FunctionGraph

# Best Practice

**Issue** 01

**Date** 2026-01-12

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

### Trademarks and Permissions


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

### Notice

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

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

## 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

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.

## Performance Optimization and Security

**Table 1-1** Performance optimization and security practices

| Practice | Description |
| --- | --- |
| Performance Optimization | Introduces performance optimization practices including cold start and function execution improvement and provides guidance to help you build more efficient and stable applications on FunctionGraph. |
| Cold Start Optimization | Optimize the cold start of a function to improve user experience in building a serverless architecture. |
| Security Best Practices | Improve the overall security capability of FunctionGraph. |

## Data Processing Practices

**Table 1-2** Data processing best practices

| Practice | Description |
| --- | --- |
| Using FunctionGraph to Convert DIS Data Format and Store the Data to CloudTable | Use functions and DIS to collect real-time IoT data streams, convert the format of the collected data, and store the data to CloudTable Service. |

| Practice | Description |
| --- | --- |
| Uploading Files Using APIs in FunctionGraph | Use Node.js and Python as examples to describe how to configure a backend parsing function and use APIG to upload files from devices to cloud servers.<br>This feature applies to web and app scenarios, such as reporting service run logs and uploading web app images. |
| Converting Device Coordinate Data in IoTDA | Use functions and IoTDA to transfer data reported by IoT devices and device status changes to FunctionGraph to trigger function running and convert coordinates (from WGS84 coordinates to GCJ02 coordinates).<br>It is applicable to scenarios such as processing device-reported data for storage in OBS, structuring and cleansing reported data before storing it in databases, and triggering event notifications based on device status changes. |
| Using FunctionGraph to Encrypt and Decrypt Files in OBS | Use a function and an **OBS Application Service** trigger to encrypt and decrypt files in OBS. |
| Identifying Abnormal Service Logs in LTS and Storing Them in OBS | Use LTS to configure a function for extracting alarm logs, identify abnormal log data in LTS, store the data in an OBS bucket, and use SMN to push alarm SMS messages and emails to service personnel. |
| Using FunctionGraph to Filter Logs in LTS in Real Time | Configure a function to extract log data, analyze and filter key information, and transfer the data to LTS. |
| Using FunctionGraph to Rotate Images Stored in OBS | Use a function flow to automatically rotate images in OBS.<br>Function flows are available in **CN East-Shanghai1** and **AP-Singapore**. |
| Using FunctionGraph to Compress and Watermark Images | Use a function flow to automatically compress and watermark images.<br>Function flows are available in **CN East-Shanghai1** and **AP-Singapore**. |

## Functional Application Practices

**Table 1-3** FunctionGraph functional application best practices

| Practice | Description |
| --- | --- |
| **Using FunctionGraph and CTS to Identify Login and Logout Operations from Invalid IP Addresses** | Configure functions for obtaining, analyzing, and processing cloud service resource operation information using CTS, and then push alarm SMS messages and emails using SMN to notify service personnel of handling the alarms. |
| **Using FunctionGraph Functions As the Backend to Implement APIG Custom Authorizers** | Quickly create an API whose backend service is FunctionGraph and call the API using custom authorizer. |
| **Using FunctionGraph HTTP Functions to Process gRPC Requests** | Process gRPC requests in FunctionGraph.<br>Currently, only the **LA-Santiago** region is supported. |
| **Using a Java Function and Log4j2 to Print Logs** | Use Java functions to configure Log4j2 to print logs. |
| **Using FunctionGraph to Deploy Stable Diffusion for AI Drawing** | Deploy Stable-Diffusion applications in the application center of FunctionGraph and provides multiple methods for customizing AI drawing applications. |
| **Deploying an MCP Server Using FunctionGraph** | Deploy popular open-source MCP server applications in one click in FunctionGraph and provide services accessible from the Internet through APIG. |

## Function Building Practices

**Table 1-4** Function building best practices

| Practice | Description |
| --- | --- |
| **Building an HTTP Function Using an Existing Spring Boot Project** | Deploy a Spring Boot application as an HTTP function on FunctionGraph. |
| **Building an HTTP Function Using Go** | Deploy a Go application as an HTTP function on FunctionGraph. |
| **Using FunctionGraph to Access RDS for MySQL** | This section describes how to access RDS for MySQL from FunctionGraph and query data, and provides sample code for testing. |

# 2 Performance Optimization and Security Practices

## 2.1 Performance Optimization

As serverless technologies become increasingly popular, performance optimization is crucial for better efficiency and user experience.

This section focuses on optimizing FunctionGraph, covering **cold start** and function execution improvements. It offers practical tips to achieve optimal performance in various scenarios, helping you build more efficient and stable applications.

### Code Optimization

- Write idempotent code. This ensures that functions handle repeated events in the same way.
- Use connection pools (HTTP, database, and Redis) properly to reduce the cold start overhead of creating new connections.
- Avoid reinitializing variables on each invocation. Use global static variables or singletons. For middleware like Redis or Kafka, initialize clients in an init method or as global variables to reduce cold start overhead.
- Enhance client retry mechanisms. When a function invocation returns non-200 status codes (such as 500, 429, 504), implement retry logic based on service needs to improve reliability.
- Log appropriately. When accessing third-party services, Huawei Cloud services, or performing key operations in FunctionGraph, log the actions for troubleshooting, performance optimization, and service analysis.

### Performance Stress Testing

Performance stress testing is crucial for choosing the optimal configuration. During the testing, use platform metrics, logs, and call chains to analyze performance data and optimize configuration. For more details on observable metrics, see **Function Metrics**.

### Code Simplification and Image Downsizing

FunctionGraph downloads function code during cold start, which affects startup time. Larger code files extend download time and increase startup duration.

For custom image-based functions, larger images also prolong startup. To reduce cold start time, optimize applications by removing unnecessary code and reducing third-party library dependencies. For example, run the command **npm prune** in Node.js and **autoflake** in Python.

In addition, clearing test cases, unused binaries, and data files in third-party libraries can also reduce code download and decompression time.

### Larger Memory

Increasing function memory can enhance CPU performance and accelerate startup and execution. Monitor execution time to assess the effect of various memory settings and select the best configuration.

For details, see **Function Monitoring**. For details about how to configure the memory, see **Configuring Basic Settings**.

### Public Dependencies

When developing functions, especially in Python, third-party dependencies are often required. During cold start, large dependencies can increase startup time.

FunctionGraph provides public dependencies that are pre-loaded on execution nodes to mitigate this issue. Therefore, prioritize using public dependencies and reduce the use of private ones. For details, see **Configuring Dependency Packages**.

### Reserved Instances

After reserved instances are created for a function, the code, dependencies, and initializer of the function are automatically loaded. Reserved instances are always alive in the execution environment, eliminating the influence of cold starts on your services. For details about how to configure reserved instances, see **Reserved Instance Management**.

