

08 Best Practices (Official)

Issue 01
Date 2025-02-28



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2025. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: <https://www.huaweicloud.com/intl/en-us/>

Contents

1 FunctionGraph Best Practices.....	1
2 Processing DIS Data.....	3
2.1 Introduction.....	3
2.2 Preparation.....	3
2.3 Building a Program.....	5
2.4 Adding an Event Source.....	11
2.5 Processing Data.....	12
3 Integrating with LTS to Analyze Logs in Real Time.....	14
3.1 Introduction.....	14
3.2 Preparation.....	15
3.3 Building a Program.....	17
3.4 Adding an Event Source.....	18
3.5 Processing Log Data.....	18
3.6 Other Application Scenarios.....	19
4 Integrating with CTS to Analyze Login/Logout Security.....	20
4.1 Introduction.....	20
4.2 Preparation.....	21
4.3 Building a Program.....	22
4.4 Adding an Event Source.....	22
4.5 Processing Operation Records.....	23
5 Periodically Starting or Stopping Huawei Cloud ECSs.....	24
6 Building an HTTP Function with Spring Boot.....	28
7 Creating a FunctionGraph Backend API That Uses a Custom Authorizer.....	33
7.1 Introduction.....	33
7.2 Resource Planning.....	34
7.3 Building a Program.....	34
7.4 Adding an Event Source.....	40
7.5 Debugging and Calling the API.....	41
8 Uploading Files with FunctionGraph and APIG.....	43
8.1 Introduction.....	43

8.2 Resource Planning.....	43
8.3 Procedure.....	44
8.3.1 Node.js.....	44
8.3.2 Python.....	46
9 Processing IoT Data.....	49
9.1 Introduction.....	49
9.2 Preparation.....	50
9.3 Building a Program.....	52
10 Workflow + Function: Automatically Processing Data in OBS.....	55
10.1 Introduction.....	55
10.2 Preparation.....	56
10.3 Building a Program.....	57
10.4 Processing Images.....	62
11 Filtering Logs in Real Time by Using FunctionGraph and LTS.....	64
11.1 Introduction.....	64
11.2 Preparation.....	65
11.3 Building a Program.....	67
11.4 Adding an Event Source.....	69
11.5 Processing Results.....	69
11.6 Extended Applications.....	70
12 Deploying AI Drawing Stable-Diffusion in the Application Center.....	72
12.1 Introduction.....	72
12.2 Preparation.....	72
12.2.1 Overview.....	72
12.2.2 Creating a VPC and Subnet.....	73
12.2.3 Creating an SFS Turbo File System.....	73
12.2.4 Creating an Agency.....	74
12.2.5 Configuring Domain Name Resolution.....	76
12.3 Application Creation and Deployment.....	78
12.4 Application Use.....	80
12.5 Custom Models.....	81
12.5.1 Initializing a Model.....	81
12.5.2 Importing and Loading Models.....	83
12.6 Advanced Use.....	84
12.6.1 Using ECS as an NFS Server to Isolate Resources of Multiple Users.....	84
12.6.2 API Access.....	89
12.6.3 Enabling WebUI Authentication.....	90
12.6.4 Sharing Models and Plugins.....	90
12.6.5 Using a Dedicated APIG Trigger.....	92
12.7 Disclaimer.....	93
13 Building an HTTP Function with Go.....	94

14 Using FunctionGraph HTTP Functions to Process gRPC Requests.....	97
15 Cold Start Optimization Practices.....	100

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

[Introduction](#)

[Preparation](#)

[Building a Program](#)

[Adding an Event Source](#)

[Processing 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

The screenshot shows the 'Buy Cluster' configuration page. It includes the following fields and options:

- Region:** A dropdown menu with a selected region.
- AZ:** Radio buttons for AZ1, AZ2, AZ3, and AZ4, with AZ4 selected.
- Billing Mode:** Radio buttons for 'Pay-per-use' (selected) and 'Yearly/Monthly'.
- Name:** A text input field containing 'cloudtable-dis'.
- VPC:** A dropdown menu showing 'vpc-6413' with a 'View VPC' link.
- Subnet:** A dropdown menu showing 'subnet-...' with a 'Create Subnet' link.
- Security Group:** A dropdown menu showing 'cluster-...' with a 'View Security Group' link.
- Database Engine:** Radio buttons for 'HBase' (selected) and 'ClickHouse'.
- HBase Version:** A dropdown menu showing '2.4.14'.

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

Figure 2-2 Creating a cluster

The screenshot shows the 'Cluster Management' page with a table of clusters. The table has the following columns and data:

Cluster Name	Cluster Status	Task Status	D...	V...	Enterprise Pr...	Created	Billing Mode	Access Address (Internal)	Operation
cloudtable-ec3	Creating	-	HBase	2.4.14	Default	-	Pay-per-use	-	View More 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

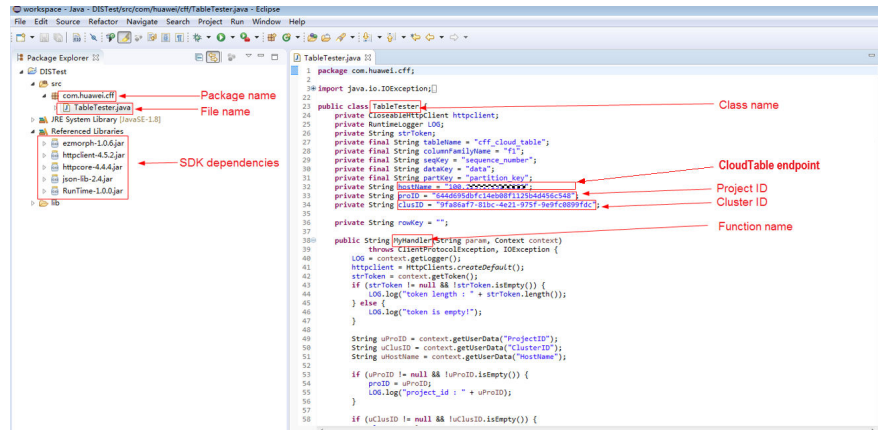
Download the [source code](#) and [program package](#) (including function dependencies) to create a function from scratch for converting DIS stream data formats.

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



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

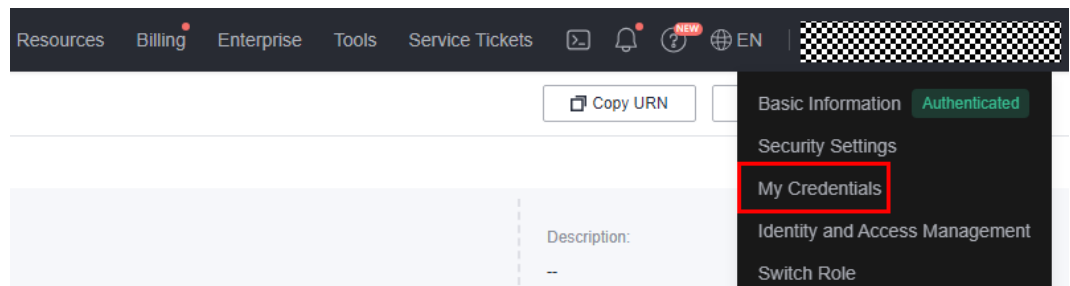
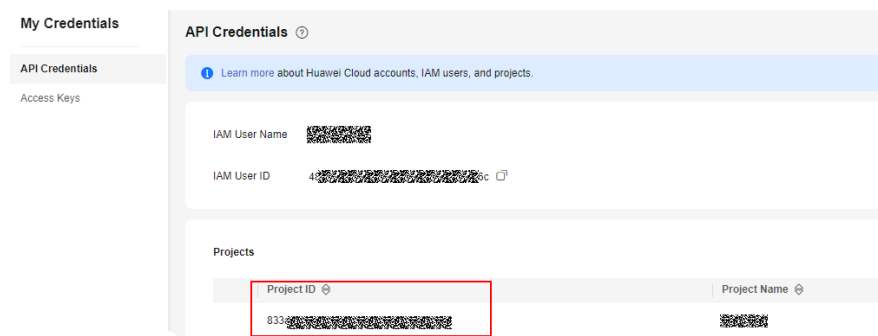


Figure 2-5 Project ID



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 **clouddtable-dis** created in [Creating a Cluster](#).

- On the **clouddata-dis** page that is displayed, find the cluster ID, as shown in **Figure 2-6**.

Figure 2-6 Cluster ID



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

Packaging the Code

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

Figure 2-7 Exporting the code

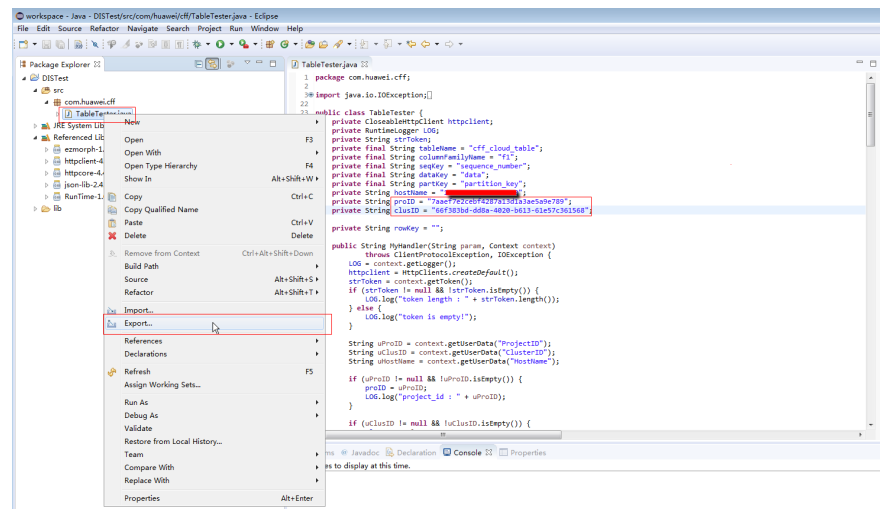


Figure 2-8 Selecting a file type

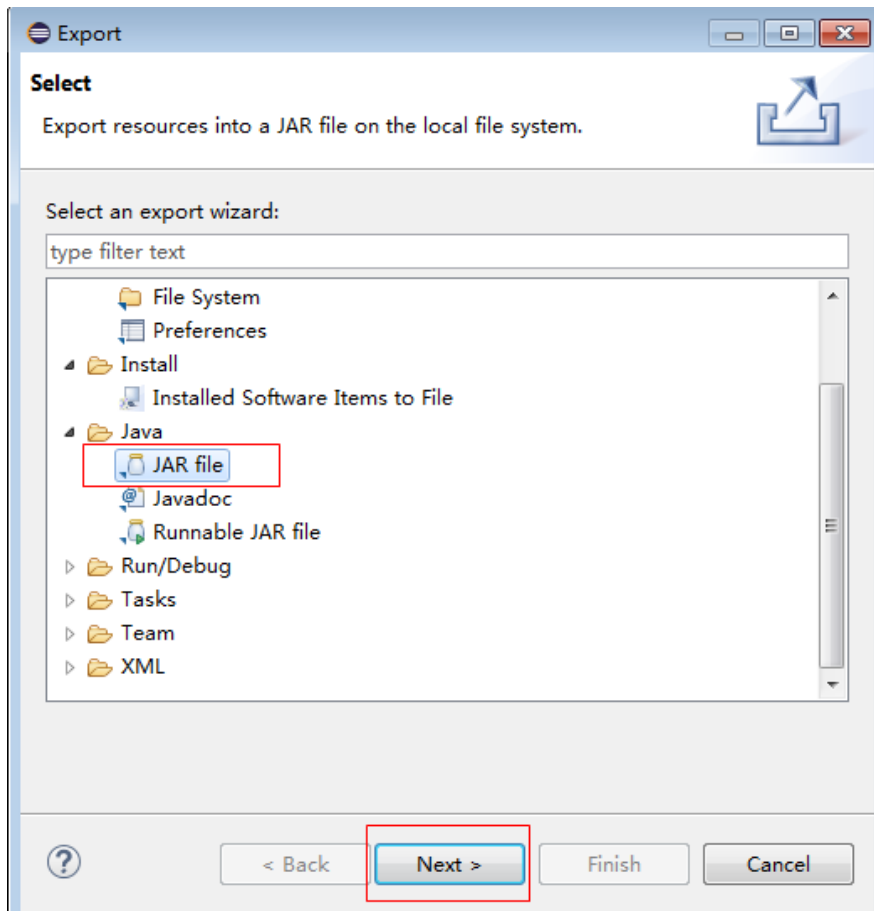
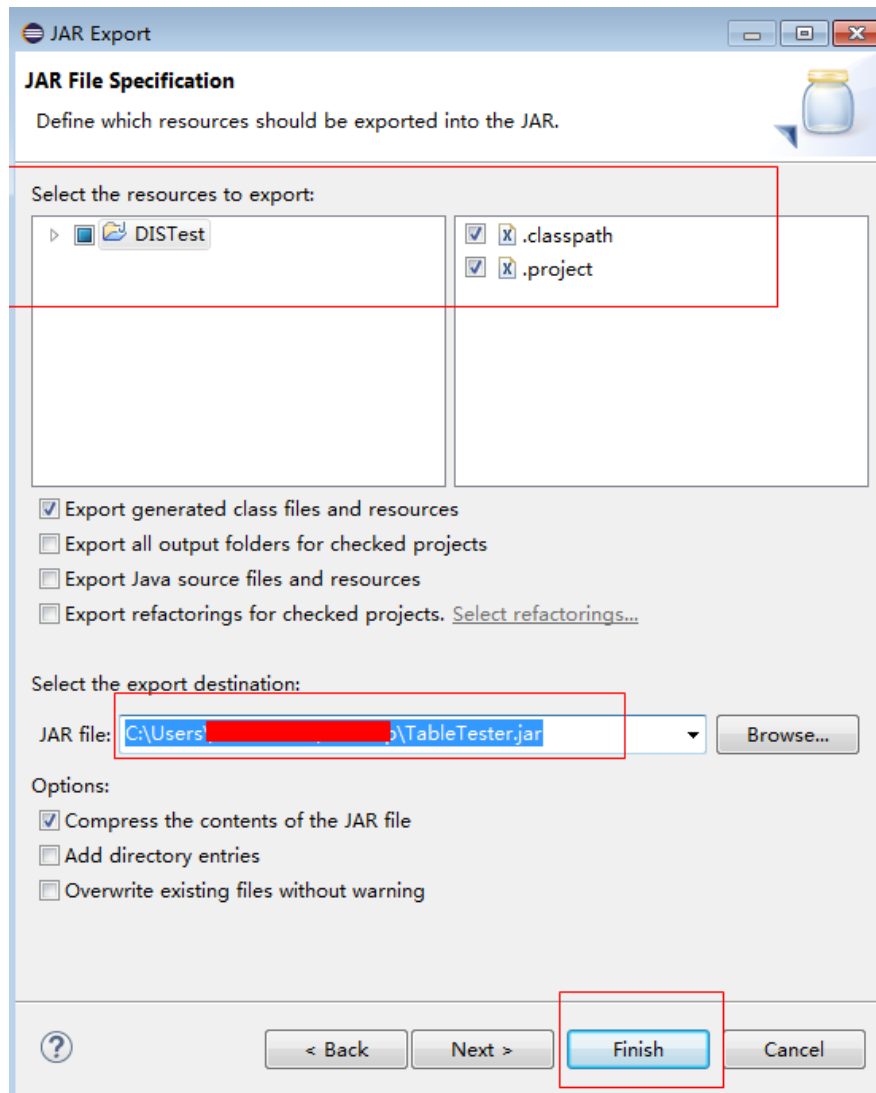


Figure 2-9 Publishing the code file

Package the function dependencies by performing the following steps:

