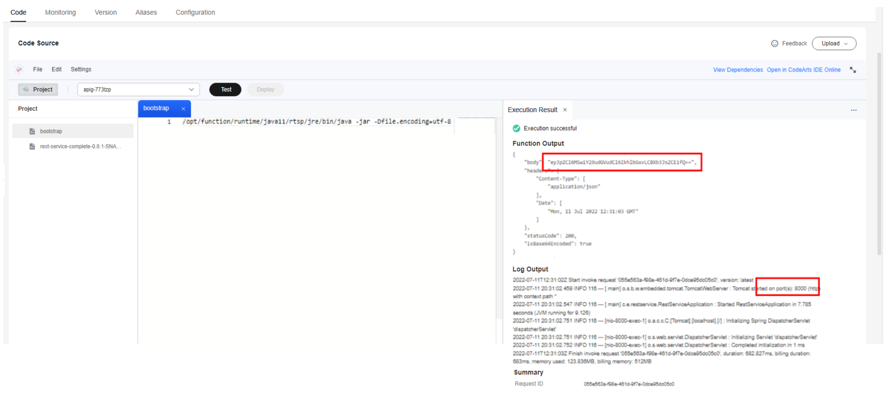
```
        "responseType": "html"
    },
    "httpMethod": "GET",
    "pathParameters": {
        "name":"fg_user"
    },
    "path":"/greeting",
    "headers": {
        "accept-language": "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.
        "accept-encoding": "gzip, deflate, br",
        "x-forwarded-port": "443",
        "x-forwarded-for": "103.218.216.98",
        "accept": "text/html,application/xhtml+xml,application/xm
        "upgrade-insecure-requests": "1",
        "host": "50eedf92-c9ad-4ac0-827e-d7c11415d4f1.apigw.regio
        "x-forwarded-proto": "https",
        "pragma": "no-cache",
        "cache-control": "no-cache",
```

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.

Figure 6-4 Viewing the returned result



```
/opt/function/runtime/java11/rtsp/jre/bin/java -jar -Dfile.encoding=utf-8
```

Execution successful

Function Output

```
{
    "body": "eyJpZCI6MSwiY29udGVudCI6IkhlbGxvLCBXb3JsZCEifQ==",
    "headers": {
        "Content-Type": [
            "application/json"
        ],
        "Date": [
            "Mon, 11 Jul 2022 12:31:03 GMT"
        ]
    },
    "statusCode": 200,
    "isBase64Encoded": true
}
```

Log Output

```
2022-07-11T12:31:02Z Start invoke request '055e563a-f98e-461d-9f7e-0dce95dc05c0', version: latest
2022-07-11 20:31:02.459 INFO 116 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8000 (http)
with context path ''
2022-07-11 20:31:02.547 INFO 116 --- [ main] c.e.restservice.RestServiceApplication : Started RestServiceApplication in 7.785
seconds (JVM running for 9.126)
2022-07-11 20:31:02.751 INFO 116 --- [nio-8000-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet
'dispatcherServlet'
2022-07-11 20:31:02.751 INFO 116 --- [nio-8000-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-07-11 20:31:02.752 INFO 116 --- [nio-8000-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-07-11T12:31:03Z Finish invoke request '055e563a-f98e-461d-9f7e-0dce95dc05c0', duration: 682.827ms, billing duration:
683ms, memory used: 123.836MB, billing memory: 512MB
```

Summary

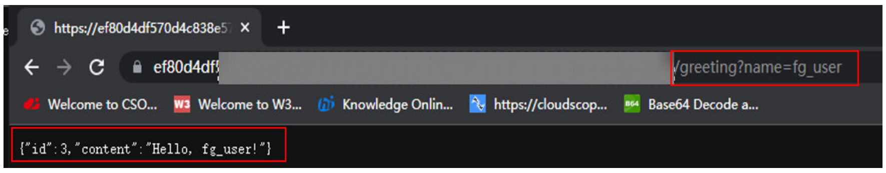
Request ID 055e563a-f98e-461d-9f7e-0dce95dc05c0

- 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 6-5**), and access the URL using a browser. The response shown in the following figure is displayed.

Figure 6-5 Invoking the function

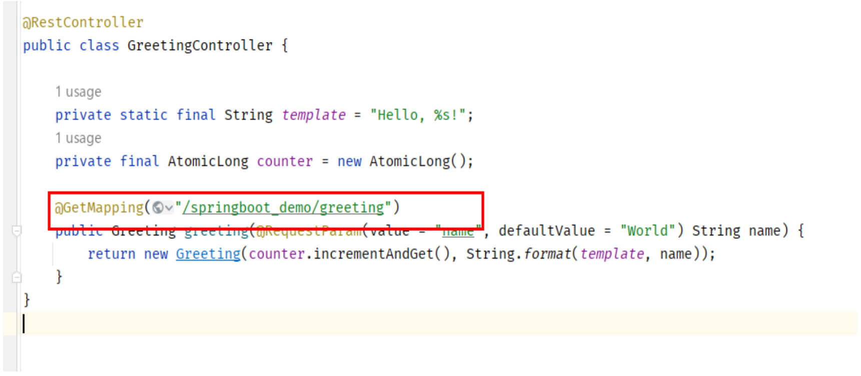


```
{"id":3,"content":"Hello, fg_user!"}
```

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 6-6 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.

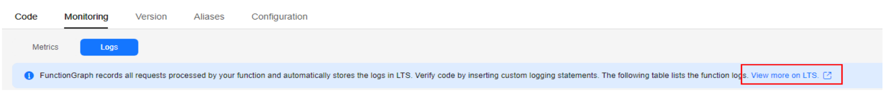
2. How Are My Logs Collected and Output?

   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.

   Figure 6-7 Accessing LTS for log analysis

   FunctionGraph records all requests processed by your function and automatically stores the logs in LTS. Verify code by inserting custom logging statements. The following table lists the function logs. View more on LTS.

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>
```

7 Creating a FunctionGraph Backend API That Uses a Custom Authorizer

7.1 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

7.2 Resource Planning

Ensure that the following resources are in the same region.

Table 7-1 Resource planning

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

7.3 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**.

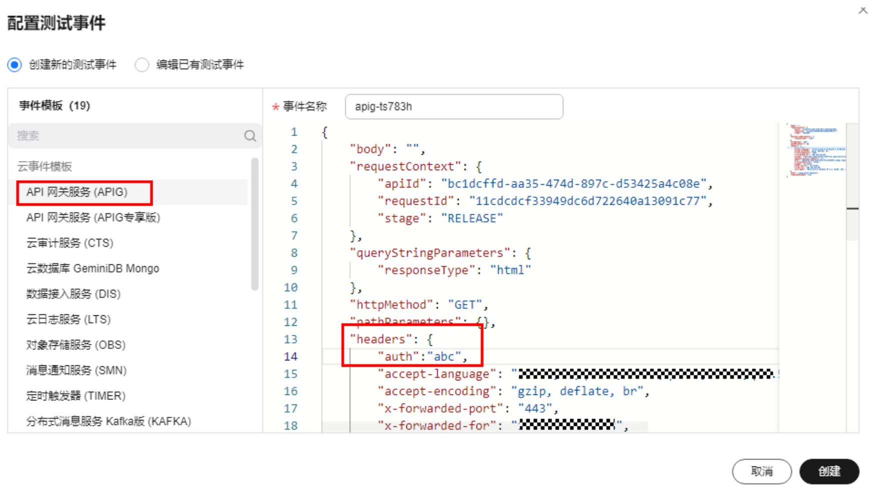
- **Template**: Select **Create from scratch**.

- **Function Type**: Select **Event Function**.
- **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 an event template. Modify the template as required, and click **Create**. In this example, add **"auth":"abc"** to **"headers"**.

Figure 7-1 Configuring a test event

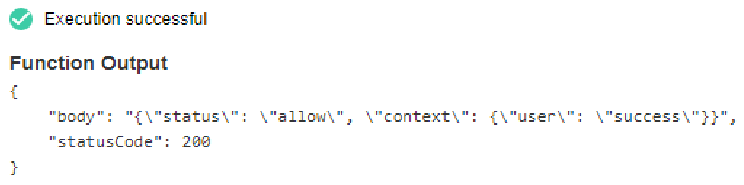


事件名称 apig-ts783h