### Function Initializer

For functions that are invoked frequently, initializing HTTP or database connections in the Initializer can make each execution much faster. For details about how to configure the function initializer, see **Configuring Initialization**.

## 2.2 Cold Start Optimization

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 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**.

## 2.3 Security Best Practices

Security is a shared responsibility between Huawei Cloud and you. Huawei Cloud is responsible for the security of cloud services. You need to properly use the security capabilities provided by cloud services to protect data. For details, see .**Shared Responsibility**.

This document provides security best practices for FunctionGraph to improve overall security capabilities. By following this guide, you can continuously assess the security status of functions, effectively integrate various security features of FunctionGraph, strengthen its security defense, and ensure data stored in FunctionGraph is not leaked or tampered with, while also safeguarding data transmission.

Make security configurations from the following dimensions to meet your service needs.

### Trusted Code and Dependencies

- Before deploying function code, you are advised to use **CodeArts Check** to perform static scanning and vulnerability analysis to ensure code security.
- Use dependency libraries from reliable sources and update them periodically. Do not use third-party libraries with known vulnerabilities.

### Sensitive Information Protection

- If your code or configuration contains sensitive information, such as AK/SK, token, and password, **encrypt environment variables**. Otherwise, the information may be displayed in plaintext on the UI or in the API return result, causing sensitive information leakage.
- Anonymize the privacy data (such as logs and personal information) during function processing. Do not print logs in plaintext to prevent sensitive information leakage.
- FunctionGraph provides temporary download links with expiration. You need to protect these links to prevent code leakage.

## Fine-grained Permission Control and Identity Authentication

- When configuring **agency permissions**, AKs, and SKs for functions through Identity and Access Management (IAM), comply with the principle of least privilege to ensure that the functions can access only specified resources. For example, you can restrict the read and write permissions of a function on a specific OBS bucket to prevent unauthorized access.
- When configuring an APIG trigger, you are advised to enable IAM or custom authorizer to ensure that only authorized requests can trigger function execution. In addition, you can use APIG to implement request throttling to prevent resource exhaustion caused by malicious requests.

## VPC Configuration

To access resources in a Virtual Private Cloud (VPC), such as RDS, you are advised to **configure VPC access** for your function to ensure that it can communicate with other cloud services in an isolated network environment.

## Version Management

FunctionGraph supports **version management**. You are advised to create multiple versions for each function and use stable versions in the production environment. In addition, you can use **aliases** to switch versions in case of security issues.

# 3 Data Processing Practices

## 3.1 Using FunctionGraph to Compress Images in OBS

### Introduction

This practice is applicable to compressing a single image or a batch of images. High-quality images consume significant storage/bandwidth, slowing website/app loading. Using OBS and FunctionGraph, you can build an image compression solution to automatically process images in buckets, optimizing storage and resource efficiency.

### Constraints

- **OBS Application Service** trigger is available only in **CN North-Beijing4**, **CN North-Ulanqab1**, and **CN East-Shanghai1**. When creating a function or an OBS bucket, select one of the preceding regions.
- Ensure that the created function and OBS bucket are in the same region.

### Resource and Cost Planning

**Table 3-1** lists the resources and costs required for using FunctionGraph to compress images in OBS.

**Table 3-1** Resource and cost planning

| Resource | Description | Billing |
| --- | --- | --- |
| OBS | • Product type: object storage<br>• Region: **CN North-Beijing4**<br>• Storage policy: Single-AZ storage<br>• Storage class: Standard<br>• Bucket policy: Private<br>• Quantity: 2 | • Billing mode: Pay-per-use<br>• For details about billing items, see **Object Storage Service (OBS) Pay-per-Use Billing**. |
| FunctionGraph | • Function type: Event function<br>• Region: **CN North-Beijing4**<br>• Quantity: 1 | • Billing mode: Pay-per-use<br>• The first 1 million invocations are free of charge in a month. For details about the billing items, see **Pay-per-Use Billing**. |

## Procedure

The following table describes how to use FunctionGraph to compress images in OBS.

**Table 3-2** Procedure

| Step | Description |
| --- | --- |
| **Step 1: Creating Two OBS Buckets** | Create two OBS buckets. The source bucket is used to store the original image, and the target bucket is used to store the compressed image. |
| **Step 2: Creating a Cloud Service Agency** | Create a cloud service agency to authorize FunctionGraph to access other cloud services so that FunctionGraph can work with OBS. |
| **Step 3: Creating an Image Compression Function** | Create a function from scratch, configure the code environment, and create an **OBS Application Service** trigger to automatically compress images uploaded or updated in the OBS source bucket. |

| Step | Description |
| --- | --- |
| Step 4: Verifying Image Compression | Verify that the image is compressed in the target bucket. |

## Step 1: Creating Two OBS Buckets

**Step 1** Log in to the OBS console, choose **Object Storage**.

**Step 2** Click **Create Bucket**.

**Step 3** On the displayed **Create Bucket** page, set the OBS source bucket parameters by referring to Table 3-3.

**Table 3-3** OBS source bucket configuration

| Parameter | Requirements | Example Value |
| --- | --- | --- |
| Region | Mandatory<br>Region where the bucket is located. Select a region close to your service to reduce network latency and improve access speed. After a bucket is created, its region cannot be changed. Currently, the **OBS Application Service** trigger supports only **CN North-Beijing4**, **CN North-Ulanqab1**, and **CN East-Shanghai1**. | CN North-Beijing4 |
| Bucket Name | Mandatory.<br>Bucket name, which must be globally unique. After a bucket is created, its name cannot be changed. | your-bucket-input |
| Data Redundancy Policy | Mandatory.<br>• **Multi-AZ storage**: Data is stored in multiple AZs to achieve higher reliability.<br>• **Single-AZ storage**: Data is stored in a single AZ, which costs less.<br>• After a bucket is created, its data redundancy policy cannot be changed. | Single-AZ storage |

| Parameter | Requirements | Example Value |
| --- | --- | --- |
| Storage Class | Mandatory.<br>• **Standard**: For storing a large number of hot files or small files that are frequently accessed (multiple times per month on average) and require fast access. Both single-AZ and multi-AZ storage are supported.<br>• **Infrequent Access**: For storing data that is less frequently accessed (less than 12 times per year on average), but when needed, the access has to be fast. Both single-AZ and multi-AZ storage are supported.<br>• **Archive**: For archiving data that is rarely accessed (once a year on average) and does not require fast access. Only single-AZ storage is supported. | Standard |
| Bucket Policy | Mandatory.<br>Controls read and write permissions for the bucket.<br>• **Private**: Only users granted permissions by the bucket ACL can access the bucket.<br>• **Public Read**: Anyone can read objects in the bucket.<br>• **Public Read/Write**: Anyone can read, write, or delete objects in the bucket. | Private |
| Enterprise Project | Mandatory.<br>Enterprise projects let you manage cloud resources by projects. The default project is **default**. | default |