1. **Download program package** `fss_examples_dis_cloudtable.zip`, and decompress it, as shown in [Figure 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

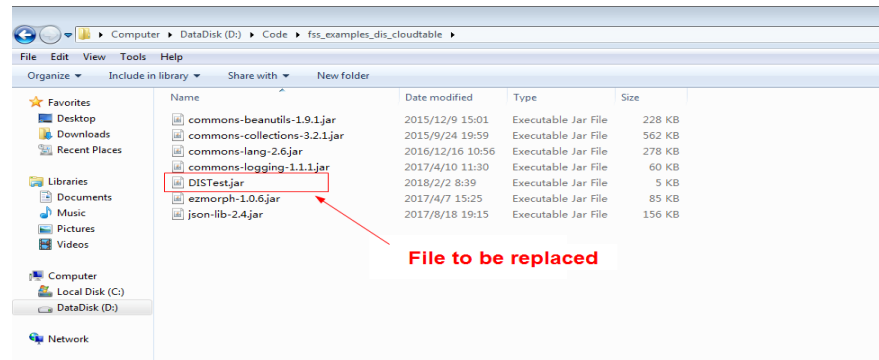


Figure 2-11 File directory after replacement

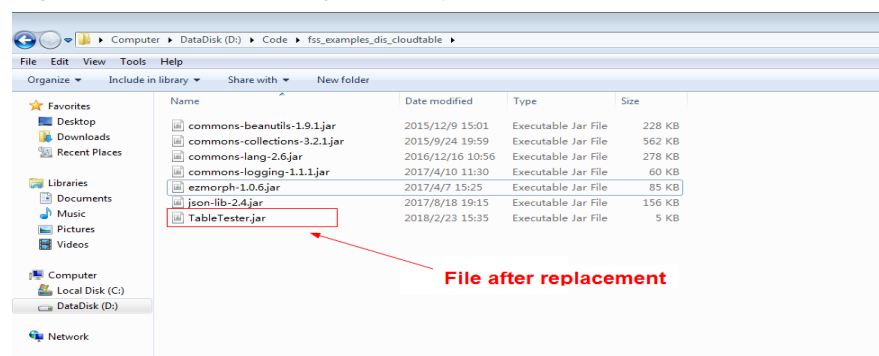
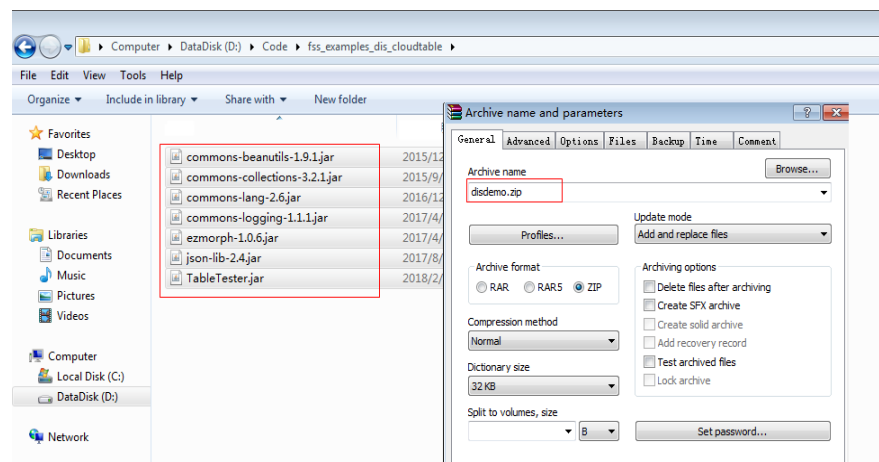


Figure 2-12 Packaging the files in ZIP format



Creating a Function

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

- Step 1** Log in to the [FunctionGraph console](#), and choose **Functions > Function List** in the navigation pane.
- Step 2** Click **Create Function**.
- Step 3** 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**.

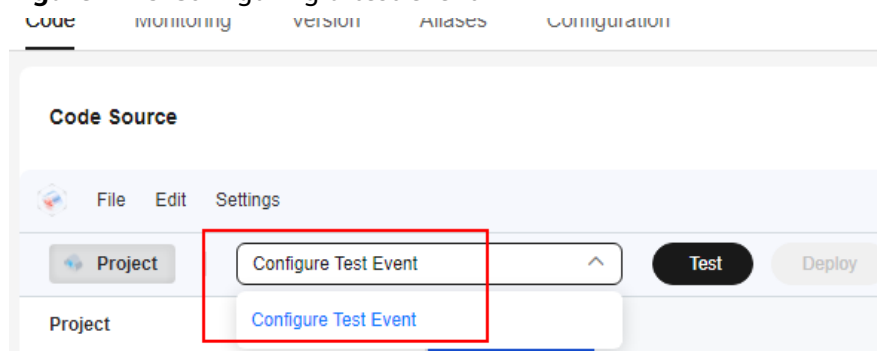
----End

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](#).

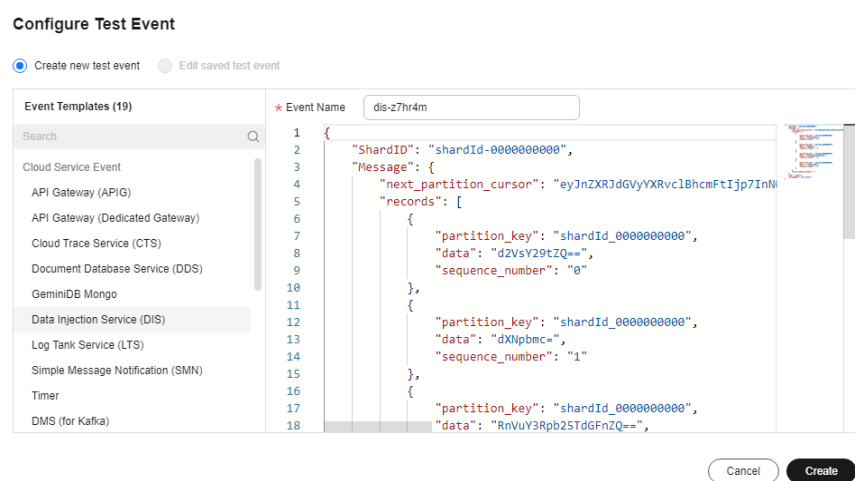
Figure 2-13 Configuring a test event



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



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

```
4mp[1520234900307,"$":{"cbhc0822fba0a0a0a0a0a0a"},{"column":"zj6c2vxdawvzvf0vfwy","timestamp":1520234900307,"$":{"tag":"1}}]}
2018-03-05 07:26:25.212400:00 - request id: 27cb802-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.212400:00 - partition_key : shardId_0000000000 sequence_number : 3 data : c2ydm1j2Q==
2018-03-05 07:26:25.212400:00 - request id: 27cb802-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.212400:00 - insert data : [{"Row":{"key":"cm93aw==","col1":[{"column":"zj6c2vxdawvzvf0vfwy","$":{"tag":"1"}},{"column":"zj6c6fyd618aw0x21eQ==","$":{"cbhc0822fba0a0a0a0a0a0a"}},{"column":"zj6c2Gf0vQ==","$":{"c2ydm1j2Q=="}]}}]}
2018-03-05 07:26:25.213400:00 - request id: 27cb802-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.213400:00 - insert url : http://100.125.1.131:8080/v1.0/taaf7e2ceb4207a13d1a3ae5a9e789/clusters/66f383bd-d08a-4020-b613-61e57c361568/abase/cff_cloud_table/row3
2018-03-05 07:26:25.226400:00 - request id: 27cb802-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.227400:00 - HTTP/1.1 200 OK
2018-03-05 07:26:25.228400:00 - log an empty string
2018-03-05 07:26:25.228400:00 - request id: 27cb802-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.229400:00 - URL: http://100.125.1.131:8080/v1.0/taaf7e2ceb4207a13d1a3ae5a9e789/clusters/66f383bd-d08a-4020-b613-61e57c361568/abase/cff_cloud_table/row3
2018-03-05 07:26:25.230400:00 - request id: 27cb802-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.230400:00 - HTTP/1.1 200 OK
2018-03-05 07:26:25.239400:00 - request id: 27cb802-f68e-40ff-a575-803021e6457b
2018-03-05 07:26:25.239400:00 - {"Row":[{"key":"cm93aw==","col1":[{"column":"zj6c2Gf0vQ==","timestamp":1520234900335,"$":{"c2ydm1j2Q=="}},{"column":"zj6c6fyd618aw0x21eQ==","timestamp":1520234900335,"$":{"cbhc0822fba0a0a0a0a0a0a"}},{"column":"zj6c2vxdawvzvf0vfwy","timestamp":1520234900335,"$":{"tag":"1}}]}]}
2018-03-05 15:26:25.247400:00 Finish request '27cb802-f68e-40ff-a575-803021e6457b', duration: 6349.853ms, billing duration: 6400ms, memory used: 160.38MB.
```

----End

3 Integrating with LTS to Analyze Logs in Real Time

[Introduction](#)

[Preparation](#)

[Building a Program](#)

[Adding an Event Source](#)

[Processing Log Data](#)

[Other Application Scenarios](#)

3.1 Introduction

Scenarios

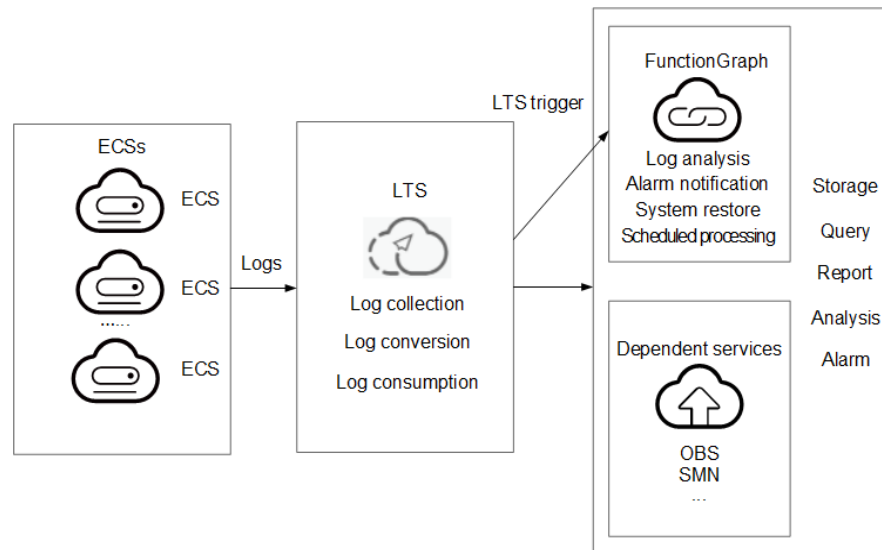
Quickly collect, process, and convert task logs of servers, such as ECSs, through Log Tank Service (LTS).

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.

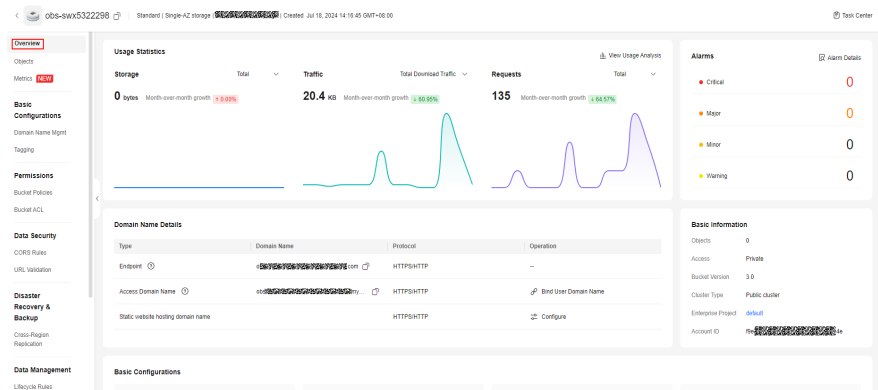
In this example, when log events trigger the specified function through an LTS trigger, the function filters alarm logs and pushes alarm message to the subscription endpoints.

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



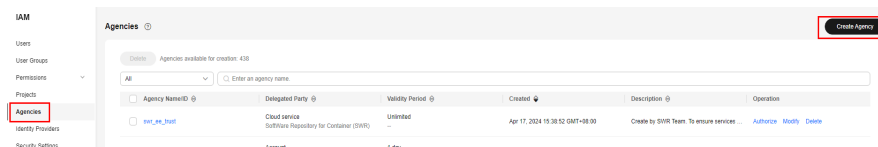
NOTE

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



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:

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: [Create Log Group](#)

* Log Stream: [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](#).