```
{
    "body": "",
    "requestContext": {
        "apiId": "bc1dcffd-aa35-474d-897c-d53425a4c08e",
        "requestId": "11cdcdcf33949dc6d722640a13091c77",
        "stage": "RELEASE"
    },
    "queryStringParameters": {
        "responseType": "html"
    },
    "httpMethod": "GET",
    "pathParameters": {},
    "headers": {
        "auth":"abc",
        "accept-language": "[redacted]
        "accept-encoding": "gzip, deflate, br",
        "x-forwarded-port": "443",
        "x-forwarded-for": "[redacted]",
```

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

Figure 7-2 Viewing the execution result



Execution successful

Function Output

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

----End

Creating a Custom Authorizer

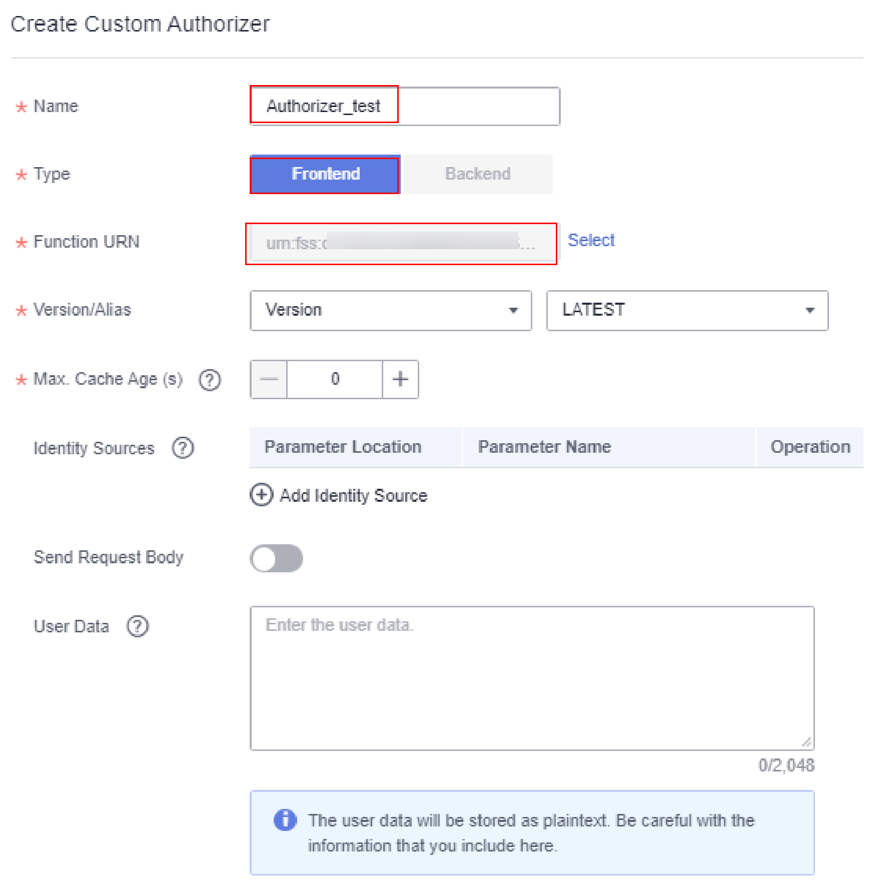
Create a custom authorizer in APIG and connect it to the frontend custom authentication function.

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 7-3 Creating a custom authorizer


Create Custom Authorizer

* Name	Authorizer_test
* Type	Frontend / Backend
* Function URN	urn:fss:[redacted] Select
* Version/Alias	Version ▾ LATEST ▾
* Max. Cache Age (s) ⓘ	— 0 +
Identity Sources ⓘ	Parameter Location / Parameter Name / Operation; Add Identity Source
Send Request Body	
User Data ⓘ	Enter the user data. 0/2,048

The user data will be stored as plaintext. Be careful with the information that you include here.

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)
```

7.4 Adding an Event Source

Creating an API

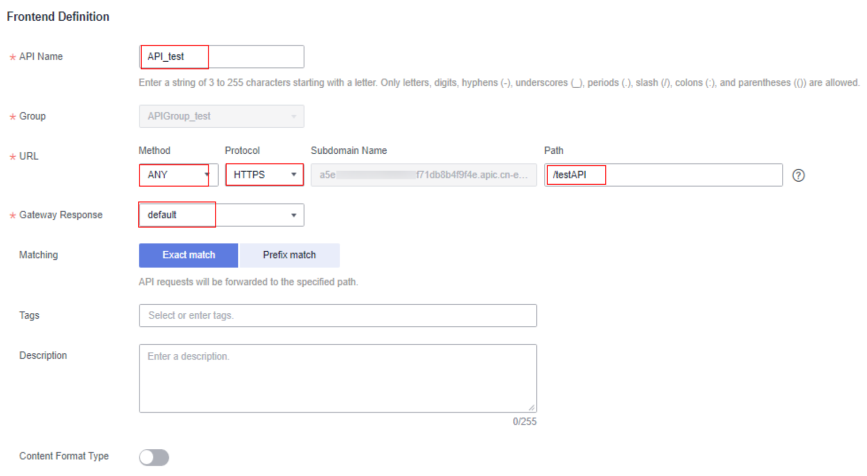
After creating an API group, custom authentication function, and backend function, create a FunctionGraph backend API that uses a custom authorizer by performing the following steps:

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 7-4** and **Figure 7-5**.

- **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 7-4 Configuring frontend definition



Frontend Definition

* API Name: API_test
  Enter a string of 3 to 255 characters starting with a letter. Only letters, digits, hyphens (-), underscores (_), periods (.), slash (/), colons (:), and parentheses (()) are allowed.
* Group: APIGroup_test
* URL: Method: ANY; Protocol: HTTPS; Subdomain Name: a5e[redacted]f71db8b4f9f4e.apic.cn-e...; Path: /testAPI
* Gateway Response: default

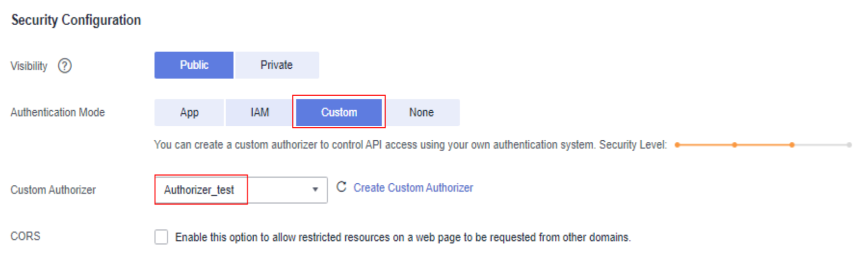
Matching: Exact match | Prefix match
API requests will be forwarded to the specified path.

Tags: Select or enter tags.

Description: Enter a description.
0/255

Content Format Type

Figure 7-5 Configuring security settings


Security Configuration

Visibility: Public

Authentication Mode: Custom

You can create a custom authorizer to control API access using your own authentication system. Security Level:

Custom Authorizer: Authorizer_test

CORS: Enable this option to allow restricted resources on a web page to be requested from other domains.

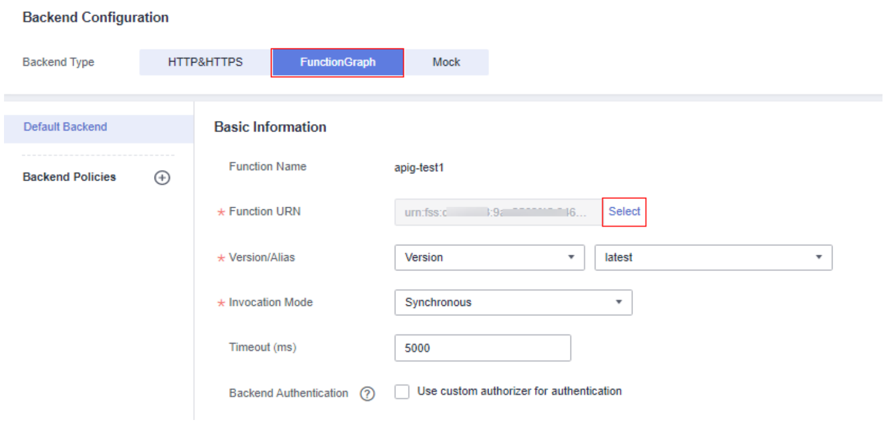
NOTE

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

Step 3 Click **Next** to configure the backend service according to **Figure 7-6**.

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

Figure 7-6 Configuring the backend service


Backend Configuration

Backend Type: FunctionGraph

Basic Information

Function Name: apig-test1

* Function URN: urn:fss:c

* Version/Alias: Version, latest

* Invocation Mode: Synchronous

Timeout (ms): 5000

Backend Authentication: Use custom authorizer for authentication

Step 4 Click **Finish**.

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

Figure 7-7 Publishing an API

API_test
ID: d26

Create Policy
Publish

----End

7.5 Debugging and Calling the API

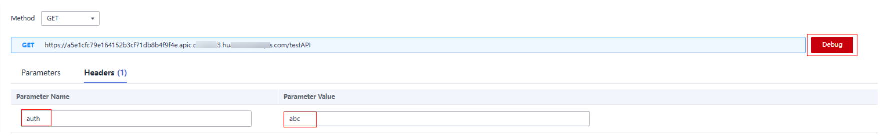
APIG provides online debugging, enabling you to check an API after configuring it.

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**.

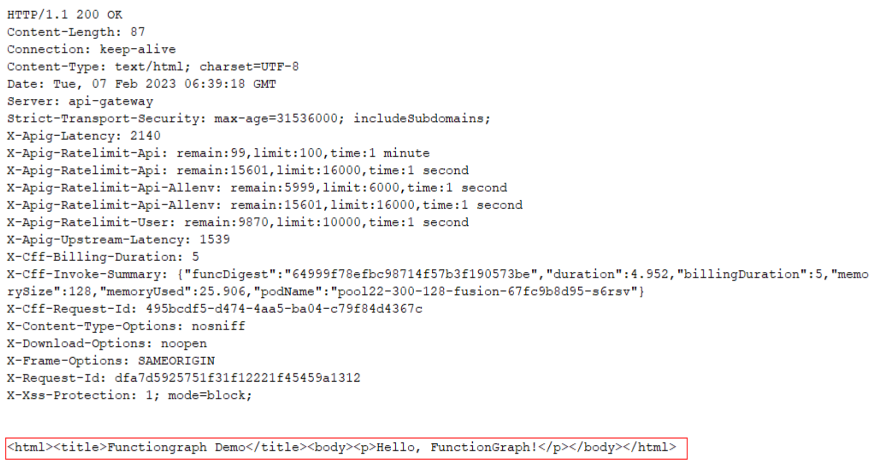
Figure 7-8 Adding a header



https://a5e1cfc79e164152b3cf71db8b4f9f4e.apic.c[redacted]3.hu[redacted]s.com/testAPI

Step 3 Check whether the API response contains the content you have defined in the service function. See **Figure 7-9**.

Figure 7-9 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

8 Uploading Files with FunctionGraph and APIG

8.1 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.

8.2 Resource Planning

Table 8-1 Resource planning

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

8.3 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.

8.3.1 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 a function name, for example, **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 a dependency name, for example, **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.

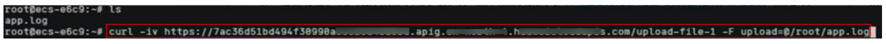
The curl tool is used as an example (**curl -F** is mainly used in Linux). You can also use other tools such as Postman. Create a file named **app.log** with any content on your local host. Example:

```
start something
run
stop all
```

Run the following command:

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

Figure 8-1 Example



```
root@ecs-e6c9:~# ls
app.log
root@ecs-e6c9:~# curl -iv https://7ac36d51bd494f30990a[redacted].apig.[redacted].h[redacted]s.com/upload-file-1 -F upload=@/root/app.log
```

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

8.3.2 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 a function name, for example, **upload-file-1**.
   - **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.