| Parameter | Requirements | Example Value |
| --- | --- | --- |
| Properties | Optional.<br>For details, see **Creating a Bucket**.<br>• **Direct Reading**: allows you to download data from the **Archive** storage class without restoring them in advance. Direct reading is a billable function.<br>• **Server-Side Encryption**: the OBS server encrypts the objects uploaded from the client before storing them. If this feature is enabled, you must specify an encryption key.<br>• **WORM**: When enabled, you can configure a retention policy for the current bucket. The object version which the retention policy is applied to cannot be deleted within a specified period.<br>• **Tags**: Tags are used to identify and classify OBS buckets. | • **Direct Reading**: disabled<br>• **Server-Side Encryption**: disabled<br>• **WORM**: disabled<br>• Tags: - |

**Step 4** Repeat **Step 3** to create the target bucket. Set the **Bucket Name** to **your-bucket-output** and keep other parameters the same as those of the source bucket.

**Step 5** View **your-bucket-input** and **your-bucket-output** in the bucket list.

**----End**

## Step 2: Creating a Cloud Service Agency

**Step 1** Log in to **IAM** console. In the navigation pane, choose **Agencies**. On the displayed page, click **Create Agency** in the upper right corner.

**Step 2** Set the following parameters:

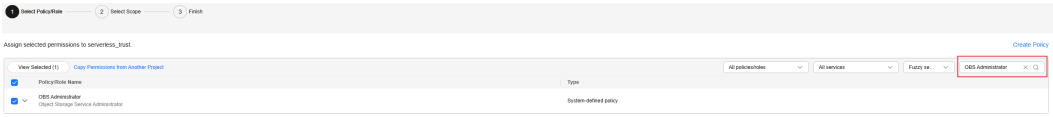
- **Agency Name**: Enter **serverless_trust**.
- **Agency Type**: Select **Cloud service**.
- **Cloud Service**: Select **FunctionGraph**.
- **Validity Period**: Select **Unlimited**.
- Description: Retain the default value.

**Step 3** Click **OK**. The system displays a message indicating that the creation is successful. Click **Authorize**.

**Step 4** On the **Select Policy/Role** page, search for and select the **OBS Administrator** policy, and click **Next**.

**Figure 3-1** Selecting a policy



**Step 5** Set the minimum authorization scope. Select **All resources** for **Scope** and click **OK**.

NOTE

The **OBS Administrator** policy does not support specifying specific region or project resources.

**Step 6** The system displays a message indicating that the authorization is successful. Click **Finish** to return to the agency list. If **serverless_trust** is displayed in the list, the agency is created successfully.

**----End**

## Step 3: Creating an Image Compression Function

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

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

**Step 3** Select **Create from scratch**, set the function information by referring to **Table 3-4**, and click **Create Function**.

**Table 3-4** Configuring function parameters

| Parameter | Requirements | Example Value |
| --- | --- | --- |
| Function Type | Mandatory.<br>• **Event Function**: triggered by triggers.<br>• **HTTP Function**: triggered once HTTP requests are sent to specific URLs. | Event Function |
| Region | Mandatory.<br>Region where the code is deployed. The region must be the same as that of the OBS bucket. | CN North-Beijing4 |
| Function Name | Mandatory.<br>Function name, which contains letters, digits, underscores (_), and hyphens (-). It must start with a letter and end with a letter or digit. The length cannot exceed 60 characters. | fss_examples_image_thumbnail |

| Parameter | Requirements | Example Value |
| --- | --- | --- |
| Enterprise Project | Mandatory.<br>Enterprise project to which the function is added. The enterprise project must be the same as that of the OBS bucket. The default enterprise project is **default**. | default |
| Agency | This parameter is optional but mandatory in this practice.<br>Name of the agency used by FunctionGraph to access other cloud services. Select the agency created in **Step 2: Creating a Cloud Service Agency**. | serverless_trust |
| Runtime | Mandatory.<br>Development language and language version of the function. CloudIDE supports Node.js, Python, and PHP only. | Python3.6 |

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

1. Download the sample code **fss_examples_image_thumbnail_eg.zip**.
2. On the **Code** tab page, select **Upload** > **Local ZIP**, add the downloaded **fss_examples_image_thumbnail_eg.zip** file, and click **OK**. The code is automatically deployed.
3. Click **Add** at the bottom of the page, add the public dependency **package pillow-7.1.2**, retain the default version **1**, and click **OK**.
4. On the **Configuration** > **Basic Settings** tab page, modify the following configuration:
   - For **Execution Timeout**, enter **40**.
   - For **Memory**, select **256**.

   Click **Save**.
5. Choose **Configuration** > **Environment Variables**, click **Edit Environment Variable**. In the dialog box that is displayed, click **Add**, add information in **Table 3-5**, and click **OK**.

**Table 3-5** Environment variables

| Key | Value | Description |
| --- | --- | --- |
| output_bucket | your-bucket-output | Name of the OBS bucket where compressed images are stored. |
| obs_endpoint | obs.cn-north-4.myhuaweicloud.com | OBS endpoint in **CN North-Beijing4** (For other regions, see **Regions and Endpoints**.) |

6. On the **Configuration** > **Triggers** tab page, click **Create Trigger**. In the displayed dialog box, configure basic information by referring to **Table 3-6** and click **OK**.

**Table 3-6** Configuring trigger parameters

| Parameter | Requirements | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Add an **OBS Application Service** trigger to trigger the function when an operation is performed on an OBS bucket. | OBS Application Service |
| Trigger Name | Mandatory.<br>Name of the trigger to be created. Only letters, digits, underscores (_), and hyphens (-) are allowed. The value cannot start with a digit or hyphen (-). Maximum length: 128 characters. | Image |
| Bucket Name | Mandatory.<br>Select the created OBS bucket to store the original images. | your-bucket-input |
| Event Type | Mandatory.<br>Triggering event type. In this practice, the function is triggered by uploading or updating bucket objects. | Create or override bucket objects via UI or Put request Create or override bucket objects via Post request |
| Object Name Prefix | Optional.<br>Enter a keyword for limiting notifications to those about objects whose names start with the matching characters. This limit can be used to filter the names of OBS objects. | Leave this parameter blank. |
| Object Name Suffix | Optional.<br>Enter a keyword for limiting notifications to those about objects whose names end with the matching characters. This limit can be used to filter the names of OBS objects. | Leave this parameter blank. |

| Parameter | Requirements | Example Value |
| --- | --- | --- |
| Object Name Encoding | Mandatory.<br>Specifies whether to encode the object name. | Enabled by default. |

**----End**

## Step 4: Verifying Image Compression

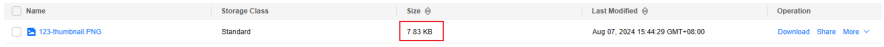
**Step 1** Log in to the **OBS console**, click the **your-bucket-input** bucket. The **Objects** tab page is displayed.