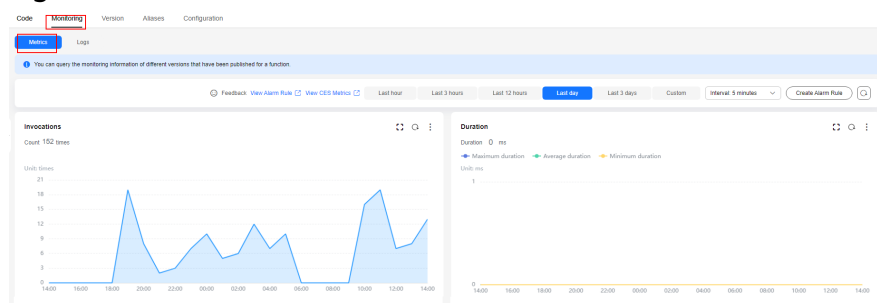
Figure 3-5 Email notification

```
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

[Introduction](#)
[Preparation](#)
[Building a Program](#)
[Adding an Event Source](#)
[Processing Operation Records](#)

4.1 Introduction

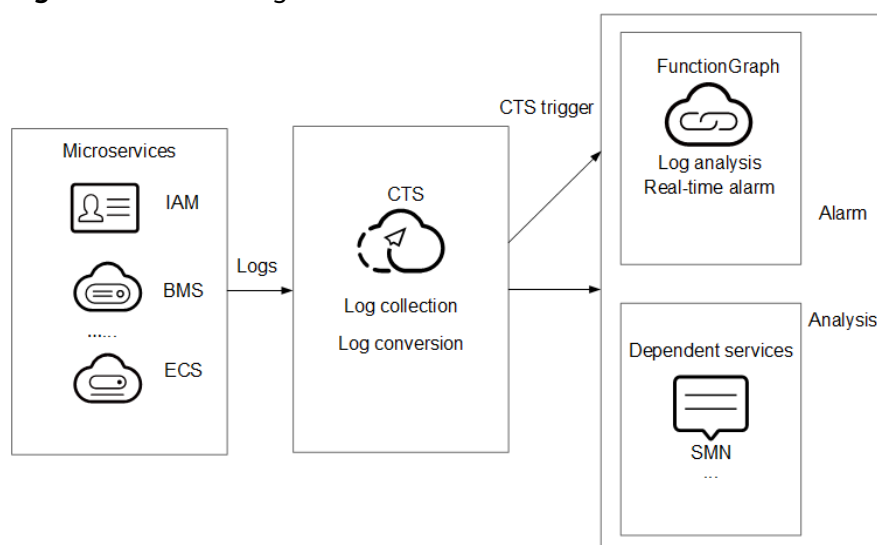
Scenarios

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

The screenshot shows the 'Create Tracker' interface. At the top, there is a warning banner: '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'. Below this, the 'Basic Information' section contains a 'Tracker Name' input field with the value 'test' and a note: '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.' There is also an 'Enterprise Project' dropdown menu currently set to 'default' and a 'View Projects' link.

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:

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	<input type="button" value="Delete.."/>
		logout	

CTS records the logins and logouts of users on IAM.

4.5 Processing Operation Records

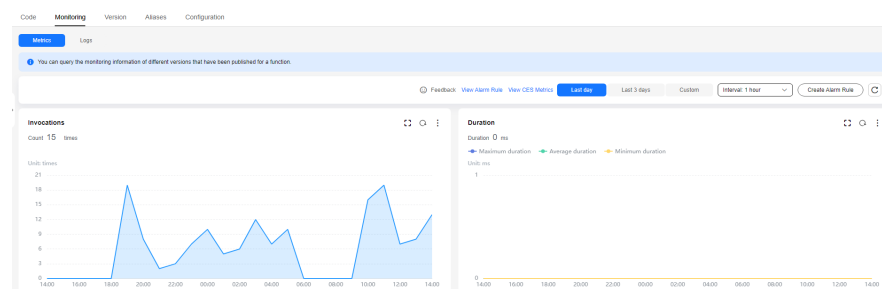
The function runs in response to account logins and logouts to filter those not from the IP address whitelist, and sends a message or email through SMN, as shown in [Figure 4-4](#).

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

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.

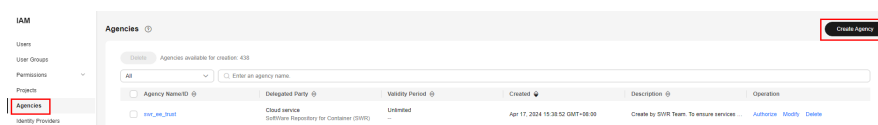
Prerequisites

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](#).

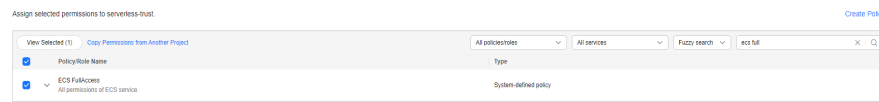
Creating an Agency

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



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

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	<ul style="list-style-type: none"> • 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		<input type="checkbox"/>	Delete
projectId		<input type="checkbox"/>	Delete
whiteLists		<input type="checkbox"/>	Delete
type	SOFT	<input type="checkbox"/>	Delete
ak		<input type="checkbox"/>	Delete
sk		<input type="checkbox"/>	Delete

[+ Add](#)

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

* Rule Fixed rate Cron expression

* 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 [SpringInitializer](#) 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

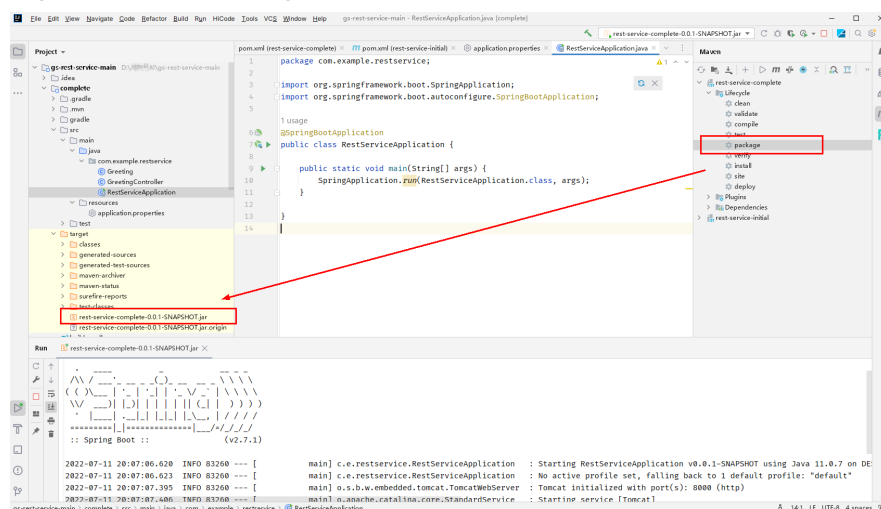
To deploy an existing project to FunctionGraph, change the listening port of the project to **8000**, and create a file named **bootstrap** in the same directory as the JAR file to include the command for executing the JAR file.

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

Building a Code Package

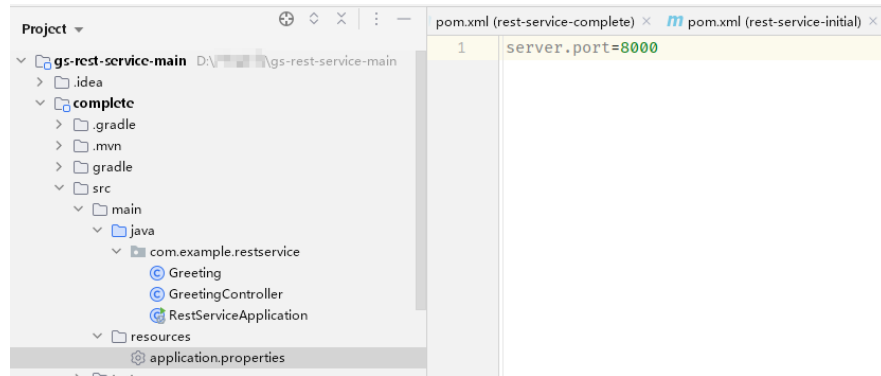
1. Open the Spring Boot project and click **package** in the **Maven** area to generate a JAR file.

Figure 6-1 Generating a JAR file



- 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



- 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.

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

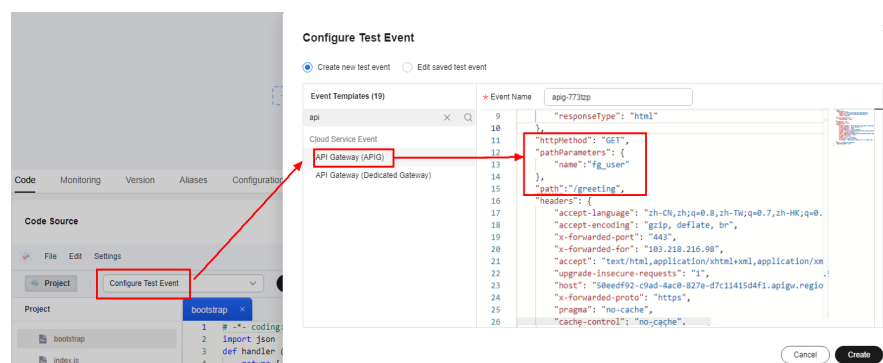
Creating an HTTP Function and Uploading Code

Create an HTTP function and upload the ZIP file. For details, see [Creating an HTTP Function](#).

Verifying the Result

- Using a test event
 - On the function details page, select a version and click **Configure Test Event**.
 - 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.

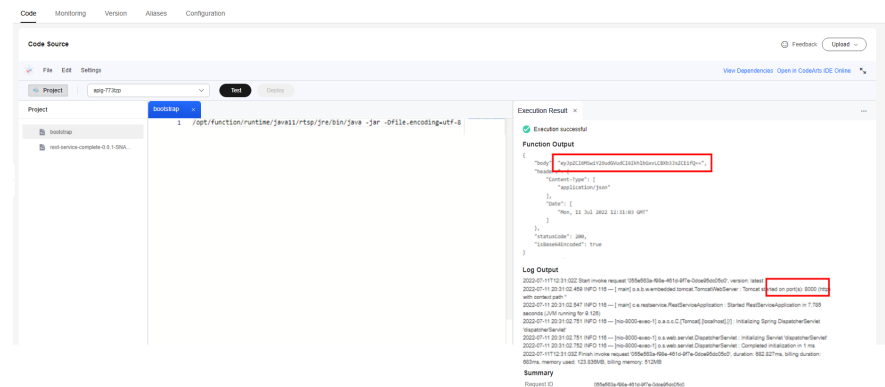
Figure 6-3 Configuring a test event



- c. Click **Create**.
- d. Click **Test** to obtain the response.

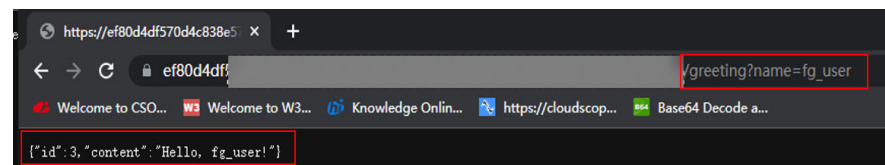
When debugging a function, increase the memory size and timeout, for example, increase them to 512 MB and 5s.

Figure 6-4 Viewing the returned result



- 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



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?

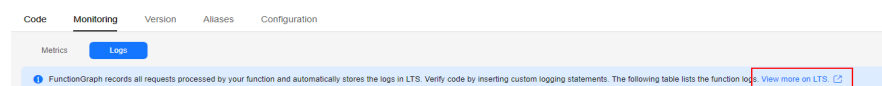
An uploaded code package is stored in the **/opt/function/code/** directory of the function (runtime environments, compute resources, or containers). However, the directory can only be read and cannot be written. If some data must be written to the function during code running and logged locally, or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

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

[Introduction](#)

[Resource Planning](#)

[Building a Program](#)

[Adding an Event Source](#)

[Debugging and Calling the API](#)

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

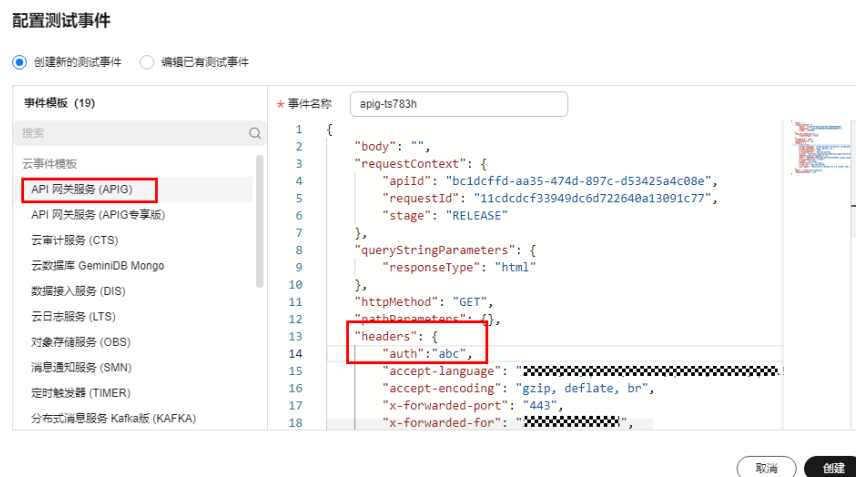
Frontend custom authentication means APIG uses a function to authenticate received API requests. To authenticate API requests by using your own system,

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



- 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

* Type Frontend Backend

* Function URN [Select](#)

* Version/Alias

* Max. Cache Age (s)

Identity Sources

Parameter Location	Parameter Name	Operation
+ Add Identity Source		

Send Request Body

User Data
0/2,048

i 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
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

* URL

Method	Protocol	Subdomain Name	Path
<input type="text" value="ANY"/>	<input type="text" value="HTTPS"/>	a5e.../71db8b4f94e.apic.cn-e...	<input type="text" value="/testAPI"/>

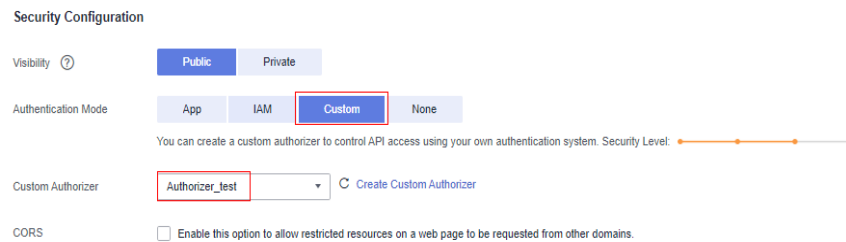
* Gateway Response

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

Tags

Description
0/255

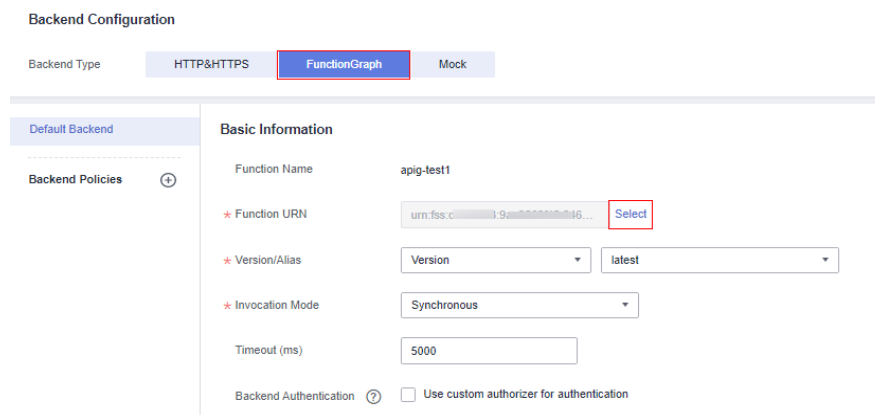
Content Format Type

Figure 7-5 Configuring security settings**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

Step 4 Click **Finish**.

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

Figure 7-7 Publishing an API

----End

7.5 Debugging and Calling the API

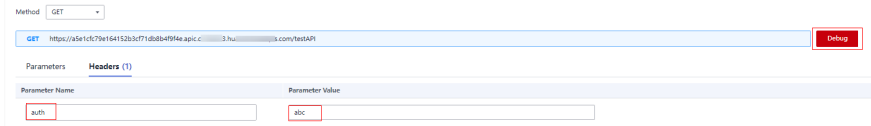
APIG provides online debugging, enabling you to check an API after configuring it.

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



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-Api-Latency: 2140
X-Api-Ratelimit-Api: remain:99,limit:100,time:1 minute
X-Api-Ratelimit-Api: remain:15601,limit:16000,time:1 second
X-Api-Ratelimit-Api-Allenv: remain:5999,limit:6000,time:1 second
X-Api-Ratelimit-Api-Allenv: remain:15601,limit:16000,time:1 second
X-Api-Ratelimit-User: remain:9870,limit:10000,time:1 second
X-Api-Upstream-Latency: 1539
X-Cff-Billing-Duration: 5
X-Cff-Invoke-Summary: {"funcDigest":"64999f78efbc98714f57b3f190573be","duration":4.952,"billingDuration":5,"memorySize":128,"memoryUsed":25.906,"podName":"pool22-300-128-fusion-67fc9b8d95-s6rsv"}
X-Cff-Request-Id: 495bcd5f5-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

[Introduction](#)

[Resource Planning](#)

[Procedure](#)

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	<ul style="list-style-type: none">• Region: AP-Singapore• Specifications: shared gateway or dedicated gateway
FunctionGraph	<ul style="list-style-type: none">• 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-ebc9:~# ls
app.log
root@ecs-ebc9:~# curl -iv https://7ac35d51bd494f39998a.....apig.....h.....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



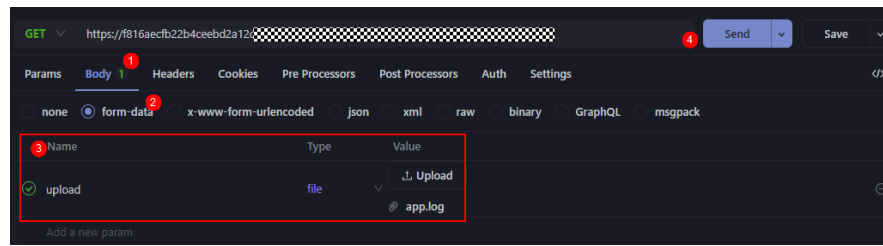
```

root@ecs-e6c9:~# ls
app.log
root@ecs-e6c9:~# curl -iv https://7ac36d51bd494f30998a...api.g.../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



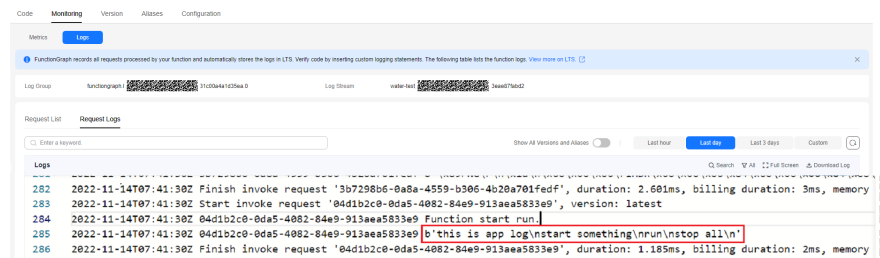
Name: Select **upload**.