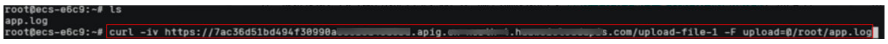
Create a file named **app.log** with any content on your local host. Example:

```
start something
run
stop all
```

- Take the curl tool as an example (**curl -F** is mainly used in the Linux environment). Run the following command:

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

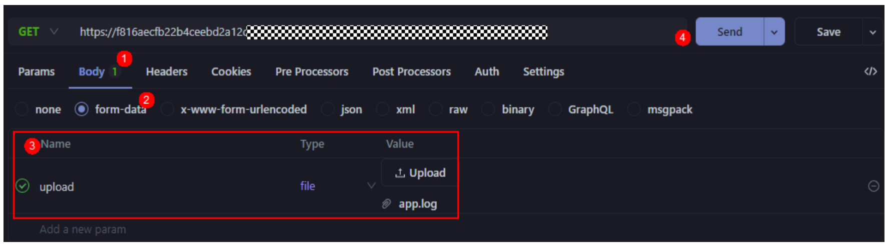
  Figure 8-2 Example

  ```
  root@ecs-e6c9:~# ls
  app.log
  root@ecs-e6c9:~# curl -iv https://7ac36d51bd494f30990a[redacted].apig.[redacted].h[redacted]s.com/upload-file-1 -F upload=@/root/app.log
  ```

- Take the Postman tool as an example. Set the following parameters and click **Send**.

Figure 8-3 Example



GET https://f816aecfb22b4ceebd2a12[redacted]

Name	Type	Value
upload	file	app.log

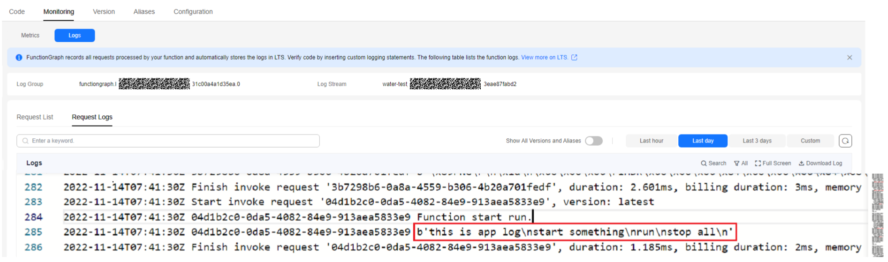
Name: Select **upload**.

Type: Select **file**.

Value: Click **Upload** to upload the created **app.log** file.

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.

Figure 8-4 View logs


Code
Monitoring
Version
Aliases
Configuration
Metrics
Logs
FunctionGraph records all requests processed by your function and automatically stores the logs in LTS. Verify code by inserting custom logging statements. The following table lists the function logs. View more on LTS.
Request List
Request Logs
Enter a keyword.
Show All Versions and Aliases
Last hour
Last day
Last 3 days
Custom
Logs
Search
All
Full Screen
Download Log

Log Group functiongraph.l[redacted]31c00a4a1d35ea.0 Log Stream water-test[redacted]3eae87fabd2

```
2022-11-14T07:41:30Z Finish invoke request '3b7298b6-0a8a-4559-b306-4b20a701fedf', duration: 2.601ms, billing duration: 3ms, memory
2022-11-14T07:41:30Z Start invoke request '04d1b2c0-0da5-4082-84e9-913aea5833e9', version: latest
2022-11-14T07:41:30Z 04d1b2c0-0da5-4082-84e9-913aea5833e9 Function start run.
2022-11-14T07:41:30Z 04d1b2c0-0da5-4082-84e9-913aea5833e9 b'this is app log\nstart something\nrun\nstop all\n'
2022-11-14T07:41:30Z Finish invoke request '04d1b2c0-0da5-4082-84e9-913aea5833e9', duration: 1.185ms, billing duration: 2ms, memory
```

----End

9 Processing IoT Data

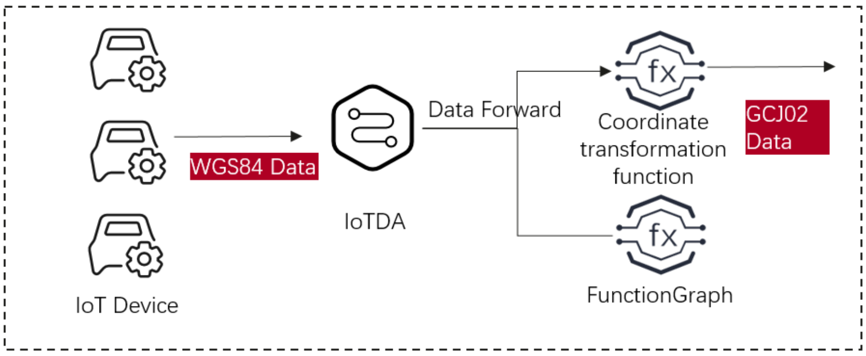
9.1 Introduction

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.

9.2 Preparation

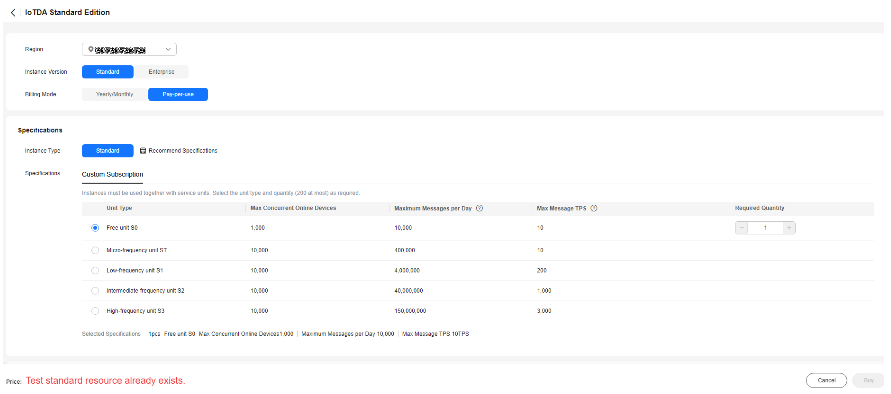
Before creating a forwarding rule, create an IoTDA instance as well as products and devices. In this best practice, we only create an instance for testing.

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 **Buy Instance**. The parameter configuration page is displayed. Set the parameters based on service requirements.

Figure 9-1 Enabling free standard edition



IoTDA Standard Edition

Region: [redacted]

Instance Version: [✓ Standard] Enterprise

Billing Mode: Yearly/Monthly [✓ Pay-per-use]

Specifications

Instance Type: [✓ Standard] Recommend Specifications

Specifications: Custom Subscription

Instances must be used together with service units. Select the unit type and quantity (200 at most) as required.

	Unit Type	Max Concurrent Online Devices	Maximum Messages per Day	Max Message TPS	Required Quantity
●	Free unit S0	1,000	10,000	10	1
○	Micro-frequency unit ST	10,000	400,000	10	
○	Low-frequency unit S1	10,000	4,000,000	200	
○	Intermediate-frequency unit S2	10,000	40,000,000	1,000	
○	High-frequency unit S3	10,000	150,000,000	3,000	

Selected Specifications 1pcs Free unit S0 Max Concurrent Online Devices1,000 | Maximum Messages per Day 10,000 | Max Message TPS 10TPS

Price: Test standard resource already exists.

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 Select **Create from scratch**. Set **Function Type** to **Event Function**, enter a name (for example, **iotdemo**) for **Function Name**, select a runtime (for example, **Python 3.9**), and click **Create Function**.

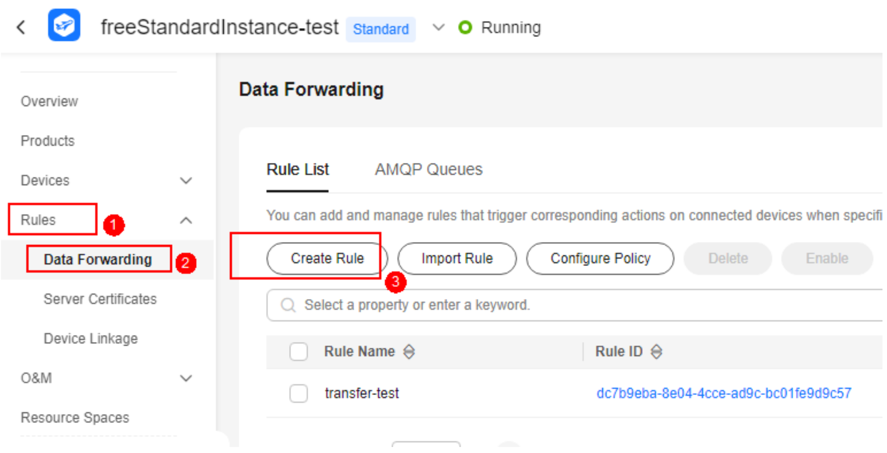
----End

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 navigation pane on the left, choose **IoT** > **IoT Device Access**. On the IoTDA console, click the instance name. On the displayed page, choose **Rules** > **Data Forwarding**, and click **Create Rule**.

Figure 9-2 Creating a rule



You can add and manage rules that trigger corresponding actions on connected devices when specifi

Rule Name	Rule ID
transfer-test	dc7b9eba-8e04-4cce-ad9c-bc01fe9d9c57

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

NOTE

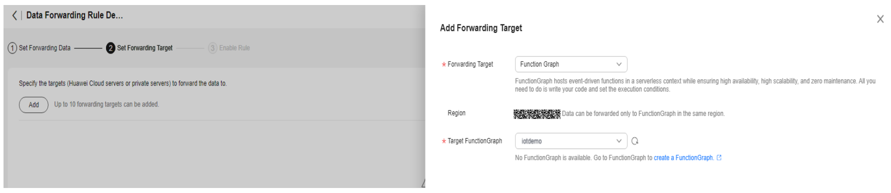
- **Rule Name**: Enter a custom rule name.
- **Data Source**: select **Device message**.
- **Trigger**: select device message reporting.
- **Resource Space**: Retain the default value.