**Step 2** Click **Upload Object**, set **Storage Class** to **Standard**, and upload an image to be compressed. After the upload is successful, the page shown in **Figure 3-2** is displayed.

**Figure 3-2** Uploading an image


**Step 3** Go to the **Objects** page of the **your-bucket-output** bucket and view the size of the compressed image.

**Figure 3-3** Viewing compressed images




**NOTICE**

To avoid unnecessary storage fees, you can delete the images stored in the two OBS buckets as required after the practice. Deleted data cannot be restored. Exercise caution when performing this operation.

**----End**

# 3.2 Using FunctionGraph to Watermark Images in OBS

## 3.2.1 Introduction

The best practice for FunctionGraph guides you through image watermarking based on a function. (**OBS Application Service** trigger is available only in **CN North-Beijing4**, **CN North-Ulanqab1**, and **CN East-Shanghai1**.)

### Scenarios

- Upload images to a specified OBS bucket.

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

  NOTE

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

## Procedure

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

  NOTE

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

  1. Two OBS buckets (respectively used for storing uploaded and processed images)
  2. An image watermarking function
  3. An **OBS Application Service** trigger, which is used to associate a function with an OBS bucket.

# 3.2.2 Preparation

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

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

## Creating OBS Buckets

**Precautions**

- The function and the source and destination buckets for storing images must be in the same region.
- Use two different OBS buckets. If only one bucket is used, the function will be executed infinitely. (When an image is uploaded to the bucket, the function is triggered to process the image and store the processed image into the bucket again. In this way, the function executes endlessly.)

**Procedure**

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

**Step 2** On the **Create Bucket** page, set the bucket information.

- For **Region**, select a region.
- For **Data Redundancy Policy**, select **Single-AZ storage**.
- For **Bucket Name**: Enter a custom bucket name, for example, **bucket-input-fg**.

- For **Default Storage Class**, select **Standard**.
- For **Bucket Policies**, select **Private**.
- For **Server-Side Encryption**: select **Disable**
- For **Direct Reading**, select **Disable**.

Click **Create Now**.

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

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

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

**----End**

### Creating an Agency

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

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

**Step 3** Set the agency information.

- For **Agency Name**: Enter 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 the description.

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

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

**----End**

## 3.2.3 Building a Program

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

### Creating a Deployment Package

This example uses a Python function to watermark images. For details about function development, see **Developing Functions in Python**. **Figure 3-4** shows the sample code directory. The service code is not described.