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



----End

9 Processing IoT Data

[Introduction](#)

[Preparation](#)

[Building a Program](#)

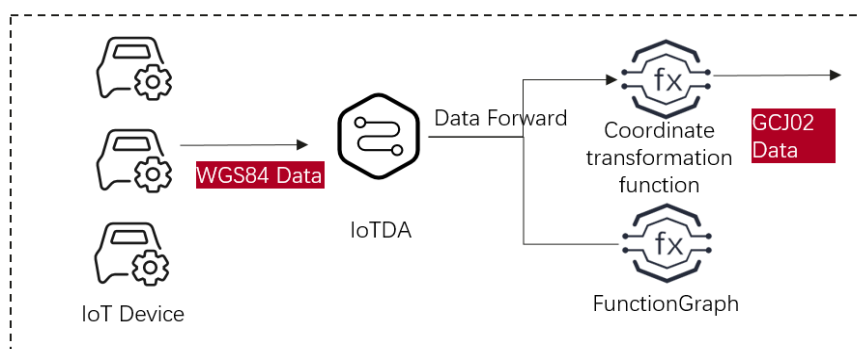
9.1 Introduction

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.

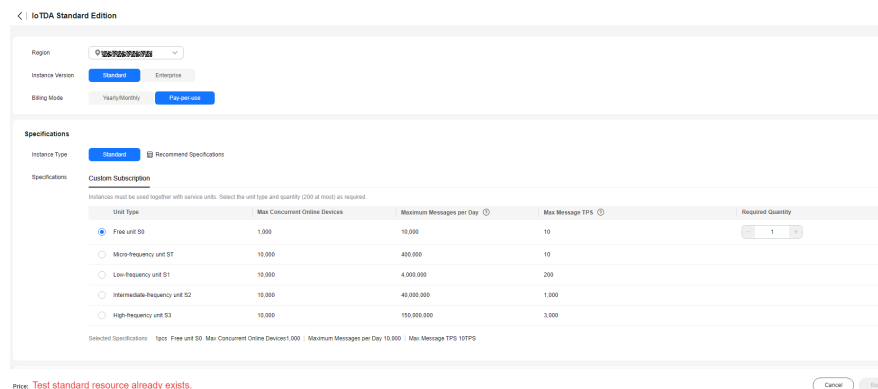
9.2 Preparation

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



- 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

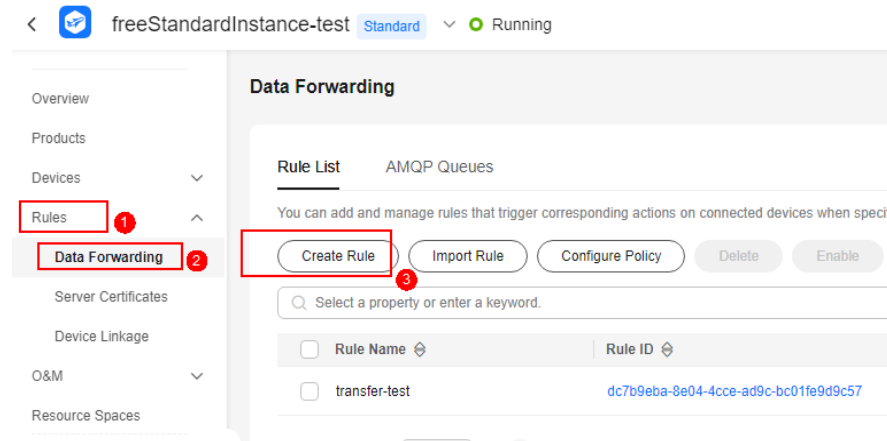
Creating a Forwarding Rule

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



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

NOTE

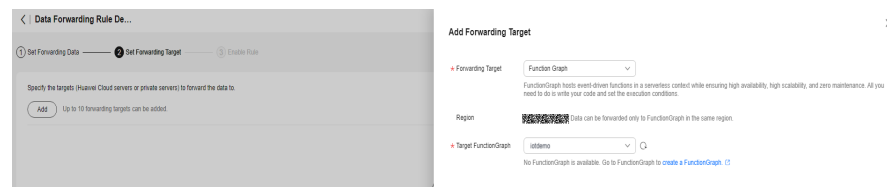
- **Rule Name:** Enter a custom rule name.
- **Data Source:** select **Device message**.
- **Trigger:** select device message reporting.
- **Resource Space:** Retain the default value.

- 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



- 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 + \
```

```

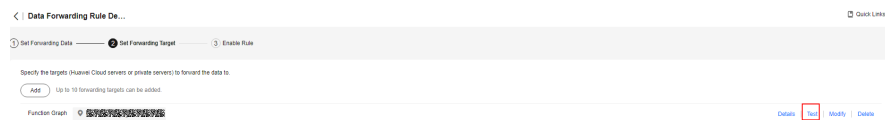
0.1 * x * y + 0.2 * math.sqrt(math.fabs(x))
ret += (20.0 * math.sin(6.0 * pi * x) +
20.0 * math.sin(2.0 * pi * x)) * 2.0 / 3.0
ret += (20.0 * math.sin(pi * y) +
40.0 * math.sin(pi / 3.0 * y)) * 2.0 / 3.0
ret += (160.0 * math.sin(pi / 12.0 * y) +
320 * math.sin(pi / 30.0 * y)) * 2.0 / 3.0
return ret

```

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



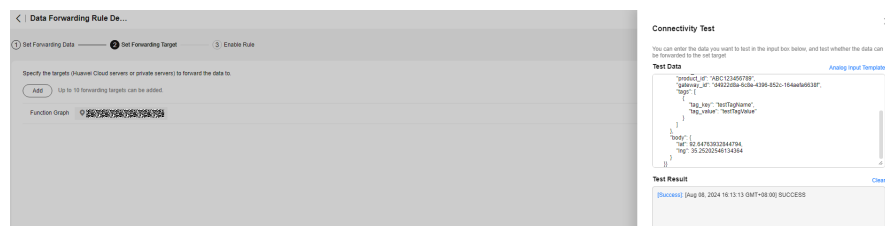
- Step 3** Enter the test data and click **Connectivity Test**.

```

{
  "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



- 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

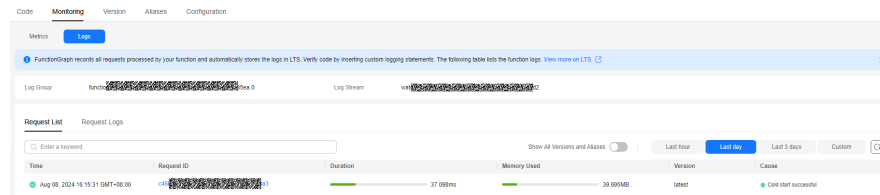
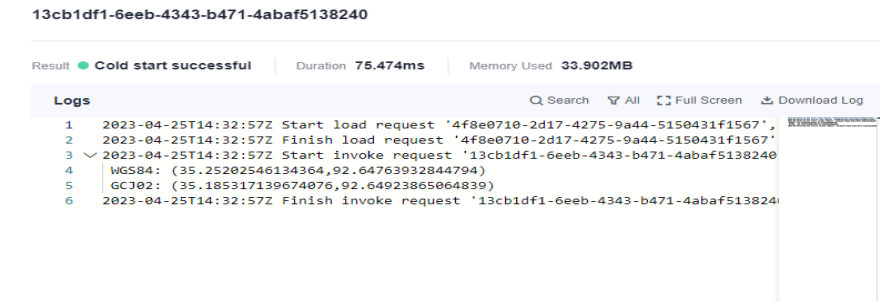


Figure 9-7 Viewing request details



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

----End

10 Workflow + Function: Automatically Processing Data in OBS

[Introduction](#)

[Preparation](#)

[Building a Program](#)

[Processing Images](#)

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

```

14 func Test(b []byte, ctx context.RuntimeContext) (interface{}, error) {
15     ak := ctx.GetAccessKey()
16     sk := ctx.GetSecretKey()
17     obsAddress := os.Getenv(key: "obsAddress")
18     bucket := os.Getenv(key: "bucket")
19     object := os.Getenv(key: "object")
20     client, err := obs.New(ak, sk, obsAddress)
21     if err != nil {
22         log.Printf(format: "err:%v", err)
23         return nil, err
24     }
25     output, err := client.GetObject(&obs.GetObjectInput{
26         GetObjectMetadataInput: obs.GetObjectMetadataInput{
27             Bucket: bucket,
28             Key: object,
29         },
30     })
31     if err != nil {
32         log.Printf(format: "err:%v", err)
33         return nil, err
34     }
35     defer output.Body.Close()
36     img, err := jpeg.Decode(output.Body)
37     if err != nil {
38         log.Printf(format: "err:%v", err)
39         os.Exit(code: -1)
40     }
41     res := rotate180(img)
42     buffer := bytes.NewBuffer(buf: nil)
43     err = jpeg.Encode(buffer, res, &jpeg.Options{Quality: 100})
44     if err != nil {
45         fmt.Println(err)
46         return nil, err
47     }
48     err = ctx.Write(bytes.NewReader(buffer.Bytes()))
49     return struct {}{}, err
50 }
51
52 func rotate180(m image.Image) image.Image {
53     rotate180 := image.NewRGBA(image.Rect(x0: 0, y0: 0, m.Bounds().Dx(), m.Bounds().Dy()))
54     for x := m.Bounds().Min.X; x < m.Bounds().Max.X; x++ {
55         for y := m.Bounds().Min.Y; y < m.Bounds().Max.Y; y++ {
56             rotate180.Set(m.Bounds().Max.X-x, m.Bounds().Max.Y-y, m.At(x, y))
57         }
58     }
59     return rotate180
60 }

```

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

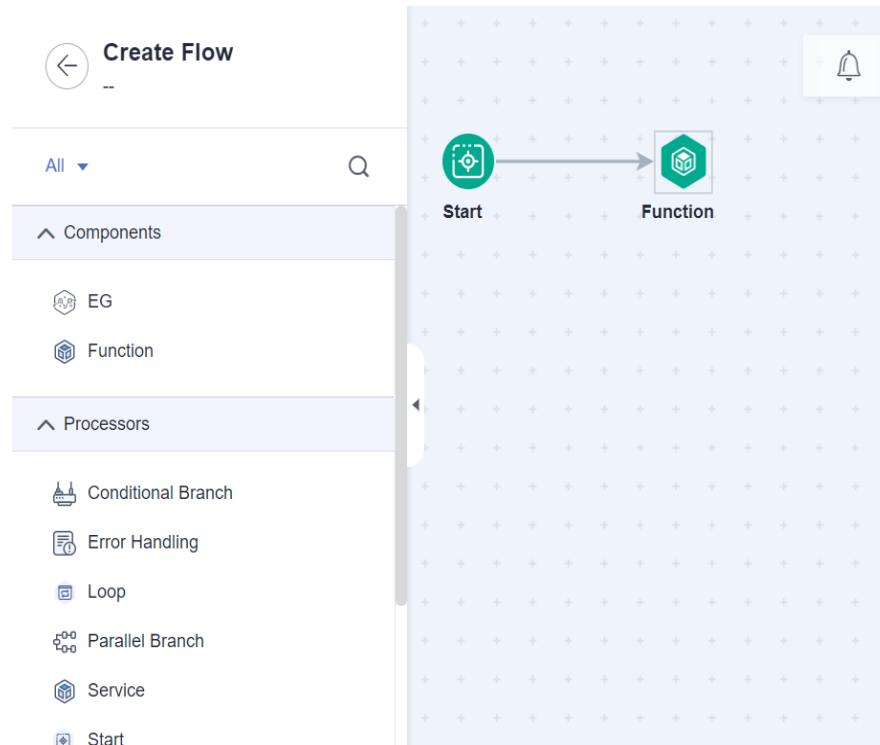
Environment Variable	Description
bucket	OBS bucket parameters defined in the handler.go file for pulling images.
object	The image name parameter defined in handler.go .
obsAddress	The bucket address parameter defined in handler.go for pulling images. The value of obsAddress is in the format of obs.{region}.myhuaweicloud.com . 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 [🔄](#)

* Function [🔄](#)

* Version [🔄](#) [View Function](#)

Function Parameters [?](#)

Key	Value	DefaultValue	Operation
+ Add			

Input Filter Expression [?](#)

Output Filter Expression [?](#)

Turn on StandbyOper...

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

* [View Enterprise Project](#)

* Logs

Streaming Response

Description
0/200

[Save](#) [Start](#)

[OK](#) [Cancel](#)

----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.

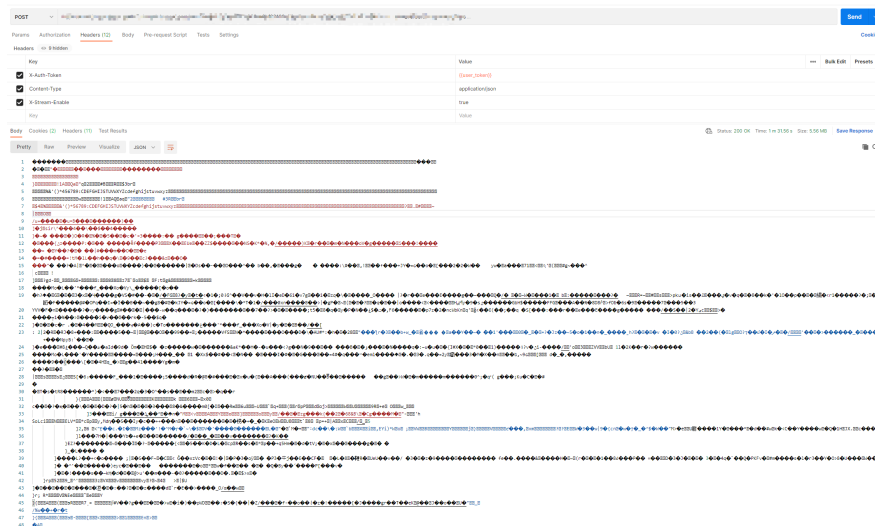
Figure 10-5 Example



Figure 10-6 Uploaded image

Name	Storage Class	Size	Last Modified	Operation
<input type="checkbox"/> image.jpg	Standard	1.86 MB	Apr 23, 2024 17:32:53 GMT+08:00	Download Share More

Using Postman to Trigger the Function Flow



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



11 Filtering Logs in Real Time by Using FunctionGraph and LTS

[Introduction](#)

[Preparation](#)

[Building a Program](#)

[Adding an Event Source](#)

[Processing Results](#)

[Extended Applications](#)

11.1 Introduction

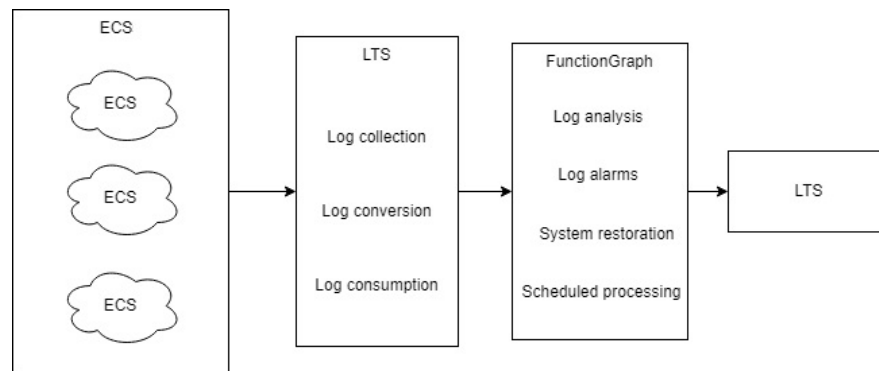
This chapter elaborates the following aspects of the practice:

- 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.