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**.

Figure 9-3 Adding a forwarding target



Specify the targets (Huawei Cloud servers or private servers) to forward the data to.

Up to 10 forwarding targets can be added.

* Forwarding Target: Function Graph

FunctionGraph hosts event-driven functions in a serverless context while ensuring high availability, high scalability, and zero maintenance. All you need to do is write your code and set the execution conditions.

Region: [redacted] Data can be forwarded only to FunctionGraph in the same region.

* Target FunctionGraph: iotdemo

No FunctionGraph is available. Go to FunctionGraph to create a FunctionGraph.

Step 6 Start the rule.

----End

9.3 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.

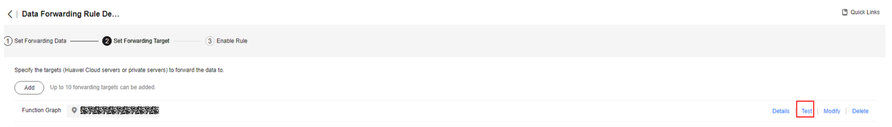
```
 # -*- coding:utf-8 -*-
import json
import math
from math import pi

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 and click an instance name. 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 9-4 Testing the forwarding rule



Data Forwarding Rule De...

Quick Links

1 Set Forwarding Data — 2 Set Forwarding Target — 3 Enable Rule

Specify the targets (Huawei Cloud servers or private servers) to forward the data to.

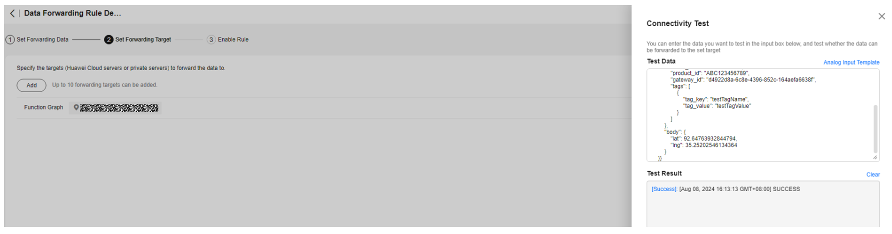
Add   Up to 10 forwarding targets can be added.

Function Graph [redacted]   Details | Test | Modify | Delete

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
        }
    }
}
```

Figure 9-5 Connectivity test result



Data Forwarding Rule De...

1 Set Forwarding Data — 2 Set Forwarding Target — 3 Enable Rule

Specify the targets (Huawei Cloud servers or private servers) to forward the data to.

Add   Up to 10 forwarding targets can be added.

Function Graph [redacted]

Connectivity Test

You can enter the data you want to test in the input box below, and test whether the data can be forwarded to the set target

Test Data   Analog Input Template

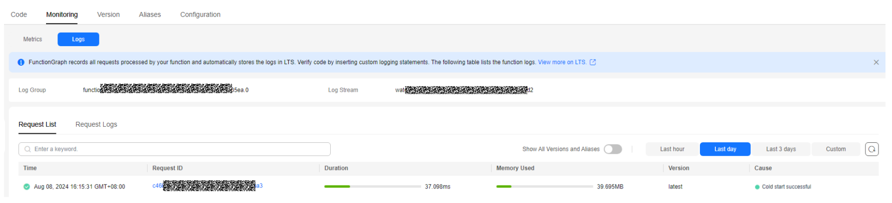
```
            "product_id": "ABC123456789",
            "gateway_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
            "tags": [
                {
                    "tag_key": "testTagName",
                    "tag_value": "testTagValue"
                }
            ]
        },
        "body": {
            "lat": 92.64763932844794,
            "lng": 35.25202546134364
        }
    }}
```

Test Result   Clear

```
[Success]: [Aug 08, 2024 16:13:13 GMT+08:00] SUCCESS
```

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

Figure 9-6 Viewing logs



Code   Monitoring   Version   Aliases   Configuration

Metrics   Logs

FunctionGraph records all requests processed by your function and automatically stores the logs in LTS. Verify code by inserting custom logging statements. The following table lists the function logs. View more on LTS.

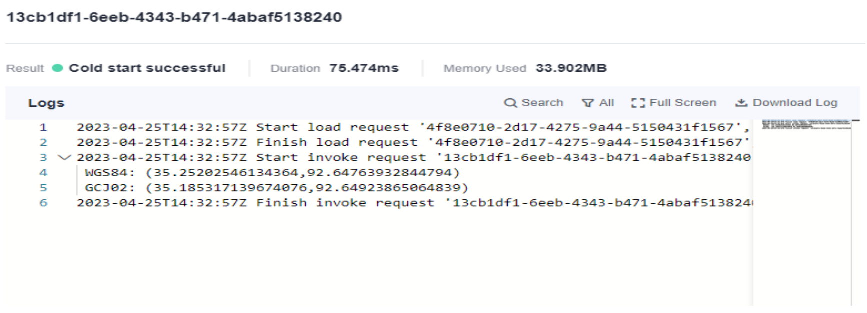
Log Group   functio[redacted]5ea.0   Log Stream   wat[redacted]d2

Request List   Request Logs

Enter a keyword.   Show All Versions and Aliases   Last hour   Last day   Last 3 days   Custom

Time	Request ID	Duration	Memory Used	Version	Cause
Aug 08, 2024 16:15:31 GMT+08:00	c46[redacted]a3	37.098ms	39.695MB	latest	Cold start successful

Figure 9-7 Viewing request details


13cb1df1-6eeb-4343-b471-4abaf5138240

Result ● Cold start successful | Duration 75.474ms | Memory Used 33.902MB

```
2023-04-25T14:32:57Z Start load request '4f8e0710-2d17-4275-9a44-5150431f1567',
2023-04-25T14:32:57Z Finish load request '4f8e0710-2d17-4275-9a44-5150431f1567'
2023-04-25T14:32:57Z Start invoke request '13cb1df1-6eeb-4343-b471-4abaf5138240
 WGS84: (35.25202546134364,92.64763932844794)
 GCJ02: (35.185317139674076,92.64923865064839)
2023-04-25T14:32:57Z Finish invoke request '13cb1df1-6eeb-4343-b471-4abaf513824
```

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

----End

10 Workflow + Function: Automatically Processing Data in OBS

10.1 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**)

10.2 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 a custom bucket name, for example, **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 an agency name, for example, **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 **OBS Administrator**.

Step 2 Click **Next**, select an authorization scope that meets your service requirements, and click **OK**.

----End

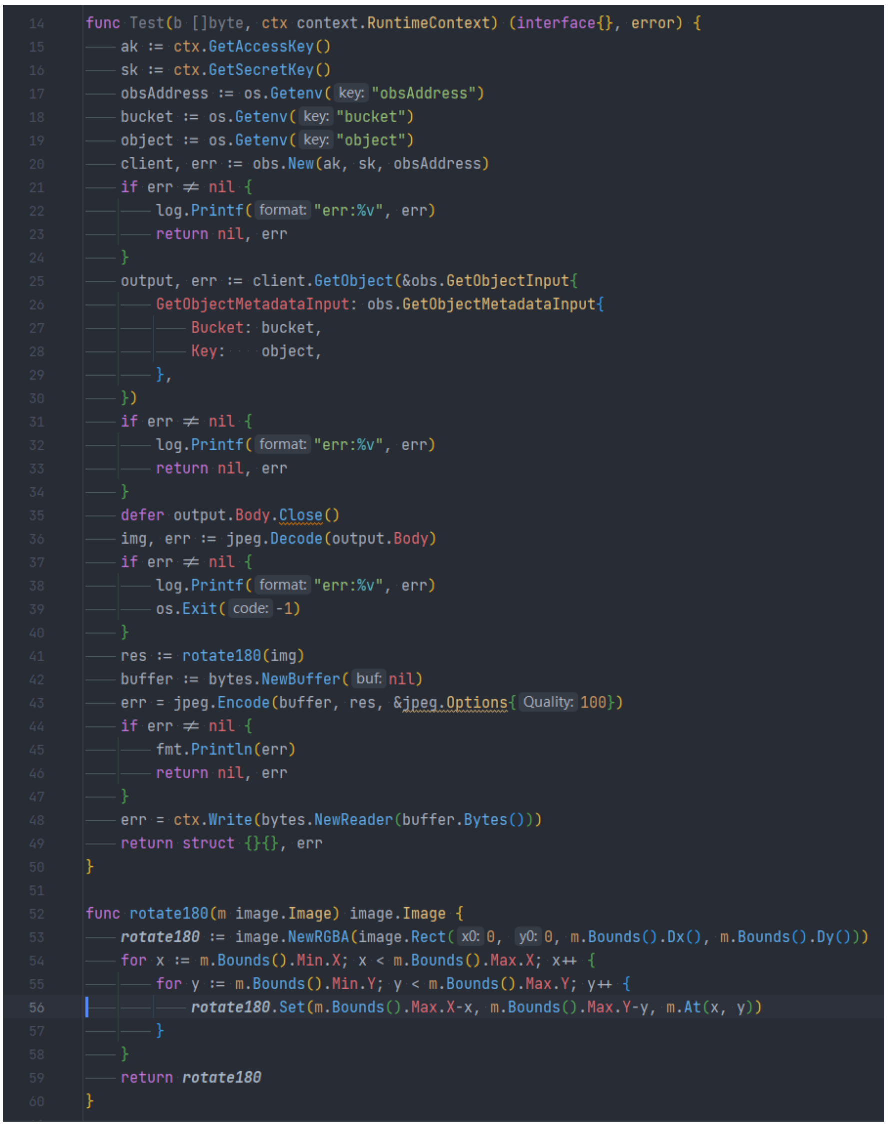
10.3 Building a Program

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 10-1** shows the sample code directory. The service code is not described.