**Figure 3-4** Sample code directory


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

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

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

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

    return 'OK'
```

## Creating a Function

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

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

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

**Step 3** Click **Create from scratch** and configure the function information.

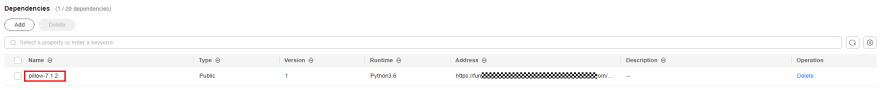
After setting the basic information, click **Create**.

- For **Function Type**, select **Event Function**.
- For **Function Name**: Enter a function name, for example, **fss_examples_image_watermark**.
- For **Agency**, select **serverless_trust** created in **Creating an Agency**.
- For **Runtime**, select **Python 3.6**.

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

**Figure 3-5** Adding a dependency



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

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

**Table 3-7** Environment variables

| Key | Value | Description |
| --- | --- | --- |
| obs_output_bucket | bucket-output-fg | Key: OBS bucket parameter for storing output watermark images, which is defined in the **index.py** file.<br>Value: OBS bucket created in **Creating OBS Buckets** for storing output watermark images. |
| obs_region | For example, **cn-north-4**. | Key: region of the OBS bucket where the output watermark image is stored, which is defined in the **index.py** file. It must be the same as the region where the function is located.<br>Value: region where the OBS bucket **obs_output_bucket** is located. For details, see **Regions and Endpoints**. |

**----End**

## 3.2.4 Adding an Event Source

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

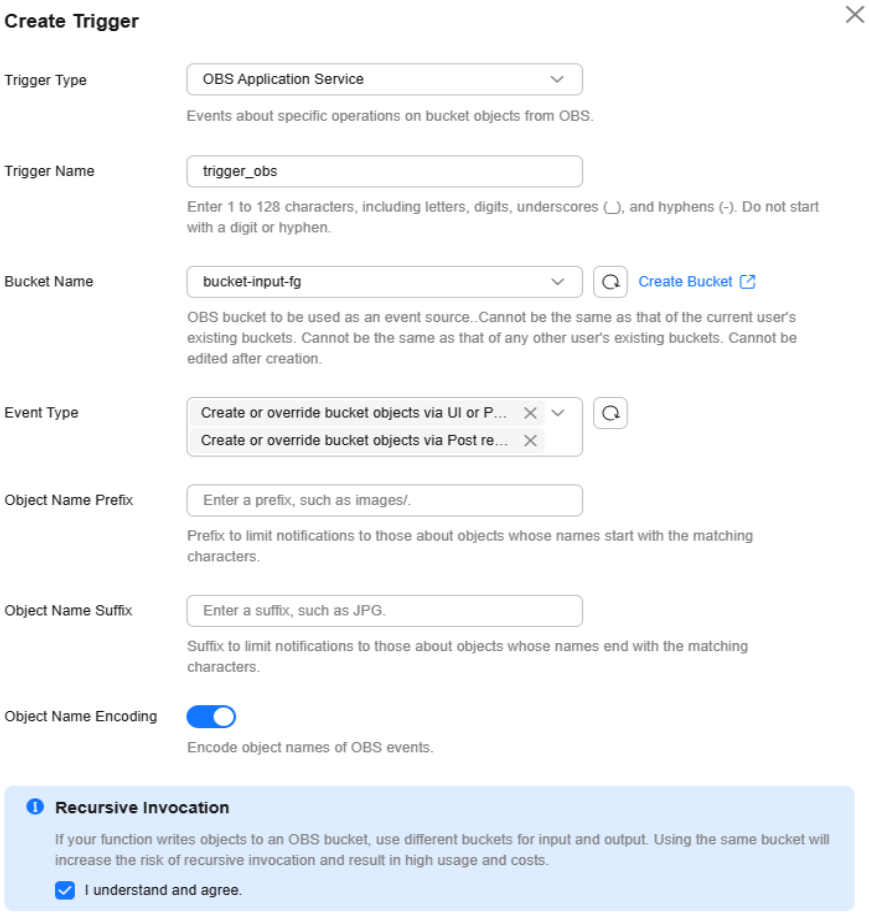
  