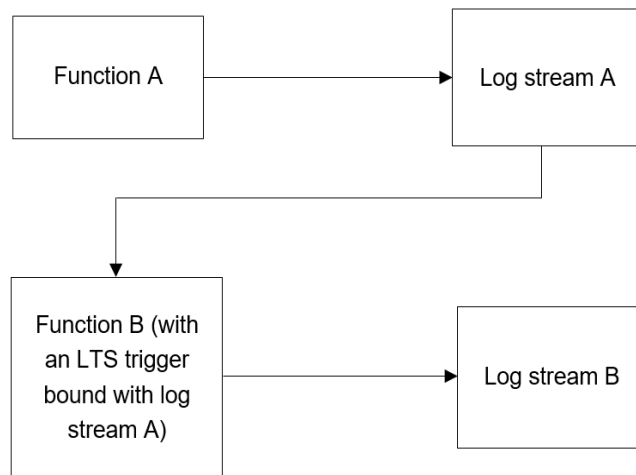
11.2 Preparation

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

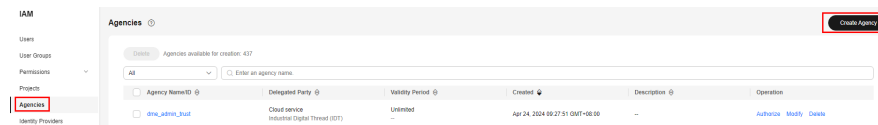


Creating an Agency

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



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

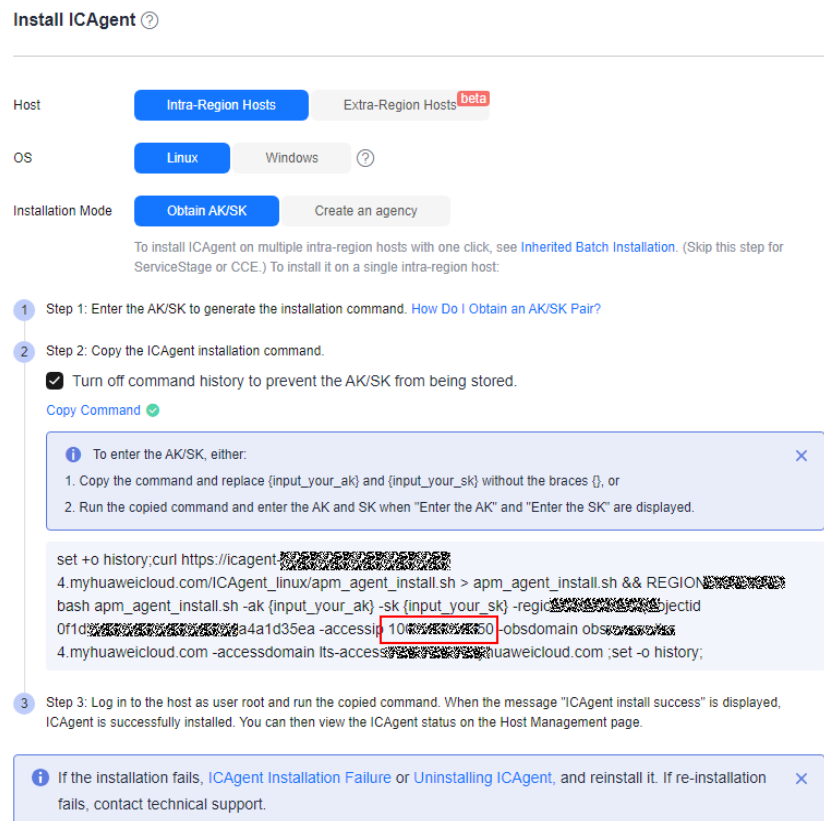
11.3 Building a Program

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



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



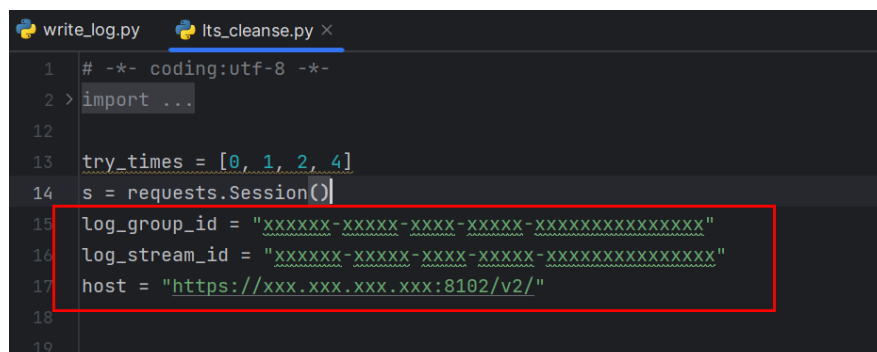
```

1 # -*- coding:utf-8 -*-
2 > import ...
6
7 try_times = [0, 1, 2, 4]
8 s = requests.Session()
9
10 log_stream_id = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx"
11 log_group_id = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx"
12 host = "https://xxx.xxx.xxx.xxx:8102/v2/"
13
14 usage

```

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

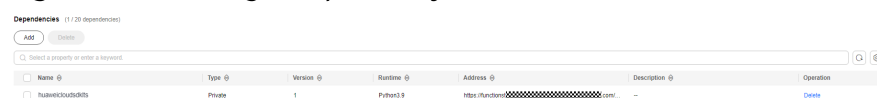


```

1 # -*- coding:utf-8 -*-
2 > import ...
12
13 try_times = [0, 1, 2, 4]
14 s = requests.Session()
15
16 log_group_id = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx"
17 log_stream_id = "xxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx"
18 host = "https://xxx.xxx.xxx.xxx:8102/v2/"
19
20

```

Figure 11-7 Adding a dependency for function B



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

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) <input type="button" value="v"/>
★ Log Group	test-1206 <input type="button" value="v"/> Create Log Group <input type="button" value="↗"/>
★ Log Stream	test-206 <input type="button" value="v"/> Create Log Stream <input type="button" value="↗"/>

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

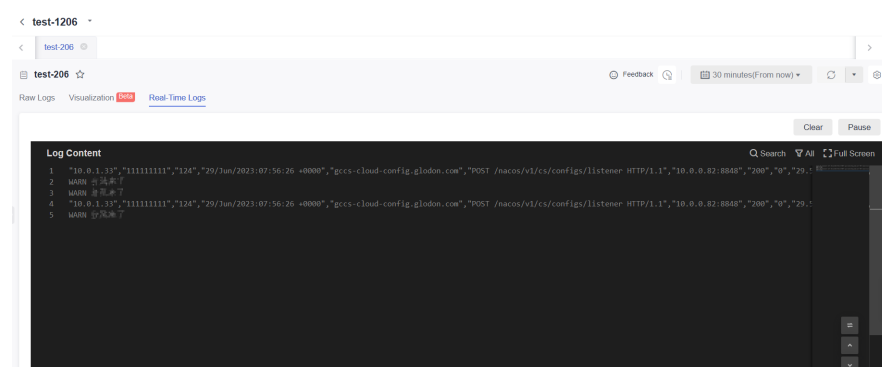
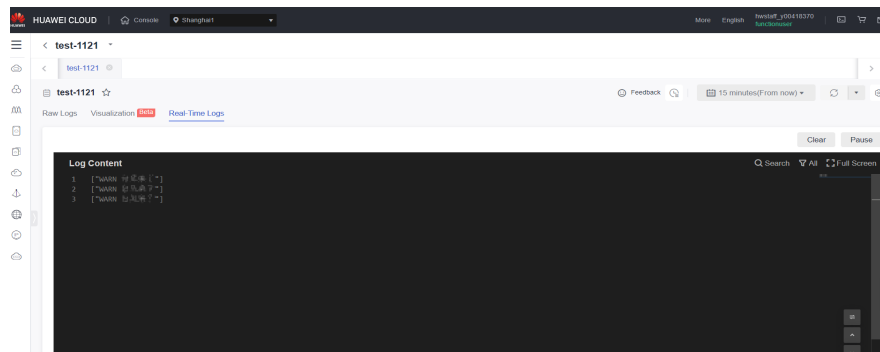


Figure 11-10 Logs after filtering



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

Figure 11-11 Function metrics (1)

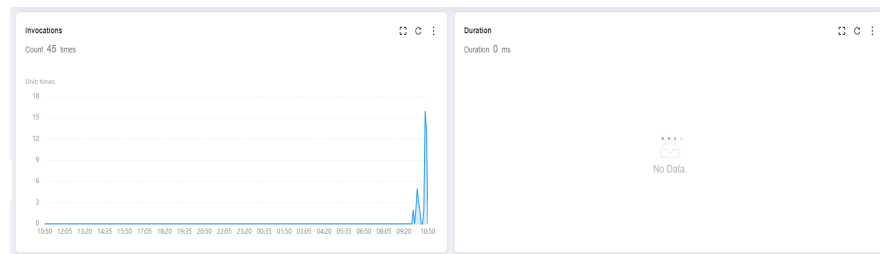


Figure 11-12 Function metrics (2)

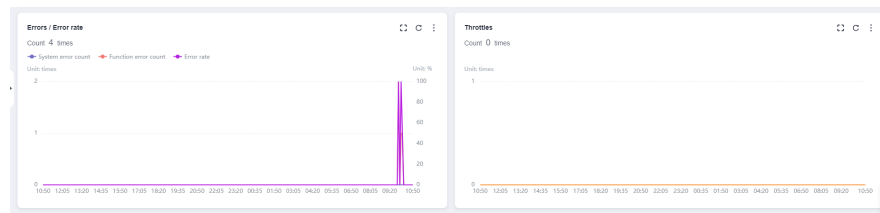


Figure 11-13 Function metrics (3)



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 Deploying AI Drawing Stable-Diffusion in the Application Center

[Introduction](#)

[Preparation](#)

[Application Creation and Deployment](#)

[Application Use](#)

[Custom Models](#)

[Advanced Use](#)

[Disclaimer](#)

12.1 Introduction

You can deploy AI drawing Stable-Diffusion in the application center of FunctionGraph and upload custom models. Also, you can quickly access the Stable-Diffusion WebUI to perform text-to-image and image-to-image transformation (Only available in **CN East-Shanghai1**).

12.2 Preparation

12.2.1 Overview

Custom Models

To use custom models, the Virtual Private Cloud (VPC) and Scalable File Service (SFS) provided by Huawei Cloud are required. You can perform the following steps:

1. [Creating a VPC and Subnet](#)
2. [Creating an SFS Turbo File System](#)

3. [Creating an Agency](#)

Custom Domain Names

To use a custom domain name, Huawei Cloud Domains service is required. You can perform the following steps:

1. Create an information template and perform identity authentication.
2. Wait for the real-name authentication result.
3. Query and purchase a domain name.
4. File the domain name. (Select FunctionGraph as the filing service.)
5. [Configuring Domain Name Resolution](#).

12.2.2 Creating a VPC and Subnet

Step 1 Log in to [Huawei Cloud Network Console](#) and click **Create VPC**. The **Create VPC** page is displayed.

Step 2 On the **Create VPC** page, configure the parameters as follows:

- **Basic Information**
 - **Region:** Select **CN East-Shanghai1**. Currently, the Stable-Diffusion application can be deployed only in this region.
 - **Name:** Enter a custom name.
 - **IPv4 CIDR Block:** Set this parameter based on site requirements.
 - **Enterprise Project:** Use **default**.
- **Default Subnet**
 - **AZ:** Retain the default value. Ensure that the AZs of the file systems created later are the same.
 - **Name:** Enter a custom name.
 - **IPv4 CIDR Block:** Set this parameter based on the site requirements.
- Retain the default values for other parameters.

Step 3 Click **Create Now**.

----End

12.2.3 Creating an SFS Turbo File System

Step 1 Log in to [Huawei Cloud SFS Console](#), select **SFS Turbo**, and click **Create File System**. The **Create File System** page is displayed.

Step 2 Set the following parameters:

- **Billing Mode:** Select a billing mode based on the site requirements. **Pay-per-use** is recommended.

NOTE

For details about SFS billing, see [Billing Overview](#). For details about the pricing, see [Price Calculator](#).

- **Region:** Select **CN East-Shanghai1**. Currently, the Stable-Diffusion application can be deployed only in this region.

- **Project:** Use **default**.
- **AZ:** The option must be the same as the subnet AZ.
- **File System Type:** Select one based on the site requirements.
- **Storage Class:** Select one based on the site requirements.
- **Capacity (TB):** Select one based on the site requirements.
- **VPC:** Select the VPC and subnet created in [Creating a VPC and Subnet](#).
- **Security Group:** Set this parameter as prompted.
- **Enterprise Project:** Use **default**.
- Retain the default values for other parameters.

Step 3 After the configuration, click **Create Now** and wait until the SFS Turbo file system is created.

----End

12.2.4 Creating an Agency

To deploy the Stable-Diffusion WebUI through Huawei Cloud serverless application center, you need to use FunctionGraph with other cloud services and authorize operation permissions on cloud service resources to FunctionGraph by configuring an agency. It is strongly recommended that you create the agency in advance as the agency takes effect 15 to 30 minutes later.

Step 1 Log in to the [IAM console](#). In the navigation pane, choose **Agencies**. On the **Agencies** page that is displayed, click **Create Agency** in the upper right corner.

Step 2 Set the following parameters:

- **Agency Name:** Enter a custom agency name.
- **Agency Type:** Select **Cloud service**.
- **Cloud Service:** Select **FunctionGraph**.
- **Validity Period:** Select **Unlimited**.
- **Description (Optional):** Enter a description.

Figure 12-1 Creating an agency

★ Agency Name

★ Agency Type Account
 Delegate another Huawei Cloud account to perform operations on your resources.

Cloud service
 Delegate a cloud service to access your resources in other cloud services.

★ Cloud Service

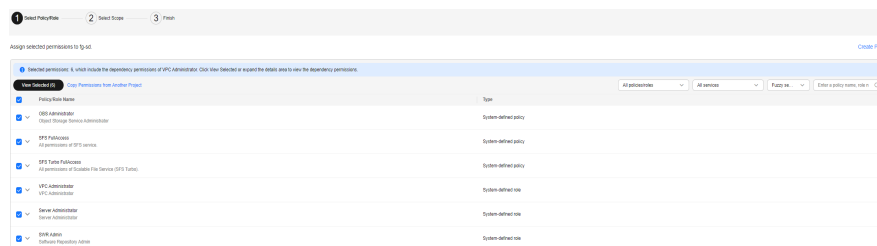
★ Validity Period

Description

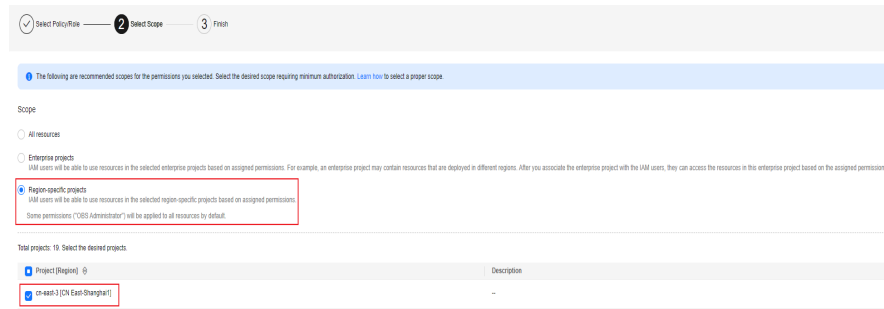
0/255

Step 3 After setting the parameters, click **Next**. Search for and select the following five system-defined policies or roles from the list, and click **Next**.