Figure 10-1 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 Type**, select **Event Function**.
- For **Function Name**: Enter a function name, for example, **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.

  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 10-1 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**.

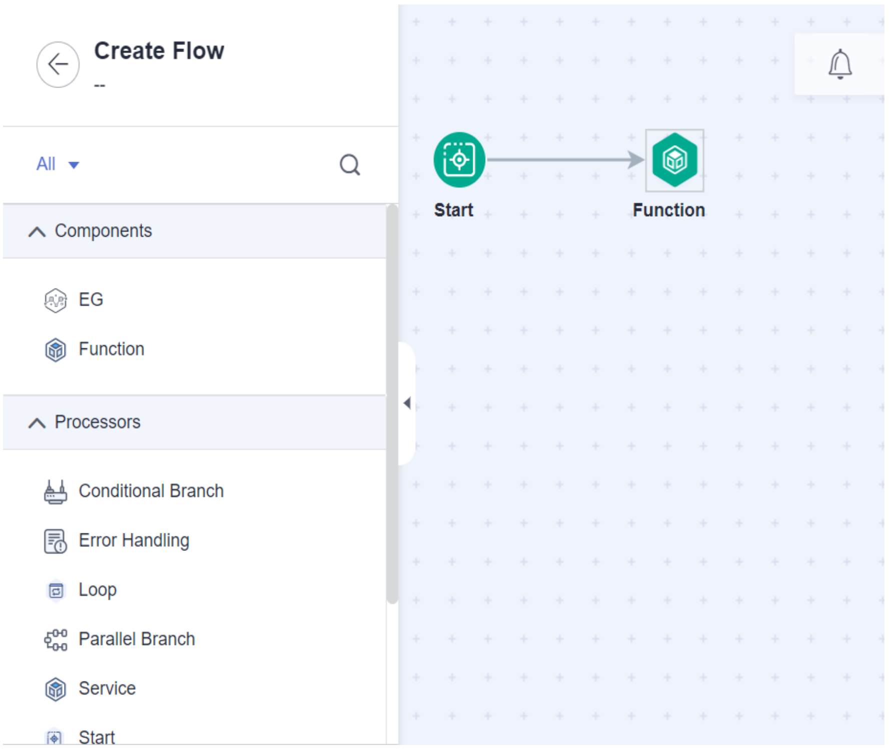
---- End

Creating a Flow

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

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

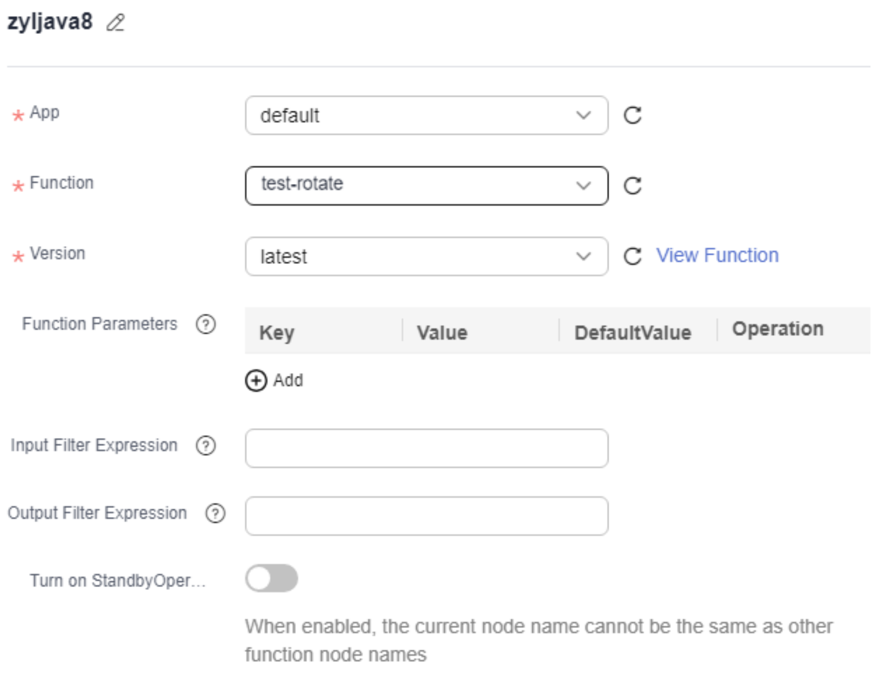
Figure 10-2 Creating an express flow



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 10-3 Configuring function node



zyljava8

* App	default
* Function	test-rotate
* Version	latest
Function Parameters	
Input Filter Expression	
Output Filter Expression	
Turn on StandbyOper...	

Key	Value	DefaultValue	Operation

When enabled, the current node name cannot be the same as other function node names

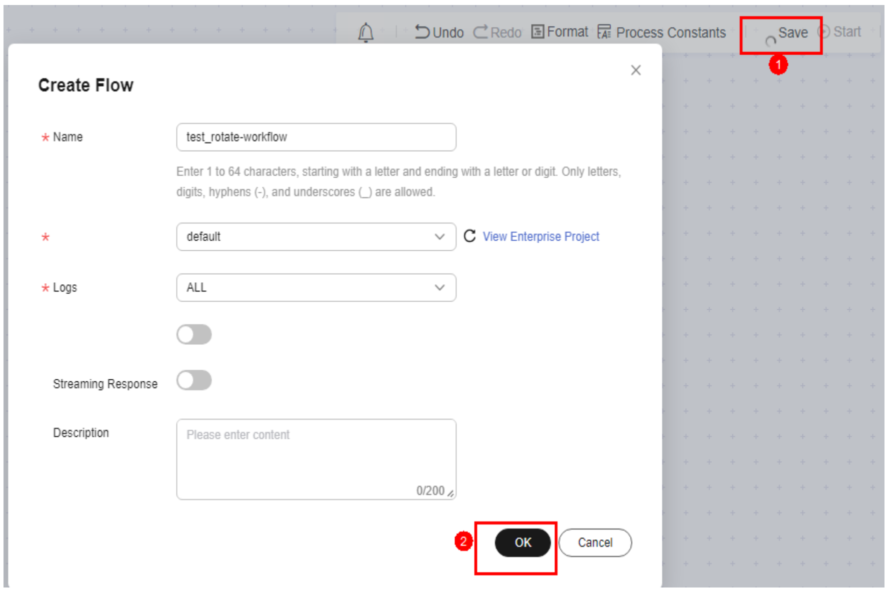
Click **OK** after the parameters are configured.

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 10-4 Saving a flow



Create Flow

* Name	test_rotate-workflow
*	default
* Logs	ALL
Streaming Response	
Description	Please enter content

Enter 1 to 64 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.

0/200

----End

10.4 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 10-5**. **Figure 10-6** shows the uploaded image.

Figure 10-5 Example

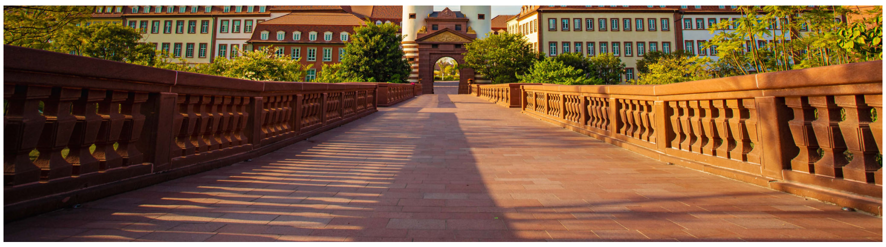
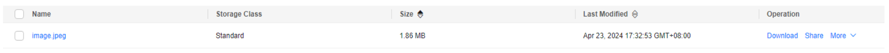
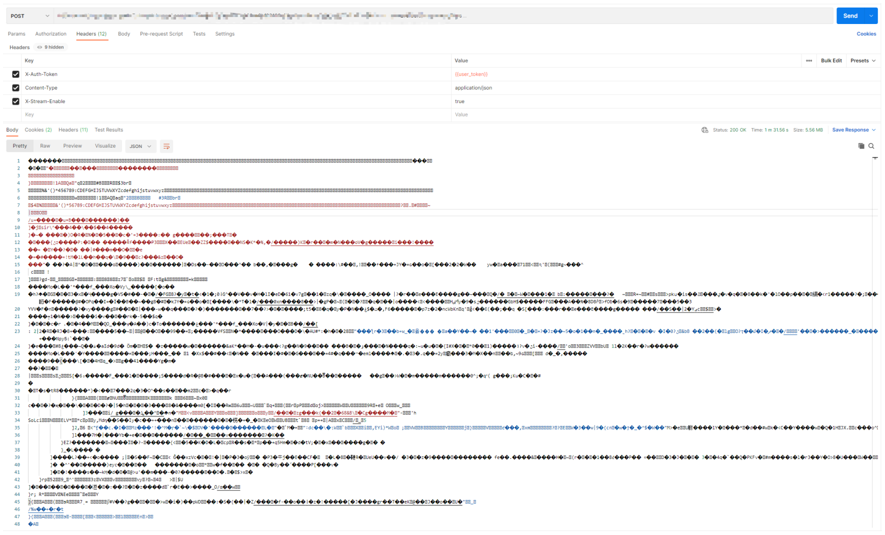


Figure 10-6 Uploaded image



Name	Storage Class	Size	Last Modified	Operation
image.jpeg	Standard	1.86 MB	Apr 23, 2024 17:32:53 GMT+08:00	Download Share More

Using Postman to Trigger the Function Flow



POST [redacted] Send

Params Authorization Headers (12) Body Pre-request Script Tests Settings Cookies

Headers 9 hidden

	Key	Value
✓	X-Auth-Token	{{user_token}}
✓	Content-Type	application/json
✓	X-Stream-Enable	true
	Key	Value

Body Cookies (2) Headers (11) Test Results — Status: 200 OK Time: 1 m 31.56 s Size: 5.56 MB Save Response

Pretty Raw Preview Visualize JSON

```
[illegible]
```

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

11 Filtering Logs in Real Time by Using FunctionGraph and LTS

11.1 Introduction

This chapter elaborates the following aspects of the practice:

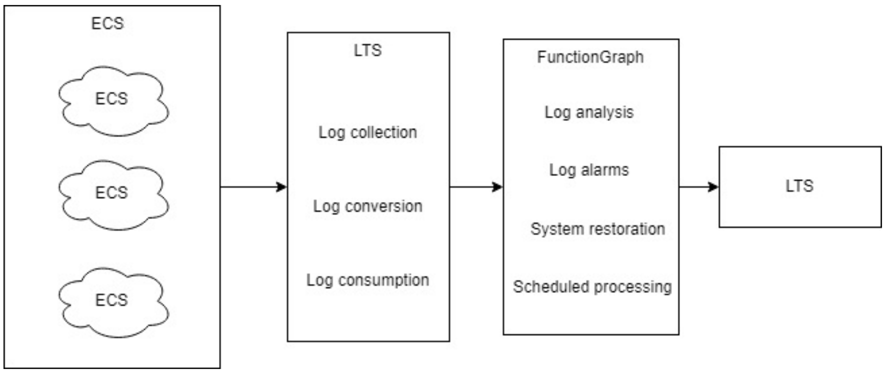
- Scenario and benefits
- Preparation
- Building a program
- Event source
- Processing results
- Extended applications

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 11-1** shows this process.