**Step 1** On the **fss_examples_image_watermark** page, click the **Triggers** tab and click **Create Trigger**.

**Step 2** Select **OBS Application Service** for **Trigger Type**, and set the trigger information, as shown in **Figure 3-6**.

- **Trigger Name**: Customize a trigger name.
- **Bucket Name**: Select **bucket-input-fg** created in **Creating OBS Buckets**.
- **Event Type**: Select **Create or override bucket objects via UI or Put request** or **Create or override bucket objects via Post request**.

**Figure 3-6** Creating an OBS Application Service trigger




**Step 3** Click **OK**.

NOTE

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

**----End**

## 3.2.5 Watermarking Images

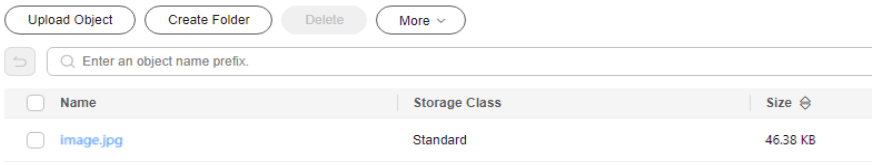
When an image is uploaded or updated to bucket **bucket-input-fg**, an event is generated to trigger the function. The function watermarks the image and stores the watermarked one into bucket **bucket-output-fg**.

## Uploading an Image to Generate an Event

Log in to the **OBS console**, go to the object page of the **bucket-input-fg** bucket, and upload the **image.jpg** image, as shown in **Figure 3-7**.

**Figure 3-7** Uploading an image




## Triggering the Function

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

The **Objects** page of the bucket **bucket-output-fg** displays the watermarked image **image.jpg**, as shown in **Figure 3-8**. In the **Operation** column, click **Download** to download the image and view the watermarking effect, as shown in **Figure 3-9**.

**Figure 3-8** Output image

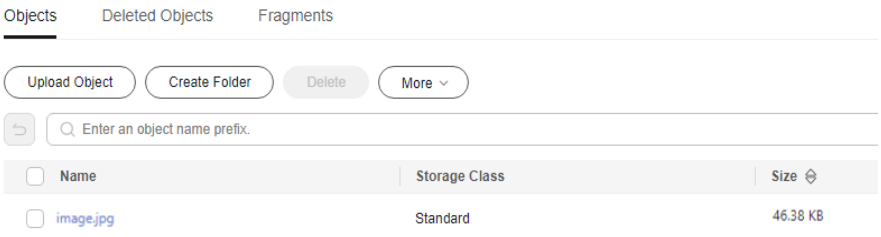



**Figure 3-9** Watermarked image

# 3.3 Using FunctionGraph to Convert DIS Data Format and Store the Data to CloudTable

## 3.3.1 Introduction

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

### Scenarios

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