- OBS Administrator
- SFS FullAccess
- SFS Turbo FullAccess
- VPC Administrator (If you select this policy, SWR Admin is automatically selected.)
- SWR Admin

Figure 12-2 Selecting a policy

Step 4 Select a scope for the permissions as required. If you are not sure about the scope, select **All resources** or **CN East-Shanghai1** in **Region-specific projects**.

Figure 12-3 Region-specific projects**NOTE**

OBS Administrator does not support this authorization scope. By default, the permissions apply to all resources.

Step 5 Click **OK**. The agency is created. Wait until the agency takes effect.

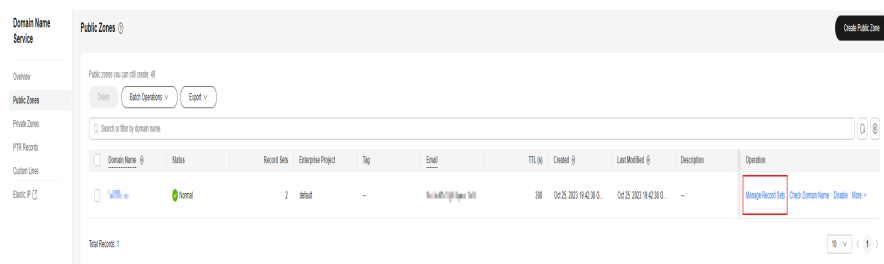
----End

12.2.5 Configuring Domain Name Resolution

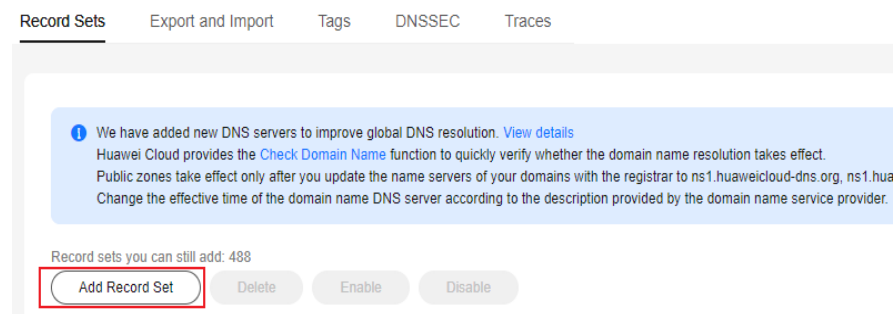
Step 1 After the steps in **Application Creation and Deployment** are complete, click **Bind now**. Click the **Summary** tab and copy the subdomain name for backup.

Figure 12-4 Copying the subdomain name

Step 2 Log in to the DNS console, choose **Public Zones**, and click **Manage Record Set** on the right of the purchased domain name.

Figure 12-5 Clicking Manage Record Sets

Step 3 On the **Record Sets** tab, click **Add Record Set**.

Figure 12-6 Adding a record set

Step 4 In the displayed **Add Record Set** dialog box, configure the information.

1. **Name:** The value can be customized.
2. **Type:** Select **CNAME – Map one domain to another**.
3. **Value:** Enter the subdomain name copied in [step 1](#).

Retain the default values of other parameters.

Figure 12-7 Configuring the record set

test10.com

Add Record Set [Documentation](#)

Type

CNAME – Map one domain to another

Name

Example: www .com

Line ?

Default

TTL (s) ?

300

Value ?

Advanced Settings (Optional)

Alias: No Weight: 1 Tag: -- Description: --

Cancel OK

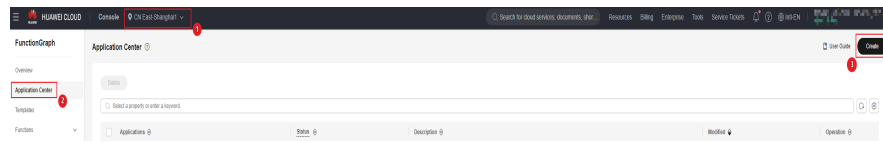
Step 5 Click **OK**.

----End

12.3 Application Creation and Deployment

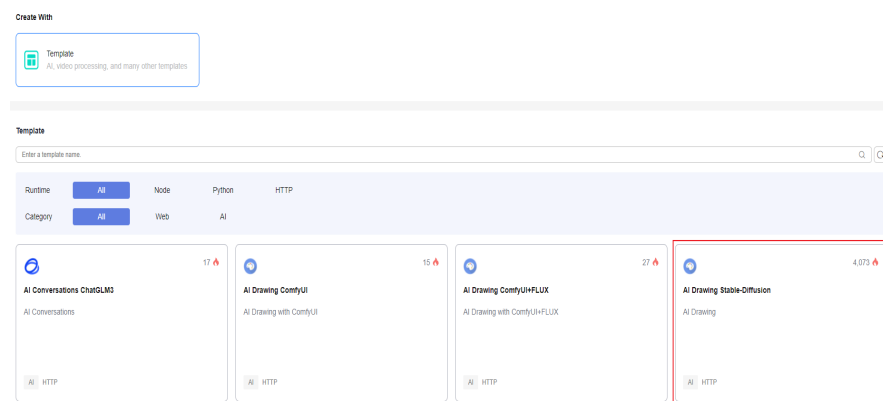
Step 1 Log in to the [FunctionGraph console](#) and select **CN East-Shanghai1** region. In the navigation pane on the left, choose **Application Center**. In the upper right corner, click **Create**. The page for selecting a template is displayed.

Figure 12-8 Creating an application



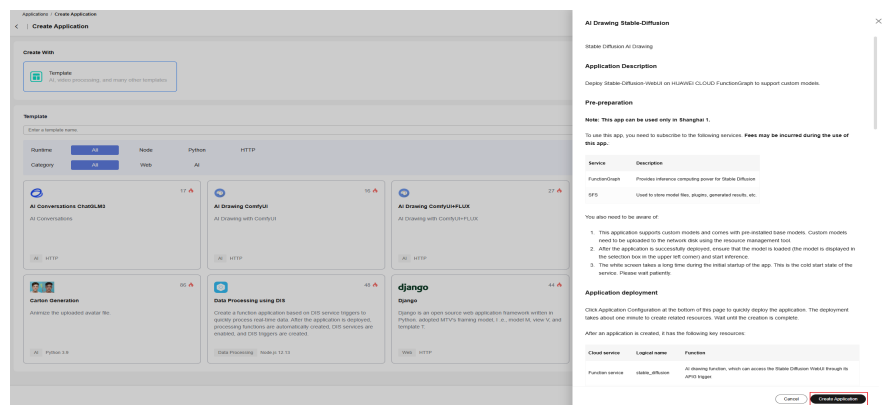
Step 2 Click **Create Application**. The application introduction page is displayed. Read it carefully. If this option is unavailable, check whether your current region is **CN East-Shanghai1**.

Figure 12-9 Selecting an application template



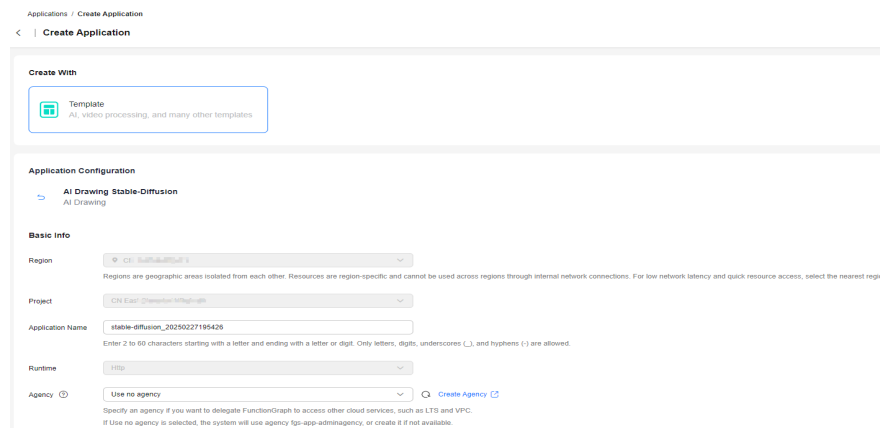
Step 3 Click **Learn More** to read the application usage document carefully and click **Create Application** in the lower right corner. The configuration page is displayed.

Figure 12-10 Template description



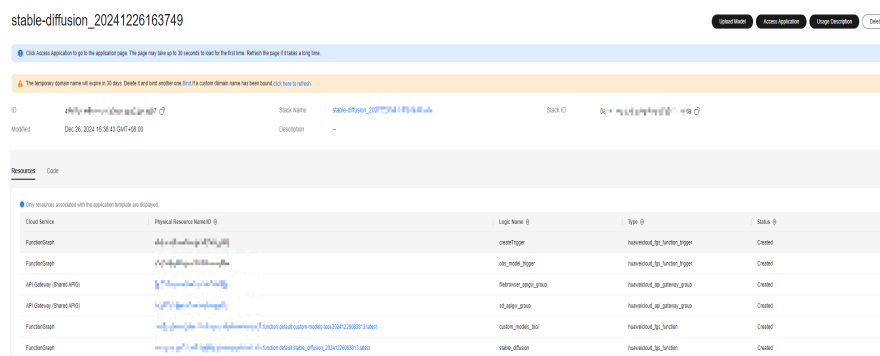
Step 4 On the **Configure Application** page, set the **Application Name**, **Agency**, and **Description** (optional). Select the agency created in **Creating an Agency** and click **Create Now** in the lower right corner.

Figure 12-11 Configuring an application



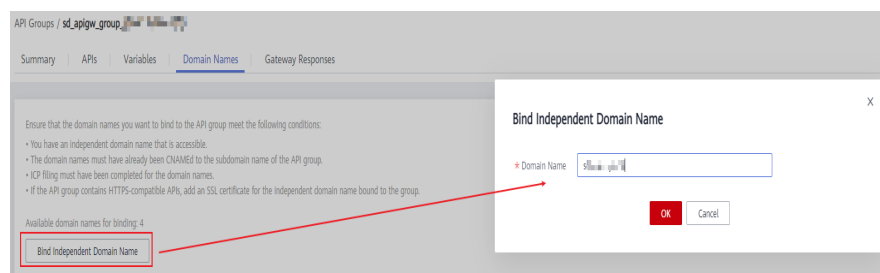
Step 5 Wait until the application is created. The created application contains resources such as functions, agencies, and triggers, as shown in **Figure 12-12**.

Figure 12-12 Application created



Step 6 The output of the text-to-image application is determined by the open-source models and user input. To use it legally, you need to bind a licensed domain name for open access. Click **Bind now** to bind the resolved custom domain name. Then return to the application page and click **Refresh** to start using Stable-Diffusion.

Figure 12-13 Binding a custom domain name



----End

12.4 Application Use

The output of the text-to-image application is determined by the open-source models and user input. To use it legally, you need to bind a licensed domain name

for open access. Click **Bind now** on the application details page to bind the custom domain name. For details, see [Configuring the Domain Name for Calling APIs](#). Open the bound domain name in the browser or click **Access Application** to access the Stable Diffusion WebUI, it takes a long time to start the system for the first time.

Figure 12-14 Getting started with Stable-Diffusion

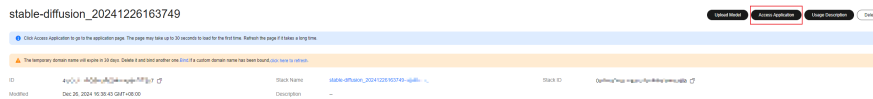
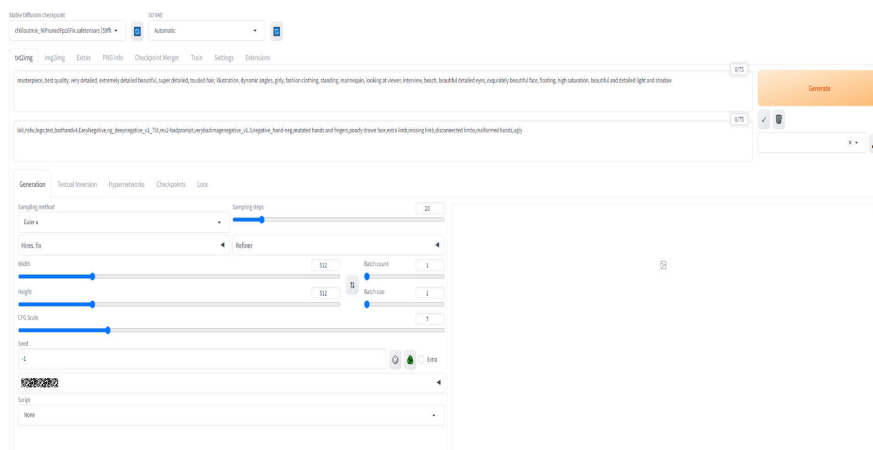


Figure 12-15 Stable-Diffusion WebUI



NOTE

If reserved instances are not used, the first cold start takes about 30 seconds. If the loading times out, refresh the page.

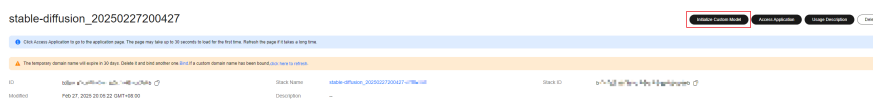
12.5 Custom Models

The Stable-Diffusion application supports custom models. You can upload your own model to the corresponding path of a specified OBS bucket. The auxiliary function will forward the model to the shared path of SFS Turbo. And you can use it after reloading the page.

12.5.1 Initializing a Model

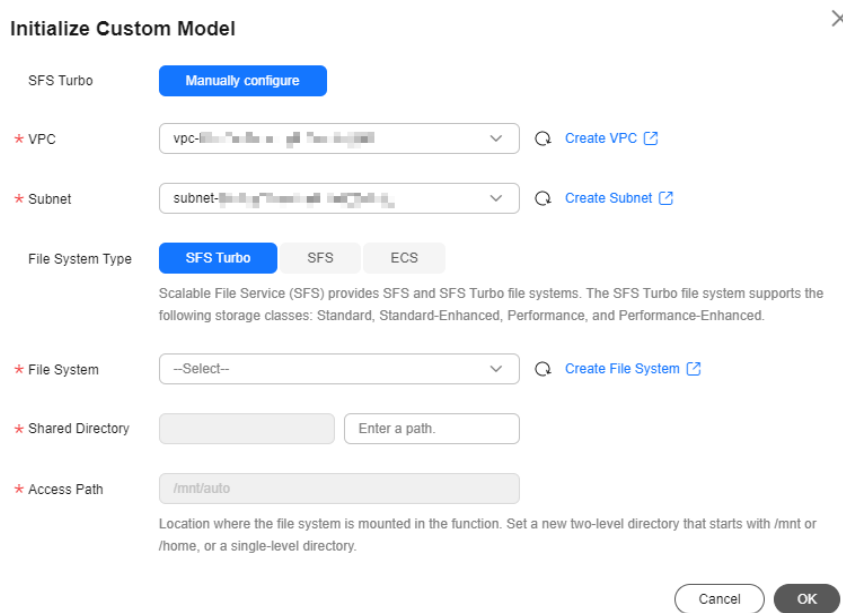
- Step 1** Log in to the [FunctionGraph console](#) and select **CN East-Shanghai1** region. In the navigation pane on the left, choose **Application Center**. Click the name of the target application in the list. The **Summary** page is displayed.
- Step 2** On the **Summary** page, click **Initialize Custom Model** to start the configuration.