Figure 11-1 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.

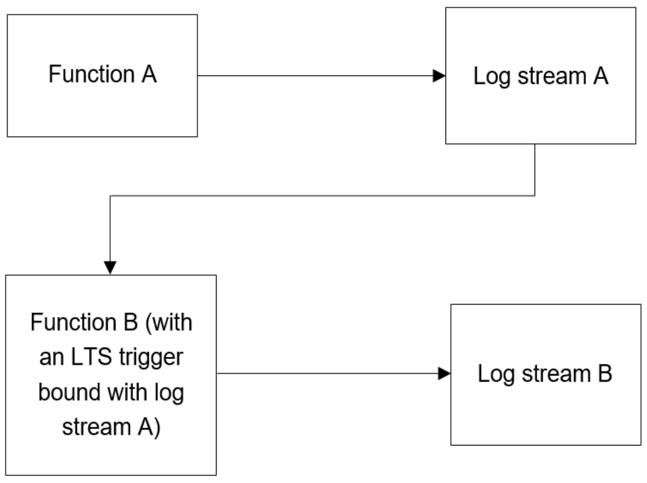
11.2 Preparation

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 11-2 Flowchart

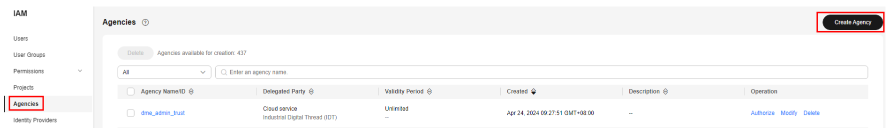


Creating an Agency

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 11-3**.

Figure 11-3 Creating an agency



Agencies available for creation: 437

	Agency Name/ID	Delegated Party	Validity Period	Created	Description	Operation
	dme_admin_trust	Cloud service Industrial Digital Thread (IDT)	Unlimited --	Apr 24, 2024 09:27:51 GMT+08:00	--	Authorize Modify Delete

Step 3 Configure the agency.

- **Agency Name**: Enter an agency name, for example, **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** in the search box on the right and select it.

NOTE

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

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

----End

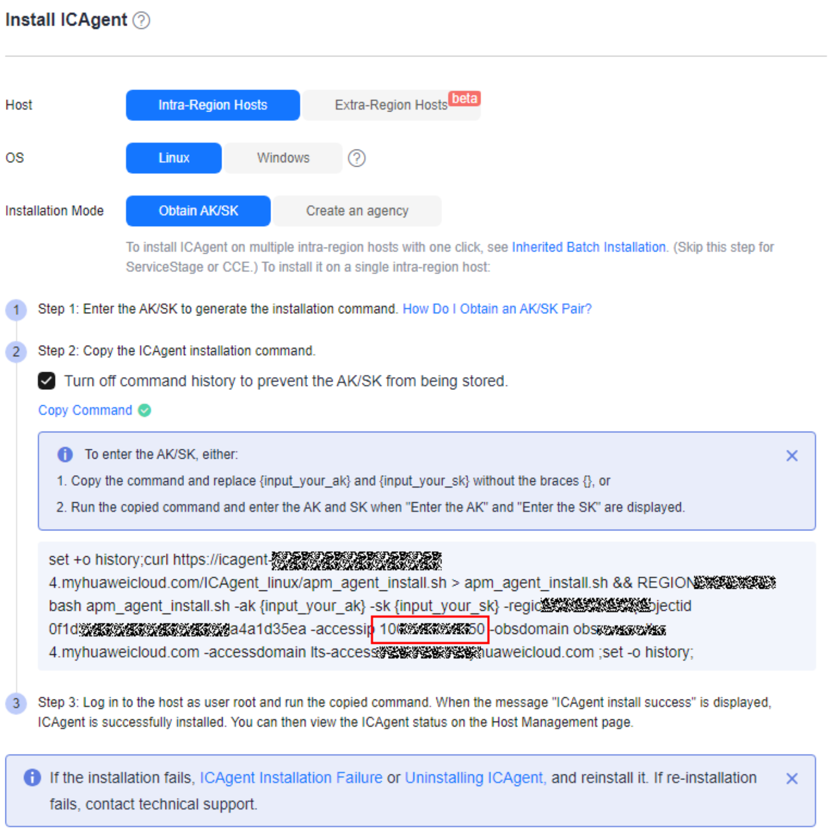
11.3 Building a Program

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 on the left, choose **Host Management** > **Hosts**.
2. In the upper right corner of the page, click **Install ICAgent**.
3. Obtain the access point IP address in the **Install ICAgent** window.

Figure 11-4 Access point IP address



Install ICAgent

Host: Intra-Region Hosts | Extra-Region Hosts beta

OS: Linux | Windows

Installation Mode: Obtain AK/SK | Create an agency

To install ICAgent on multiple intra-region hosts with one click, see Inherited Batch Installation. (Skip this step for ServiceStage or CCE.) To install it on a single intra-region host:

1 Step 1: Enter the AK/SK to generate the installation command. How Do I Obtain an AK/SK Pair?

2 Step 2: Copy the ICAgent installation command.

☑ Turn off command history to prevent the AK/SK from being stored.

Copy Command

To enter the AK/SK, either:

1. Copy the command and replace {input_your_ak} and {input_your_sk} without the braces {}, or
2. Run the copied command and enter the AK and SK when "Enter the AK" and "Enter the SK" are displayed.

```
set +o history;curl https://icagent-[redacted]
4.myhuaweicloud.com/ICAgent_linux/apm_agent_install.sh > apm_agent_install.sh && REGION[redacted]
bash apm_agent_install.sh -ak {input_your_ak} -sk {input_your_sk} -regio[redacted]bjectid
0f1d[redacted]a4a1d35ea -accessip 10[redacted]50 -obsdomain obs[redacted]
4.myhuaweicloud.com -accessdomain lts-access[redacted]uaweicloud.com ;set -o history;
```

3 Step 3: Log in to the host as user root and run the copied command. When the message "ICAgent install success" is displayed, ICAgent is successfully installed. You can then view the ICAgent status on the Host Management page.

If the installation fails, ICAgent Installation Failure or Uninstalling ICAgent, and reinstall it. If re-installation fails, contact technical support.

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**.

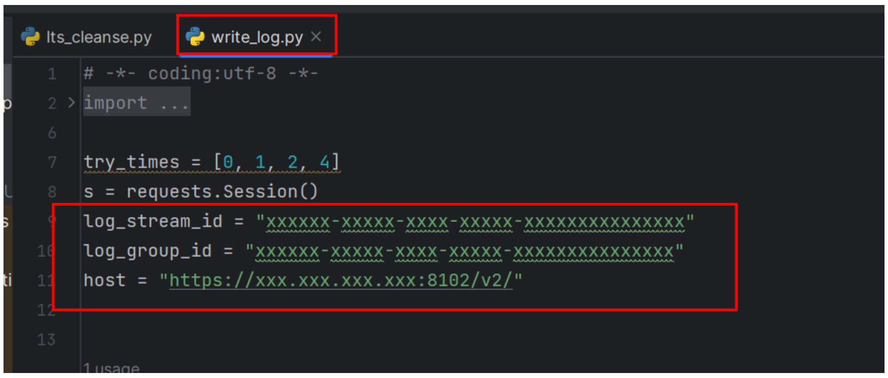
3. Create the LTS dependency required by function **B**. For details, see **How Do I Create a Dependency on the FunctionGraph Console?** and **How Do I Add a Dependency to a Function?** You can run the **pip install huaweicloudsdklts** command to create the dependency. The sample code contains the **huaweicloudsdklts** 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 11-5**.