### Procedure

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

## 3.3.2 Preparation

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

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

### Creating a VPC

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

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

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

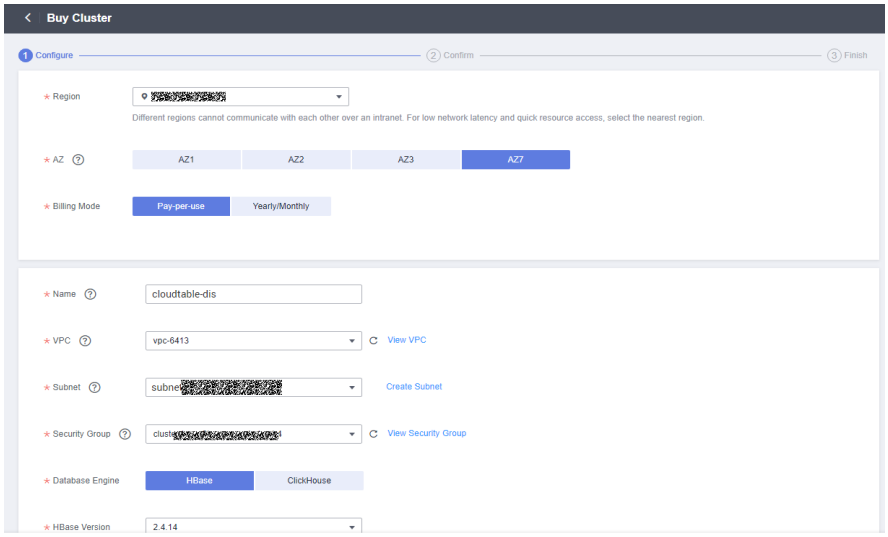
**Step 1** In the left navigation pane of the management console, choose **Analytics** > **CloudTable Service** to go to the CloudTable console. On the **Cluster Mode** page, click **Buy Cluster**.

**Step 2** Set the cluster information.

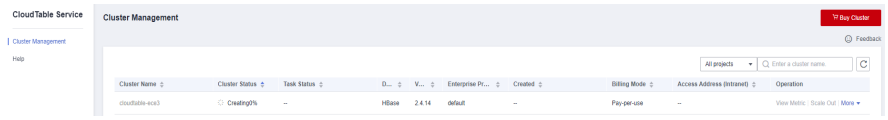
- **Region**: Use the default region.
- **Name**: Enter a name, for example, **cloudtable-dis**.
- **VPC**: Select **vpc-cloudtable** created in **Creating a VPC**.
- Retain the default values for other parameters.

**Figure 3-10** Buying a cluster




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

**Figure 3-11** Creating a cluster




**NOTE**

Creating a cluster takes a long time. You can check the creation progress according to **Figure 3-11**.

**----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**

# 3.3.3 Building a Program

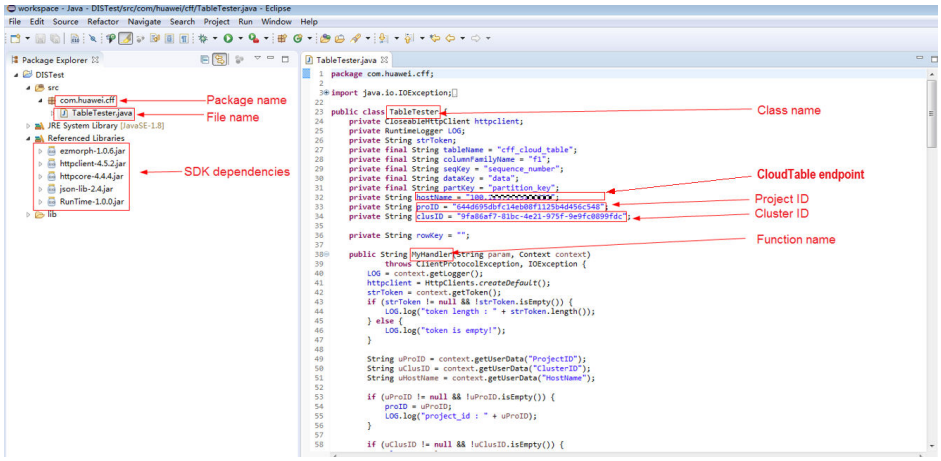
Download the **source code** and **program package** (including function dependencies) to create a function from scratch for converting DIS stream data formats.

## Creating a Project

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

Download the sample source code package **fss_examples_dis_cloudtable_src.zip**, decompress the file, and import it to Eclipse, as shown in **Figure 3-12**.

**Figure 3-12** Sample code




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

To obtain the project ID, perform the following steps:

1. Under the current login account in the upper right corner, choose **My Credentials**, as shown in **Figure 3-13**.