Figure 12-16 Initializing a custom model



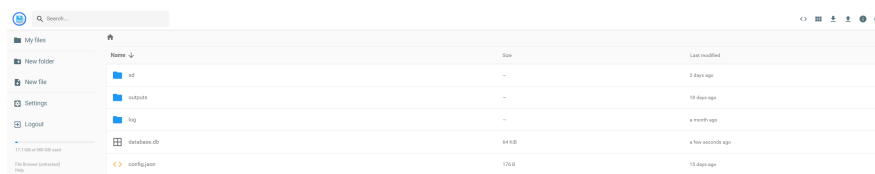
Step 3 Select the created **VPC**, **subnet**, and **file system**. The default function access path is **/mnt/auto**. Set other parameters based on service requirements. After the configuration is complete, click **OK** and wait until the initialization is complete.

Figure 12-17 Configuring custom model initialization



Step 4 After **step 3** is complete, click **Upload Model**. The file management page is displayed. The default username and password are **admin**. Change the password after login to ensure data security.

Figure 12-18 File management



Step 5 **Table 12-1** lists some key directories.

Table 12-1 Key directory path

Path	Description
sd/models/Stable-diffusion	Stores Checkpoint model files.
sd/models/VAE	Stores VAE files.
sd/models/Lora	Stores LoRA models.
sd/extensions	Stores plug-ins.
sd/outputs	Stores outputs.

----End

12.5.2 Importing and Loading Models

Step 1 Upload the custom models, such as Checkpoint, VAE, and LoRA models, to the designated path.

Figure 12-19 Checkpoint models

Name	Size	Last modified
readme.md	invalid link	3 days ago
MR 3DQ_SDXL V0.2.safetensors	6.62 GB	6 months ago
MR 3DQ_1.5.5.5.v2.safetensors	5.28 GB	7 months ago
chilloutmix_NIPrunedFp16Fix.safetensors	invalid link	3 days ago
atomixAnime_v20.safetensors	1.99 GB	3 days ago

Figure 12-20 VAE models

Name	Size	Last modified
vae-ft-mse-840000-ema-pruned.safetensors	319.14 MB	3 days ago
vae-ft-pruned_2.0.safetensors	319.14 MB	3 days ago
Put your VAE release project folder here.txt	invalid link	3 days ago

Figure 12-21 LoRA models

Name	Size	Last modified
lora_v1.0.safetensors	144.11 MB	3 days ago
luer lion head.safetensors	144.11 MB	3 days ago
v1.0.safetensors	144.11 MB	3 days ago
Put your Lora release project folder here.txt	invalid link	3 days ago

Step 2 After the upload is complete, return to the Stable-Diffusion WebUI and reload the model. The loading may take a long time and after that the new models will be displayed.

Figure 12-22 Checkpoint models displayed

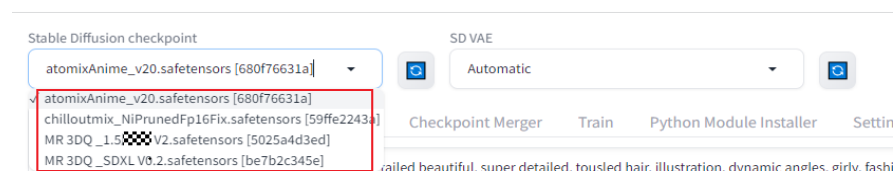


Figure 12-23 VAE models displayed

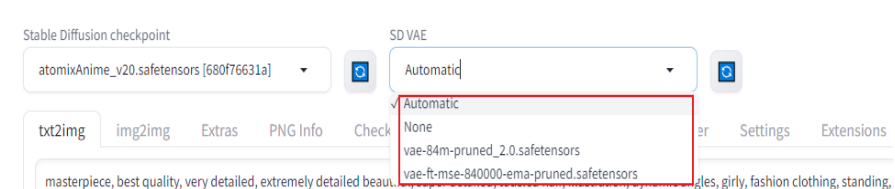


Figure 12-24 LoRA models displayed



----End

12.6 Advanced Use

12.6.1 Using ECS as an NFS Server to Isolate Resources of Multiple Users

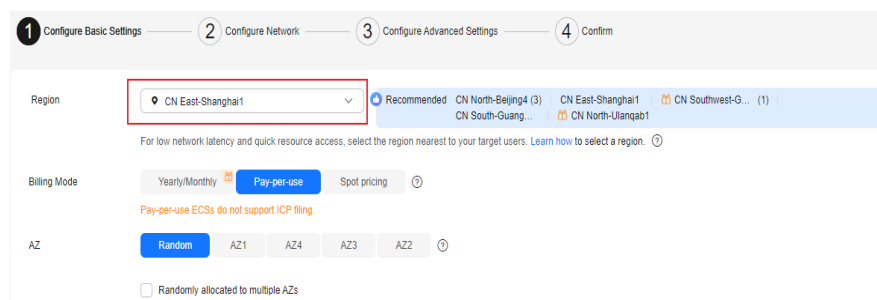
Using ECS as an NFS Server to Isolate Resources of Multiple Users

In addition to SFS file systems, NFS shared paths on ECS can also be mounted to FunctionGraph functions. Using ECS makes it easier to manage resources for multiple tenants.

Step 1 Buy an ECS. Pay attention to the following aspects:

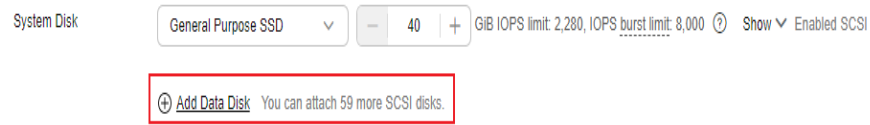
- Select **CN East-Shanghai1** for **Region**. You can select the ECS specification and image version based on service requirements. EulerOS 2.5 64bit(40 GiB) is used as an example. Some Linux commands vary depending on the image version.

Figure 12-25 Basic information



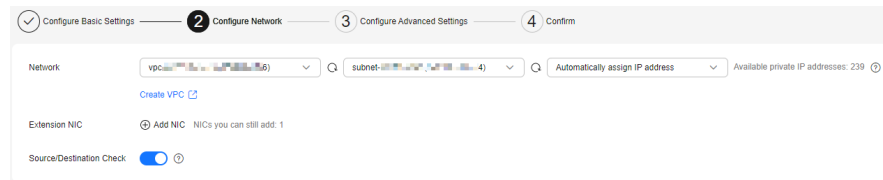
- The size of many model files ranges from 1 GB to more than 10 GB. It is advised to select a system disk or mount a data disk as required. Click **Next: Configure Network**.

Figure 12-26 Configuring a system disk



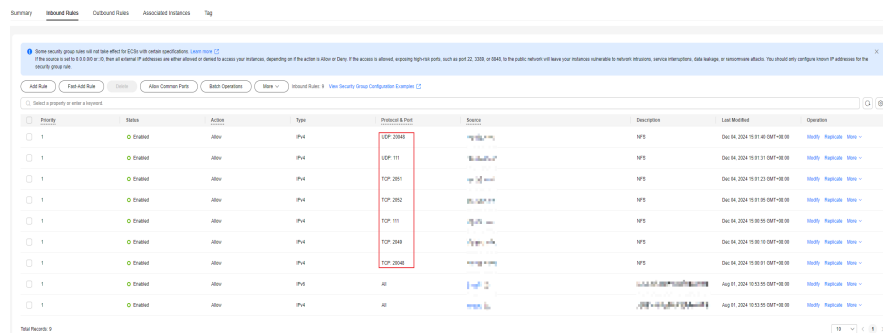
- Select the **VPC and subnet** created in the previous step.

Figure 12-27 Configuring the VPC and subnet



- For details about how to configure a security group, see the following figure. For inbound rules, enable ports 111, 2049, 2051, 2052, and 20048 for the IP address segment in the subnet for the NFS service. Configure other ports, such as port 22 for SSH and SFTP and port 21 for FTP.

Figure 12-28 Configuring a security group



- Purchase an EIP based on service requirements.

Figure 12-29 Purchasing an EIP



Step 2 Configure NFS sharing.

After purchasing an ECS, you can configure NFS sharing. The following uses user1 and user2 as an example.

1. Add users user1 and user2 and create the **home** directory.

```
useradd -m user1 && useradd -m user1
```

2. Change the password.

```
passwd user1
passwd user2
```

3. Create a shared directory for users and change the operation permission on the shared directory to 777.

```
mkdir /home/user1/share && chmod 777 /home/user1/share
mkdir /home/user2/share && chmod 777 /home/user2/share
```

NOTE

The shared directory is a subdirectory of the **home** directory and inaccessible to other users. This ensures that the function has the operation permission after mounting the directory. Therefore, permission 777 will not cause unauthorized operations.

4. Install the NFS service.

```
yum install rpcbind nfs-utils // Run the corresponding command for images that use apt or other package management tools.
```

5. Add the following content to the **/etc/exports** file:

```
/home/user1/share xx.xx.xx.xx/xx(rw) // Fill xx.xx.xx.xx/xx with the network segment of the created subnet.
/home/user2/share xx.xx.xx.xx/xx(rw) // Fill xx.xx.xx.xx/xx with the network segment of the created subnet.
```

6. Start the NFS service.

```
systemctl start rpcbind nfs
```

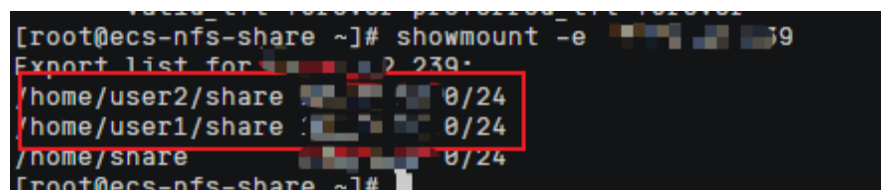
7. Enable the NFS service to automatically start upon system startup.

```
echo "xx.xx.xx.xx:/home/user1/share /nfs nfs4 defaults 0 0" >> /etc/fstab // Enter the IP address of the ECS in the subnet.
echo "xx.xx.xx.xx:/home/user2/share /nfs nfs4 defaults 0 0" >> /etc/fstab // Enter the IP address of the ECS in the subnet.
mount -av
```

8. Check the sharing information. If the following is displayed, the NFS sharing is successfully configured.

```
showmount -e xx.xx.xx.xx (IP address of the server)
```

Figure 12-30 Checking sharing information



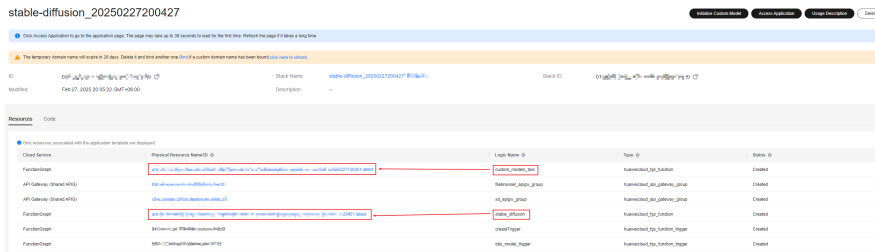
----End

Mounting an Application

In the FunctionGraph application center, create a Stable-Diffusion application for user1 and user2, respectively. The following uses user1 as an example. The procedure for user2 is the same.

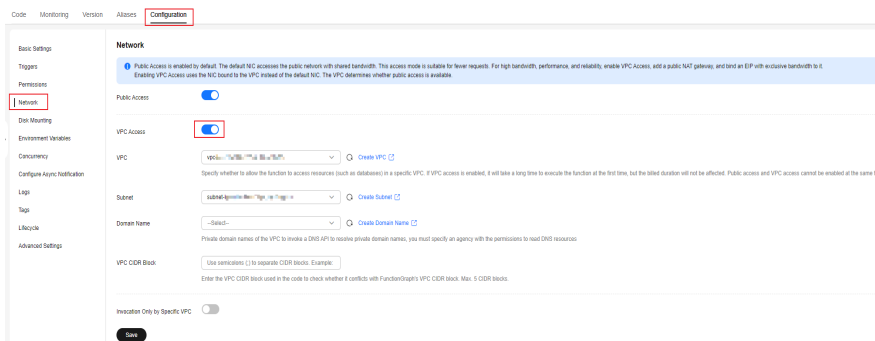
- Step 1** Go to the **Summary** page of user1 application, find the function whose logical name is **stable_diffusion** or **custom_models_tool** in **Resources**, and click the link to go to the function details page. The operations of the two functions are the same. The **stable_diffusion** function is used as an example.

Figure 12-31 Functions of user1



Step 2 On the function details page, choose **Configuration > Network**, enable **VPC Access**, configure the VPC and subnet, select the same VPC and subnet as those of ECS, and click **Save**.

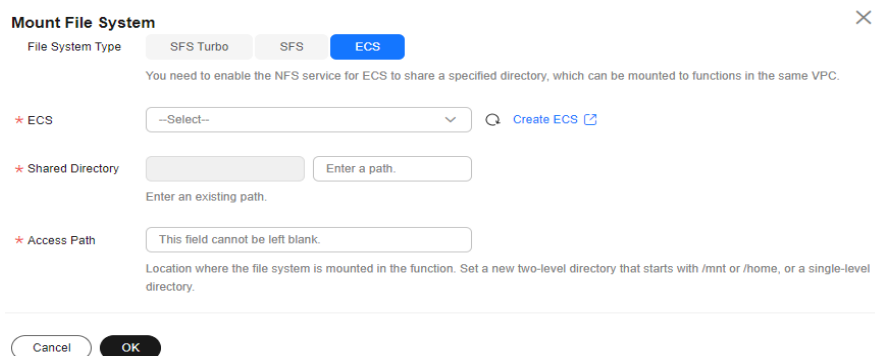
Figure 12-32 Configuring network



Step 3 In the navigation pane, choose **Disk Mounting > Mount File System**.

- **File System Type:** Select **ECS**.
- **ECS:** Select the ECS used to configure the NFS sharing.
- **Shared Directory:** Enter **/home/user1/share** (e/home/user2/share for user2).
- **Access Path:** Enter **/mnt/auto**.

Figure 12-33 Mounting a file system



Step 4 After the configuration is complete, click **OK**.

----End

Accessing the WebUI Program to Create Directories and Files

- Step 1** Go to the **Summary** page of user1, click **Access Application** in the upper right corner, and wait until the function is started. The function automatically creates a directory required by the application in the mounting directory.
- Step 2** Go back to the **Summary** page of user1 application and click **Upload Model** to open the file management tool.

Figure 12-34 File management tool

Name	Size	Last modified
textual_inversion_templates	—	3 days ago
scripts	—	3 days ago
root	—	3 days ago
repositories	—	3 days ago
output	—	3 days ago
models	—	3 days ago
localizations	—	3 days ago
extensions-builtin	—	3 days ago
extensions	—	3 days ago
embeddings	—	3 days ago
configs	—	3 days ago
ui-config.json	66.92 KiB	a few seconds ago
styles.css	0 B	3 days ago
config.json	10.68 KiB	2 minutes ago
cache.json	164.13 KiB	3 days ago

Step 3 Save your model and plug-in files to the corresponding directories. The following is an introduction of some main directories.

- **sd/models/Stable-diffusion:** Path for storing Stable-Diffusion Checkpoint model files.
- **sd/models/VAE:** Path for storing VAE files.
- **sd/models/Lora:** Path for storing LoRA models.
- **sd/extensions:** Path for storing plug-ins.

Step 4 Reload the WebUI. The newly imported models are displayed.