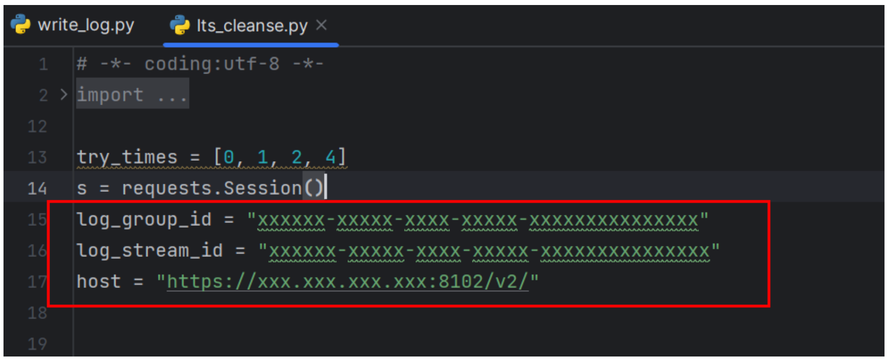
Figure 11-5 write_log.py



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

try_times = [0, 1, 2, 4]
s = requests.Session()
log_stream_id = "xxxxxx-xxxxx-xxxx-xxxxx-xxxxxxxxxxxxxxx"
log_group_id = "xxxxxx-xxxxx-xxxx-xxxxx-xxxxxxxxxxxxxxx"
host = "https://xxx.xxx.xxx.xxx:8102/v2/"
```

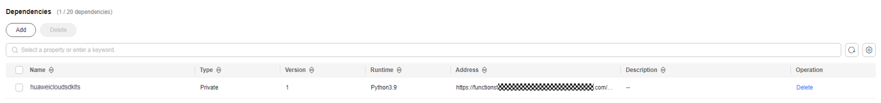
Create function **B**. For the sample code of this function, see the **lts_cleanse.py** file. In the code of function **B**, replace **host**, **log_group_id**, and **log_stream_id** with the access point and the IDs of log group **test-1121** and log stream **test-1121**, and add the **huaweicloudsdklts** dependency to this function, as shown in **Figure 11-6** and **Figure 11-7**.

Figure 11-6 lts_cleanse.py



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

try_times = [0, 1, 2, 4]
s = requests.Session()
log_group_id = "xxxxxx-xxxxx-xxxx-xxxxx-xxxxxxxxxxxxxxx"
log_stream_id = "xxxxxx-xxxxx-xxxx-xxxxx-xxxxxxxxxxxxxxx"
host = "https://xxx.xxx.xxx.xxx:8102/v2/"
```

Figure 11-7 Adding a dependency for function B



Dependencies (1 / 20 dependencies)
Add
Delete
Select a property or enter a keyword.

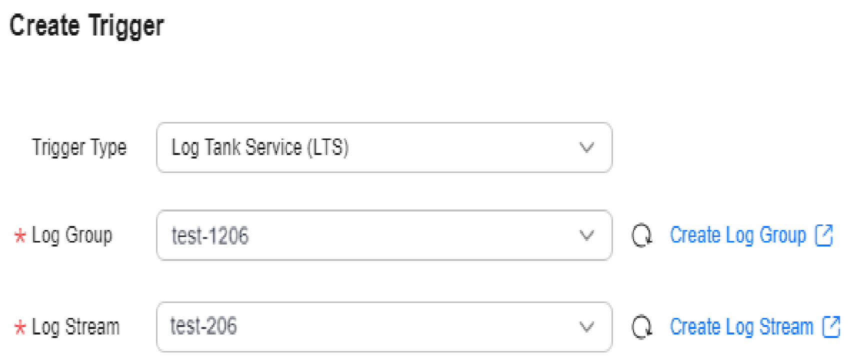
Name	Type	Version	Runtime	Address	Description	Operation
huaweicloudsdklts	Private	1	Python3.9	https://functionst[redacted].com/...	--	Delete

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.

11.4 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 11-8**.

Figure 11-8 Creating an LTS trigger


Create Trigger

Trigger Type: Log Tank Service (LTS)

* Log Group: test-1206

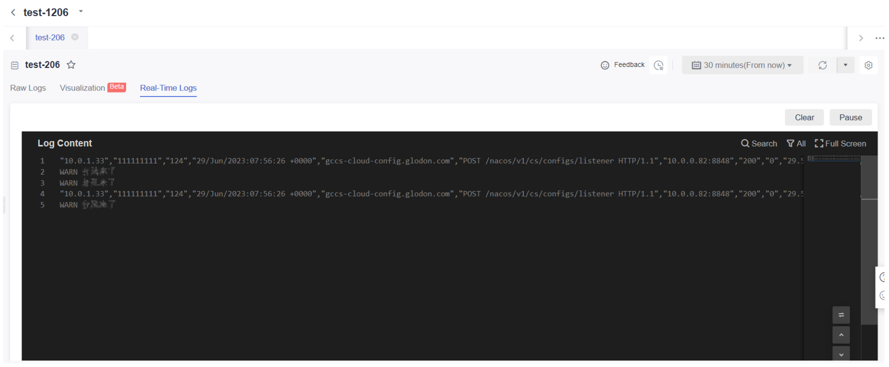
* Log Stream: test-206

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.

11.5 Processing Results

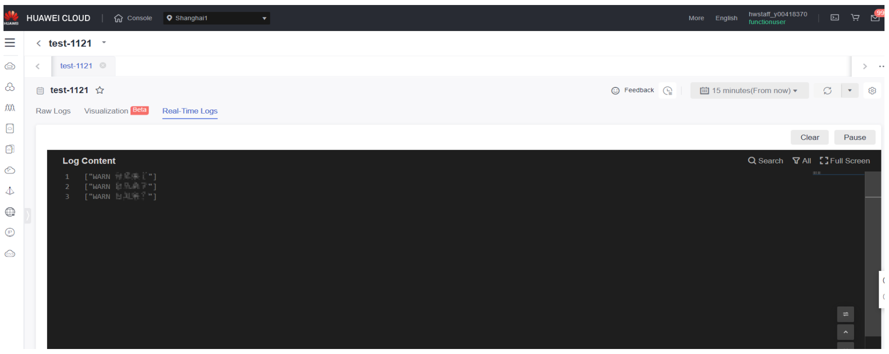
Filter alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR**, and transfer them to a specified log stream. **Figure 11-9** and **Figure 11-10** show the real-time logs before and after filtering, respectively.

Figure 11-9 Logs before filtering



```
"10.0.1.33","111111111","124","29/Jun/2023:07:56:26 +0000","gccs-cloud-config.glodon.com","POST /nacos/v1/cs/configs/listener HTTP/1.1","10.0.0.82:8848","200","0","29.5
WARN [redacted]
WARN [redacted]
"10.0.1.33","111111111","124","29/Jun/2023:07:56:26 +0000","gccs-cloud-config.glodon.com","POST /nacos/v1/cs/configs/listener HTTP/1.1","10.0.0.82:8848","200","0","29.5
WARN [redacted]
```

Figure 11-10 Logs after filtering



```
["WARN [redacted]"]
["WARN [redacted]"]
["WARN [redacted]"]
```

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

Figure 11-11 Function metrics (1)

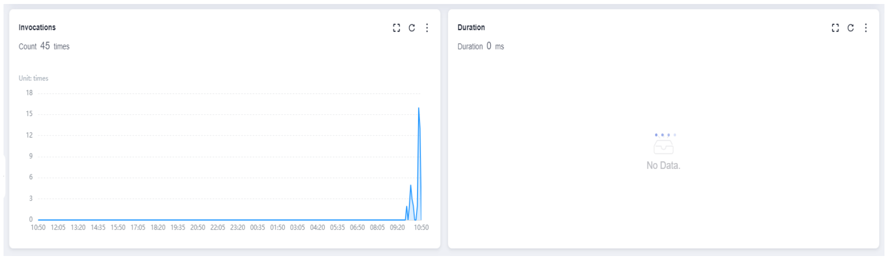


Figure 11-12 Function metrics (2)

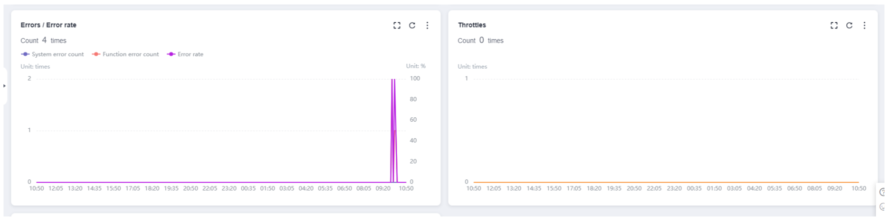
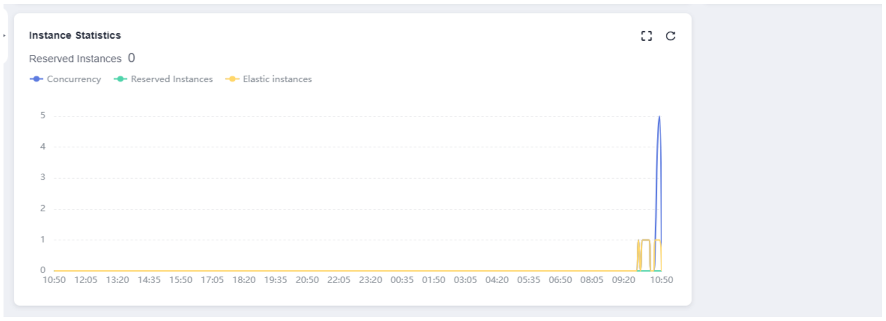


Figure 11-13 Function metrics (3)



11.6 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.

12 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 FunctionGraph.