Figure 12-35 Checkpoint models

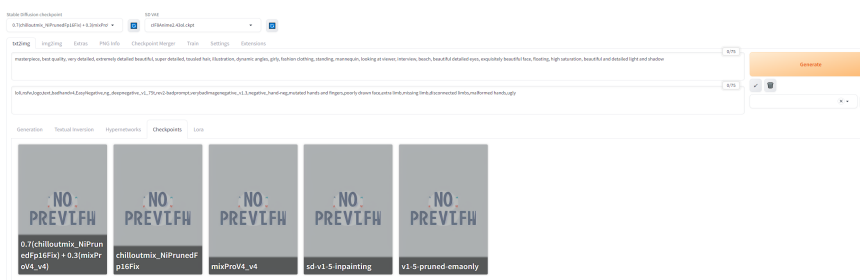
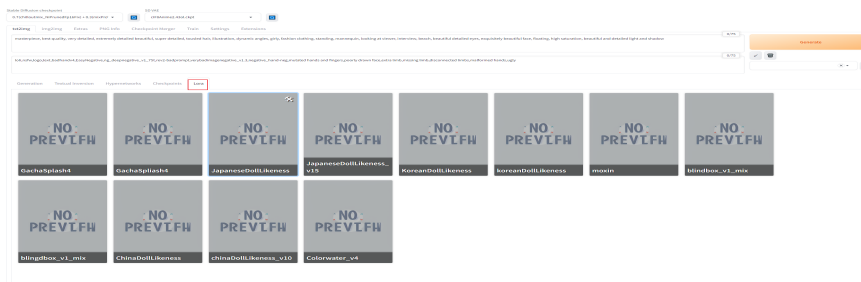
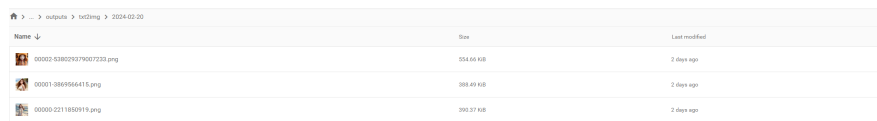


Figure 12-36 LoRA models



Step 5 Click **Generate** in the upper right corner. The image is automatically saved to the `/home/user1/share/sd/outputs/txt2img/202x-xx-xx` directory.

Figure 12-37 Directory for saving images



----End

12.6.2 API Access

By default, API access is disabled for the deployed Stable-Diffusion application. To enable it, configure and save the following environment variable. For details, see [Configuring Environment Variables](#).

Table 12-2 Environment variable

Key	Value
EXTRA_ARGS	--api --api-auth username1:password1,username2:password2 --nowebui

Configure concurrency parameters for the Stable-Diffusion function by referring to [Configuring Multiple Concurrency for a Single Instance](#). The recommended parameters are as follows:

- WebUI mode
 - Requests per instance ≥ 100 . According to the test result, the number of requests for a single instance is about 15, and you are advised to set the number for multiple users to greater than 100.
 - Max. instances per function: 1. When generating graphs in WebUI mode, the progress is checked continuously. However, if there are multiple instances, the requests may be sent in a disorderly manner, which prevents the progress from being displayed and the final result from being viewed. To avoid this, it is advised to set the maximum number of instances for a single function to 1.
- API mode
 - Requests per instance: 1–5. This ensures that an instance does not have too many queuing requests. When the number of concurrent requests

- reaches the threshold, a new instance is displayed to ensure the graph generation speed.
- Maximum instances per function: 400 (default). You can change the value as required.

Using Moderation to Review the Generation Result

Stable-Diffusion is an AIGC inference model. The final result of image generation may be uncertain due to different prompts and models, which may cause pornographic and violent violation risks. You are advised to use Stable-Diffusion with Huawei Cloud Moderation to review the generated result. For details, see [Image Moderation \(V3\)](#).

12.6.3 Enabling WebUI Authentication

The WebUI authentication is disabled by default. To prevent your functions from being stolen due to domain name leakage, you can configure environment variables for Stable-Diffusion functions to enable WebUI authentication and for [API Access](#) at the same time. For details, see [Environment Variables](#). After the setting is complete, refresh the WebUI. Enter the username and password to start drawing.

Table 12-3 Environment variable

Key	Value
EXTRA_ARGS	--gradio-auth user1:password1

12.6.4 Sharing Models and Plugins

[Using ECS as an NFS Server to Isolate Resources of Multiple Users](#) describes using an ECS as the NFS server to isolate resources when multiple users use the Stable-Diffusion WebUI. This solution is applicable to scenarios where strong isolation is required between users.

In some scenarios, if you want multiple users to share some resources, such as model files and plug-ins, the storage space is wasted if each user copies a file. This problem can be solved by mounting different applications to the same SFS file system. However, user operations and configurations may interfere with each other, perform the following operations to solve it:

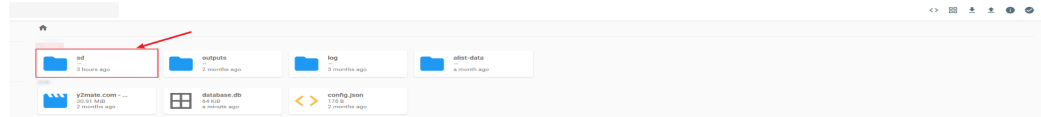
The following operations take user1 and user2 in [Using ECS as an NFS Server to Isolate Resources of Multiple Users](#) as an example. You need to create a Stable-Diffusion application for each of the two users and use the same SFS file system to initialize the custom model. For details, see [Application Creation and Deployment](#) and [Custom Models](#). In this case, the two users have shared the model and plug-in. The subsequent steps will solve the operation and configuration isolation problem.

Creating a Configuration File

Open the file management page of any user by referring to [step 4](#) in [Initializing a Model](#). After login, go to the `sd` directory. If the `sd` directory does not exist or

the **config.json** file does not exist in the directory, start the WebUI of any user by referring to [Application Use](#) and perform operations in this document again.

Figure 12-38 Entering the sd directory



Locate the **config.json** file, copy it to the **sd** directory, and rename it to **config_user1.json**. Repeat for user2, naming the copied file to **config_user2.json**.

Figure 12-39 Copying the config.json File

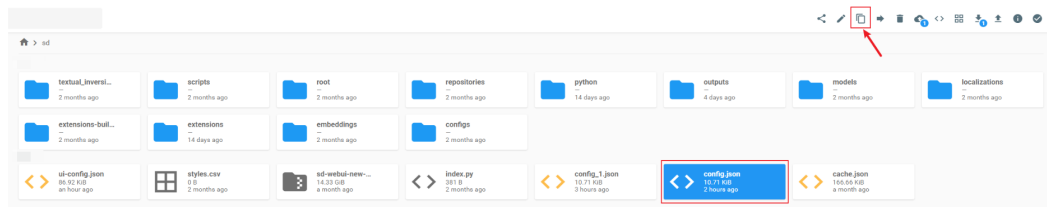
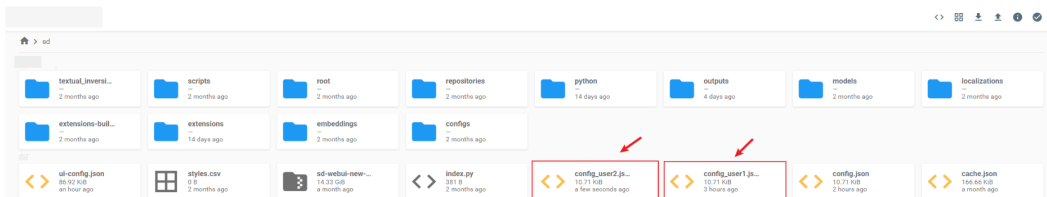


Figure 12-40 Creating the config_user1.json and config_user2.json files



Modifying Environment Variables and Using the New Configuration File

In [Creating a Configuration File](#), you have created a configuration file for each user. Now, modify environment variables to enable different users to use their own configuration files. Configure the following environment variables for the Stable-Diffusion function by referring to [Environment Variables](#) and save the settings (you can set the environment variables together with [API Access](#) and [Enabling WebUI Authentication](#)). The following are the environment variables of user1 and user2.

Table 12-4 Environment variables of user1 and user2

Key	Value
EXTRA_ARGS	--ui-settings-file=/mnt/auto/sd/config_user1.json
EXTRA_ARGS	--ui-settings-file=/mnt/auto/sd/config_user2.json

Changing the Path for Saving Images

After the preceding configurations are complete, the two users can share models and plug-ins without affecting each other. To further isolate the inference results of different users, you can change the result save path in the settings.

12.6.5 Using a Dedicated APIG Trigger

The Stable-Diffusion application created in the application center uses the shared APIG trigger. Resources such as bandwidth are shared by all shared APIG users. Therefore, if your service has higher requirements, you can use the dedicated APIG trigger. The procedure is as follows:

Buying a Dedicated Gateway

Purchase a dedicated APIG gateway based on your service requirements. For details, see [Buying a Gateway](#). Note that you need to enable public access to the WebUI and set the public access bandwidth as required.

Configuring a Dedicated APIG Trigger

- Step 1** Go to the details pages of the two functions by referring to [step 1 in Mounting an Application](#). The operations for the two functions are the same. This section uses the **stable-diffusion** function as an example.
- Step 2** On the function details page, choose **Configuration > Triggers > Create Trigger**. Select **API Gateway (Dedicated Gateway)** for **Trigger Type**, the purchased instance for **API Instance**, different groups for the two functions, **None** for **Security Authentication**, and **HTTPS** for **Protocol**. Set **Timeout** to **60,000** and retain the default values for other parameters, and click **OK**.
- Step 3** Delete the shared APIG trigger created by the system by default.

----End

Modifying the Trigger Backend Timeout

- Step 1** Modify the value of **backend_timeout** to **600,000** by referring to [Configuring Gateway Parameters](#).
- Step 2** Go to the details pages of the two functions by referring to [step 1 in Mounting an Application](#). The operations for the two functions are the same. This section uses the **stable-diffusion** function as an example.
- Step 3** On the function details page, choose **Configuration > Triggers**, and click the trigger name to go to the API management page.
- Step 4** Click **Edit > Next** to go to the **Define Backend Request** step, change the value of **Timeout (ms)** to the required value (the maximum value is 600,000, that is, 600 seconds), and click **Finish**.
- Step 5** After the modification is complete, return to the API management page, click **Publish** in the upper right corner, and then click **Publish**.

----End

Binding a Domain Name

Bind domain names to the APIG groups of your two functions by referring to [Configuring the Domain Name for Calling APIs](#).

12.7 Disclaimer

1. All projects used in this application, such as [Stable-Diffusion](#), [Stable-Diffusion-WebUI](#), and the [Image Building Project](#), are open-source community projects. Huawei Cloud only provides computing power support.
2. This application is only for reference and learning purposes. To use it in a production environment, optimize it by referring to the image building project. If you encounter any issues while using FunctionGraph, submit a service ticket. For questions related to open source projects, seek help from the open source community.
3. A gateway will be created on APIG upon application deployment. Bind a custom domain name as prompted and use it to access the WebUI.

13 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
```

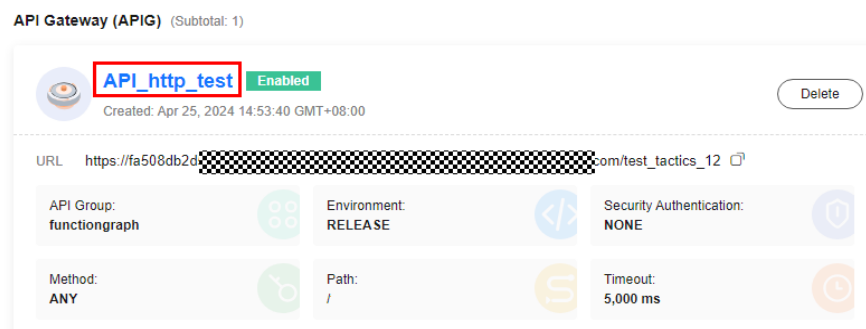
Creating an HTTP function and uploading code

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.

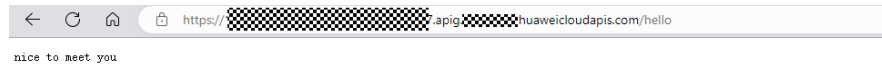
Figure 13-1 APIG trigger



Invocation test

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

Figure 13-2 Request result



14 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

```

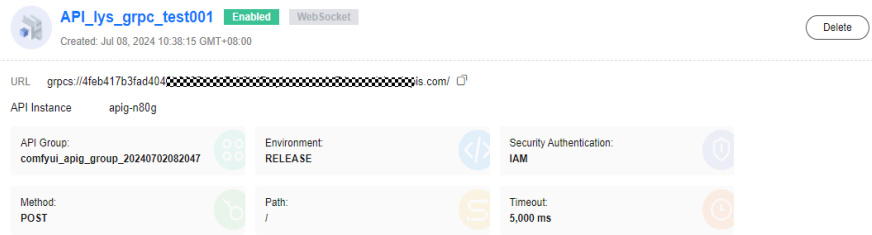
3. Creating an HTTP function and uploading code

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.

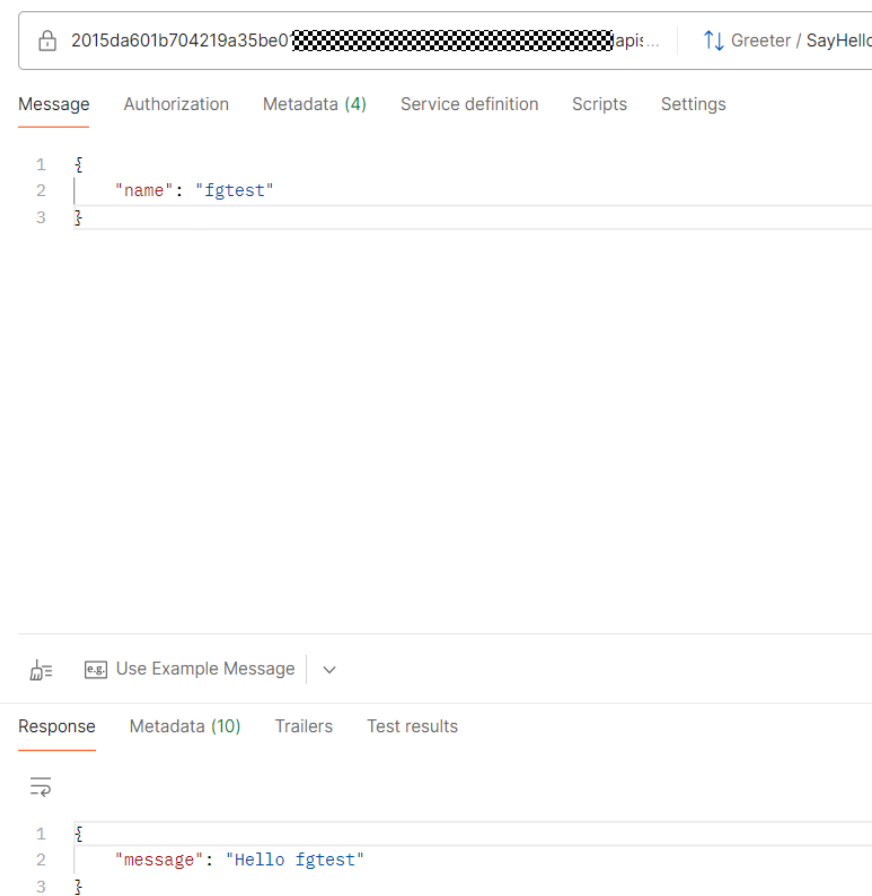
Figure 14-1 APIG trigger



5. Invocation test

Use Postman to debug gRPC requests.

Figure 14-2 gRPC request result



15 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](#).