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
```

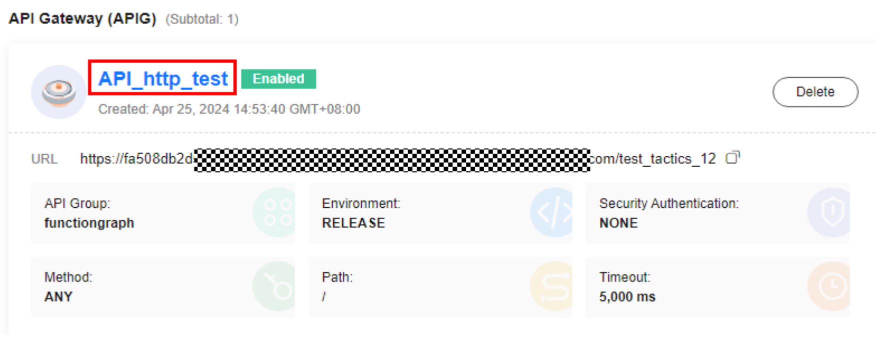
Creating an HTTP function and uploading code

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 12-1 APIG trigger



API Gateway (APIG) (Subtotal: 1)

API_http_test Enabled
Created: Apr 25, 2024 14:53:40 GMT+08:00

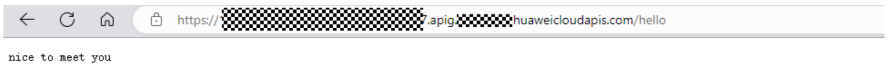
URL https://fa508db2d[redacted]com/test_tactics_12

API Group: functiongraph
Environment: RELEASE
Security Authentication: NONE
Method: ANY
Path: /
Timeout: 5,000 ms

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 12-2 Request result



https://[redacted].apig[redacted]huaweicloudapis.com/hello

```
nice to meet you
```

13 Using FunctionGraph HTTP Functions to Process gRPC Requests

Introduction

This chapter describes how to use gRPC and process gRPC requests in FunctionGraph.

This chapter uses **example/helloworld** in the **gRPC example code** project as an example to describe how to process gRPC requests with HTTP functions in FunctionGraph. Since HTTP functions do not directly support Go code deployment, this chapter provides an example of using binary conversion to deploy a Go program on FunctionGraph.

NOTE

- This feature is supported only in the **LA-Santiago** region.
- By default, you do not have gRPC permissions. To use gRPC, **submit a service ticket** to add your account to the whitelist.

Procedure

1. **Building a code package**

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

```
// Package main implements a grpc_server for Greeter service.
package main

import (
    "context"
    "flag"
    "fmt"
    "log"
    "net"

    pb "helloworld/helloworld"

    "google.golang.org/grpc"
)

var (
    port = flag.Int("port", 8000, "The grpc_server port")
)

// server is used to implement helloworld.GreeterServer.
```

```
type server struct {
    pb.UnimplementedGreeterServer
}

// SayHello implements helloworld.GreeterServer
func (s *server) SayHello(ctx context.Context, in *pb.HelloRequest) (*pb.HelloReply, error) {
    log.Printf("Received: %v", in.GetName())
    return &pb.HelloReply{Message: "Hello " + in.GetName()}, nil
}

func main() {
    flag.Parse()
    lis, err := net.Listen("tcp", fmt.Sprintf("127.0.0.1:%d", *port))
    if err != nil {
       log.Fatalf("failed to listen: %v", err)
    }
    s := grpc.NewServer()
    pb.RegisterGreeterServer(s, &server{})
    log.Printf("grpc_server listening at %v", lis.Addr())
    if err := s.Serve(lis); err != nil {
       log.Fatalf("failed to serve: %v", err)
    }
 }
# bootstrap
$RUNTIME_CODE_ROOT/grpc-server
```

In **main.go**, a gRPC server is started using port **8000** and **helloworld.GreeterServer** is registered. When the service is invoked, **Hello XXX** will be returned.

2. **Compiling and packaging**
   a. On the Linux server, compile the preceding code using the **go build -o grpc-server main.go** command. Then, compress **grpc-server** and **bootstrap** into a ZIP package named **xxx.zip**.
   b. 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 grpc-server main.go

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

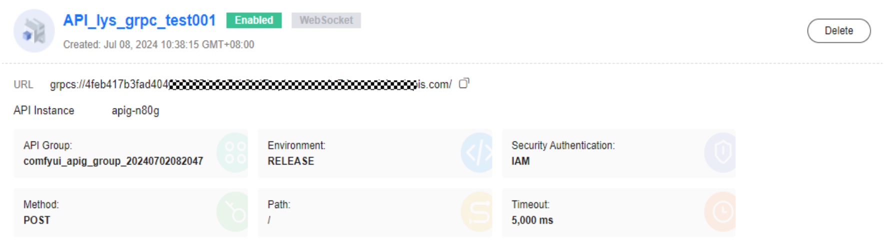
3. **Creating an HTTP function and uploading code**

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

4. **Creating an APIG trigger**

   Create an APIG trigger. For details, see **Using an APIG Trigger**. You are advised to set **Request Protocol** to **gRPC** and **Security Authentication** to **None** to simplify the debugging process.

Figure 13-1 APIG trigger



API_lys_grpc_test001 Enabled WebSocket
Created: Jul 08, 2024 10:38:15 GMT+08:00

URL grpcs://4feb417b3fad404[redacted]is.com/

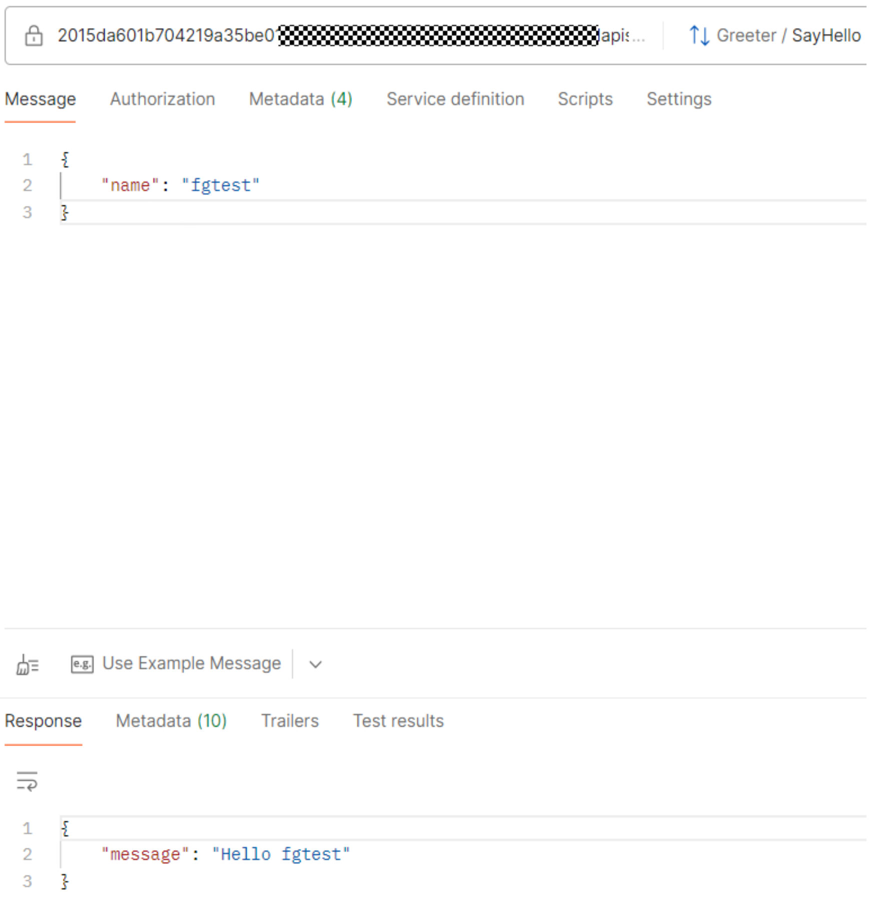
API Instance apig-n80g

API Group: comfyui_apig_group_20240702082047
Environment: RELEASE
Security Authentication: IAM
Method: POST
Path: /
Timeout: 5,000 ms

5. **Invocation test**

   Use Postman to debug gRPC requests.

Figure 13-2 gRPC request result



2015da601b704219a35be0[redacted]apis... Greeter / SayHello

```
{
    "name": "fgtest"
}
```

```
{
    "message": "Hello fgtest"
}
```

14 Cold Start Optimization Practices

The serverless architecture features pay-per-use, auto scaling, and complexity shielding, making it a new paradigm of next-generation cloud computing. However, in real-time scenarios, cold start poses a significant challenge. When building web services with serverless, if the cold start and initialization time exceed 5 seconds, it can greatly impact user experience. Therefore, accelerating cold start and improving the user experience is a pressing issue when constructing serverless architectures.

The lifecycle of a serverless instance consists of three phases:

- **Initialization**: FunctionGraph attempts to reuse a previous execution environment, or if none is available, it creates resources, downloads function code, initializes extensions and runtime, and runs initialization code (non-main program code).
- **Execution**: In this phase, the instance starts to execute the function after receiving an event, and waits for the next event after the function completes.
- **Shutdown**: This phase starts if the function does not receive any calls within a period of time. In this phase, FunctionGraph closes the runtime, then sends a shutdown event to each extension, and deletes the environment.

When FunctionGraph is triggered, if no activated function instance is available, the function code is downloaded and an execution environment is created. The period from the time when a function is triggered to the time when a new FunctionGraph environment is created is called cold start. In the serverless architecture, the cold start cannot be avoided.

Currently, FunctionGraph has optimized the cold start on the system side. For details about the cold start on the user side, see the following solutions.

Memory

Given a fixed level of request concurrency, higher function memory leads to better cold start performance with more CPU resources allocated.

Cold Start with Snapshot

Cold start is quite prominent in Java applications. Huawei Cloud FunctionGraph has proposed a process snapshot–based cold start acceleration solution, which helps you break through the performance bottleneck while involving zero or few code changes. In this solution, the execution environment is restored from a snapshot captured after initialization, avoiding complex framework startup and service initialization. The startup latency of Java applications is significantly reduced, and the performance is improved by over 90%.

You can use a Java function to enable snapshot-based cold start. For details, see **Configuring Snapshot-based Cold Start**. FunctionGraph executes the initialization code of the function, captures a snapshot of the initialization context, and then caches the snapshot after encryption. When the function is invoked and a cold start is triggered for scale-out, the execution environment is restored from the snapshot instead of an initialization process.

Code Simplification and Image Downsizing

FunctionGraph downloads function code during cold start, but larger code packages or custom images can prolong download and cold start time. Optimize your application by removing unnecessary code (using commands like **npm prune** in Node.js and **autoflake** in Python) and third-party library dependencies (test case source code, useless binary files, and data files) to speed up the download and decompression process.

Public Dependencies

When developing applications, especially in Python, third-party libraries are often included. Large dependencies can slow down cold start as they need to be downloaded. FunctionGraph offers both public and private dependency modes. Public dependencies are recommended because they are pre-downloaded to execution nodes to save time.

Warming

When an event triggers a function, if a function instance in an active state can be called, cold start can be avoided, and a response time can be reduced. You can use either of the following warming methods:

- Use timer triggers. For details, see **Using a Timer Trigger**.
- Use reserved instances. For details, see **Reserved Instance Management**.