2. Obtain the project ID in the project list, as shown in **Figure 3-14**.

**Figure 3-13** My Credentials

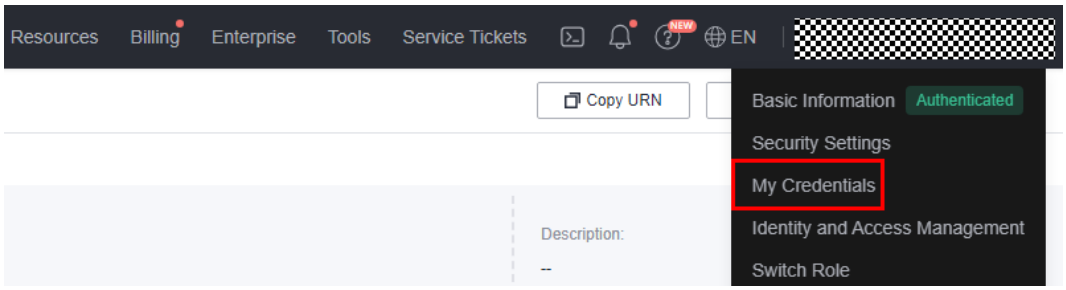
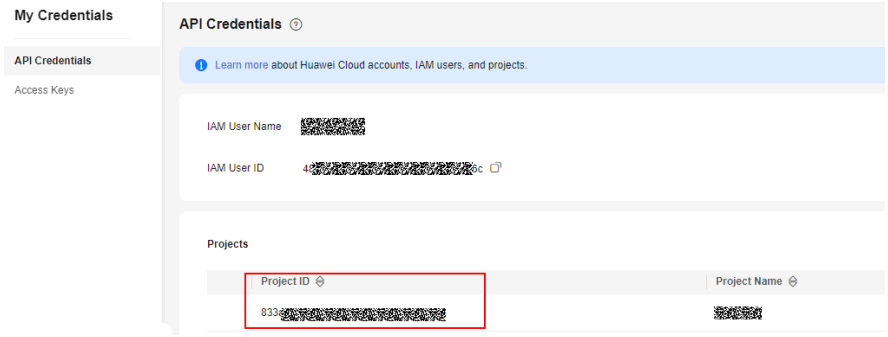



**Figure 3-14** Project ID




To obtain the cluster ID, perform the following steps:

1. Log in to the **CloudTable console**.
2. In the navigation pane, choose **Cluster Management**. Click cluster **cloudtable-dis** created in **Creating a Cluster**.
3. On the **cloudtable-dis** page that is displayed, find the cluster ID, as shown in **Figure 3-15**.

**Figure 3-15** Cluster ID


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

## Packaging the Code

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

**Figure 3-16** Exporting the code

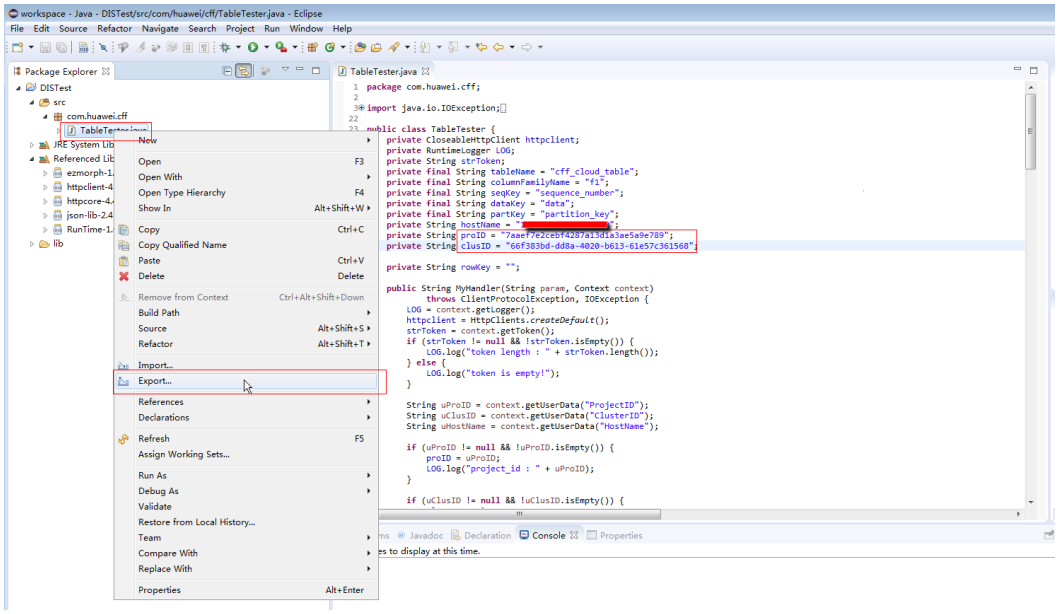


**Figure 3-17** Selecting a file type

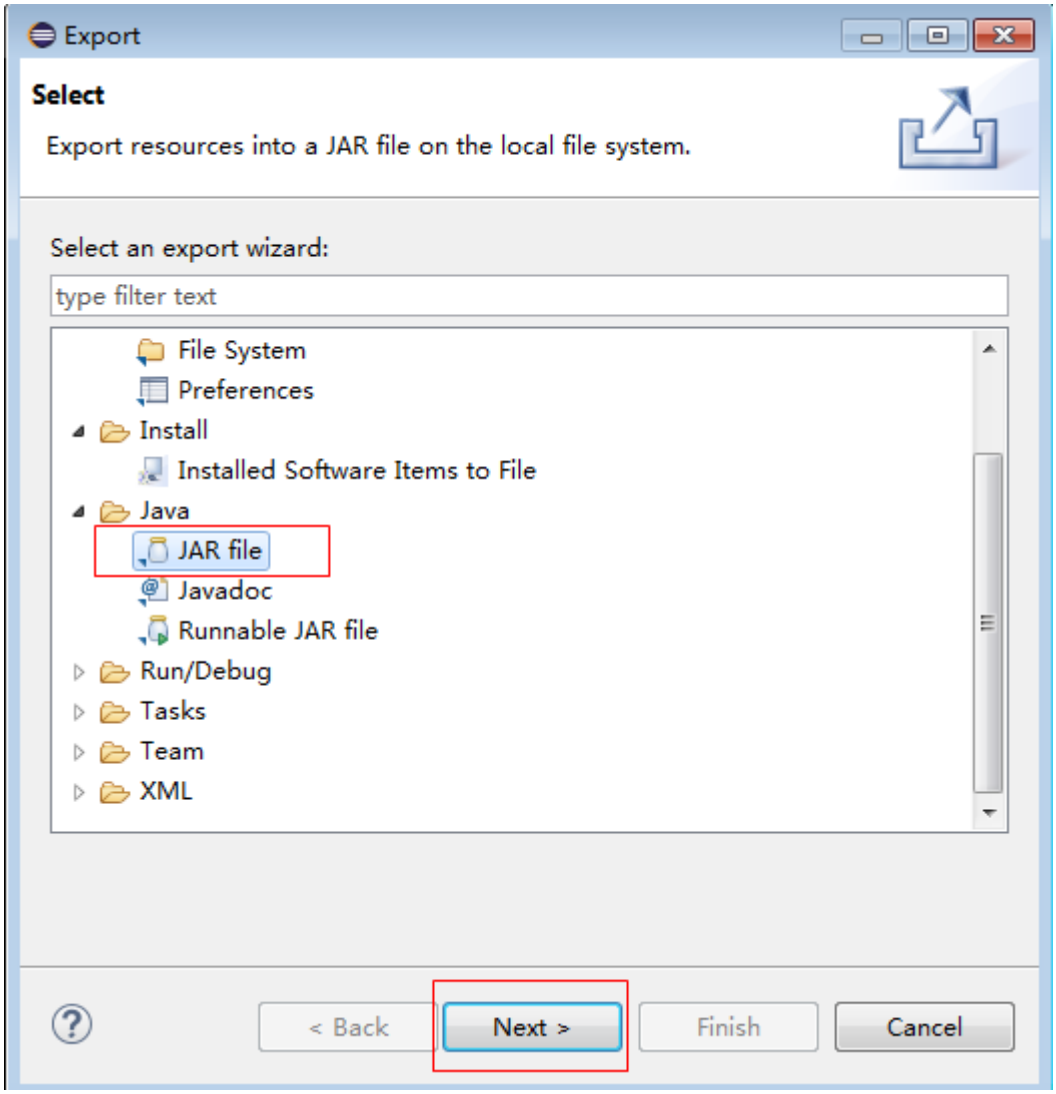
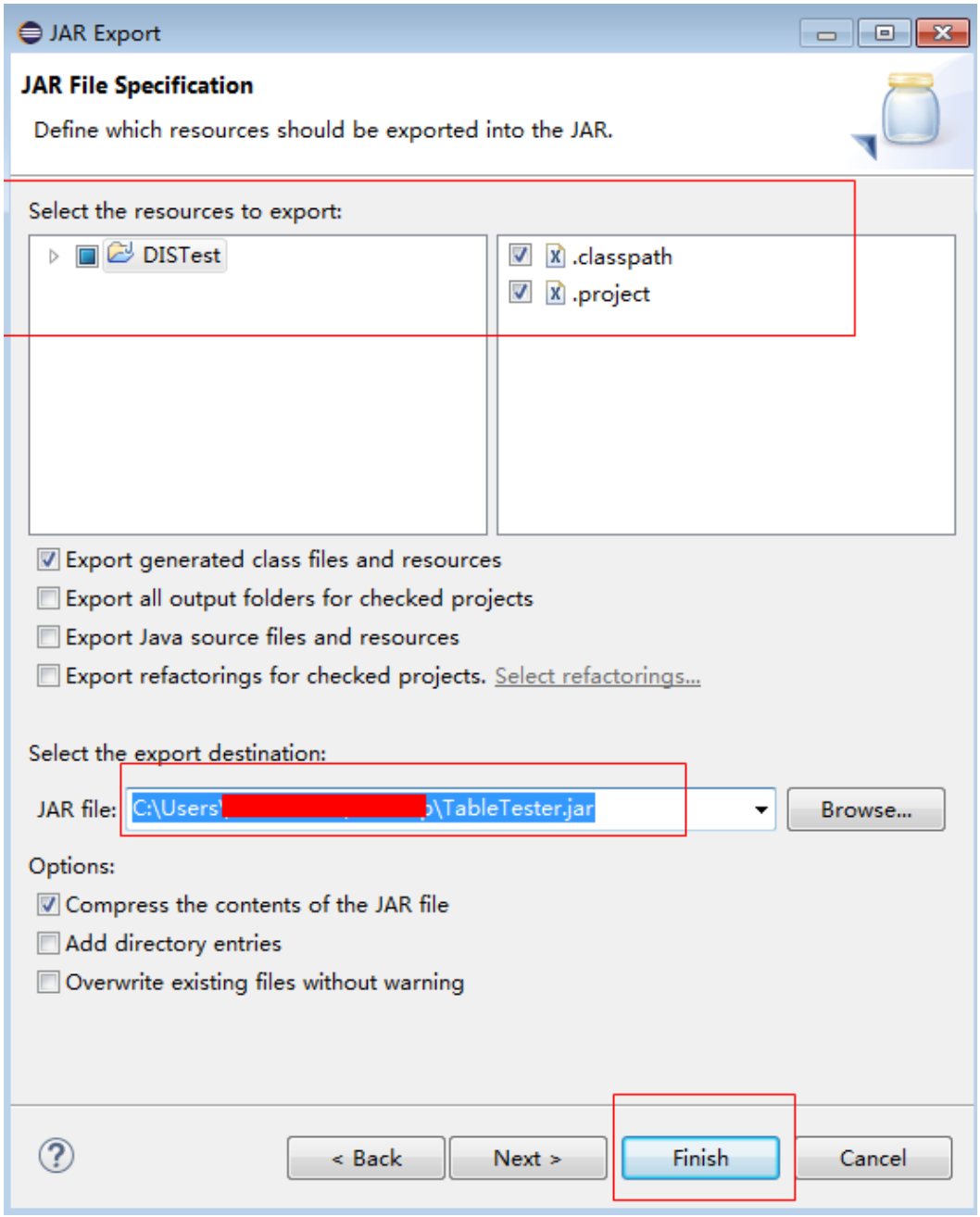


**Figure 3-18** Publishing the code file




Package the function dependencies by performing the following steps:

1. **Download program package** **fss_examples_dis_cloudtable.zip**, and decompress it, as shown in **Figure 3-19**.
2. Use **Table Tester.jar** to replace **DIS Test.jar**, as shown in **Figure 3-20**.
3. Package all of the files into **disdemo.zip**, as shown in **Figure 3-21**.

**Figure 3-19** File directory before replacement

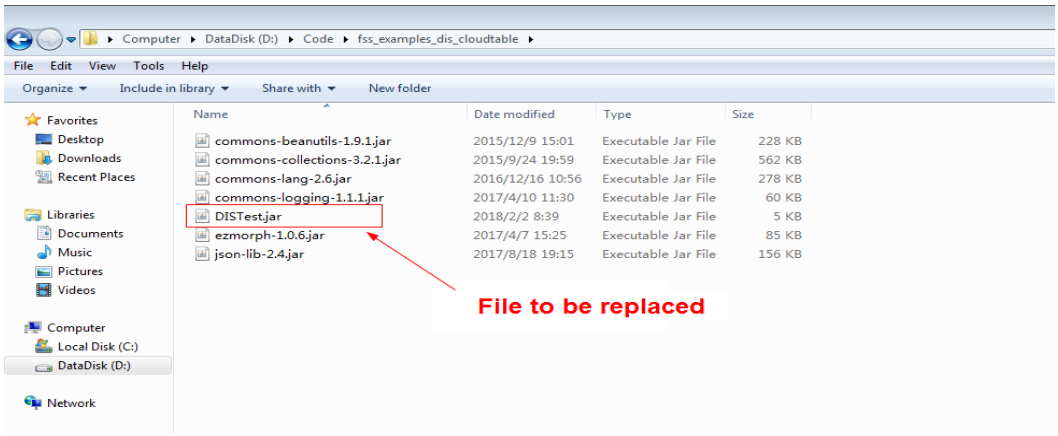



**Figure 3-20** File directory after replacement

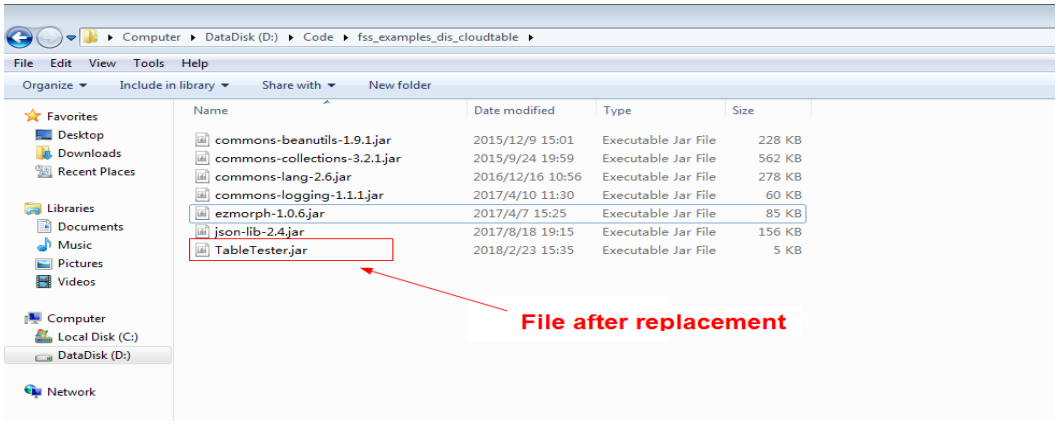
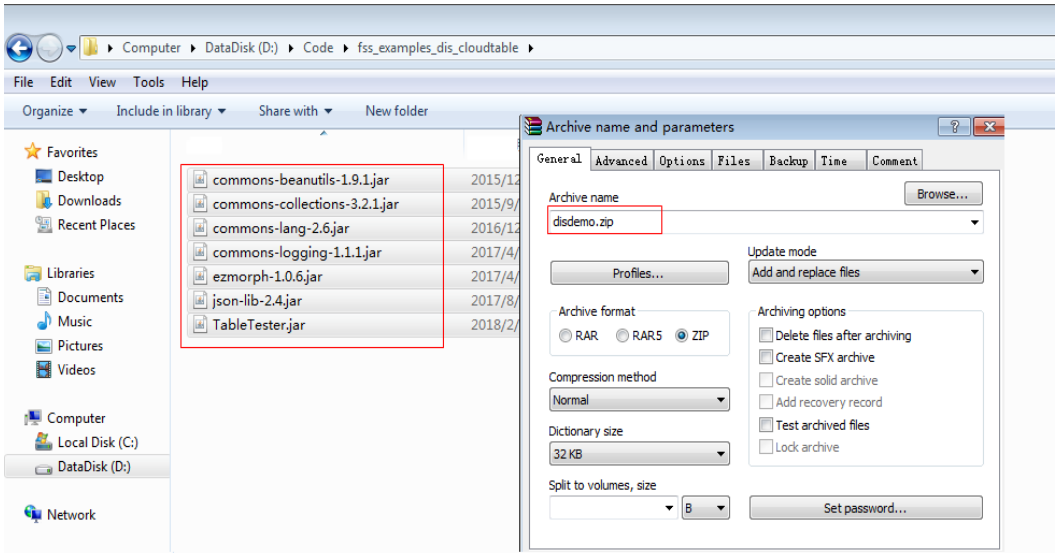



**Figure 3-21** Packaging the files in ZIP format




## 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**

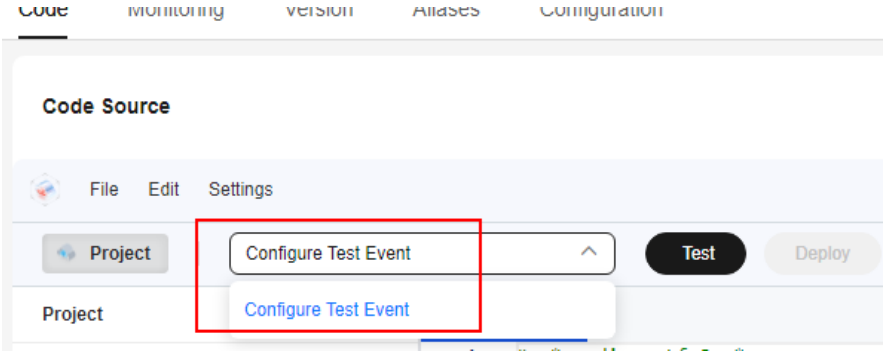
## 3.3.4 Adding an Event Source

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

**Step 1** On the **DISDemo** page, select **Configure Test Event** on the **Code** tab, as shown in **Figure 3-22**.

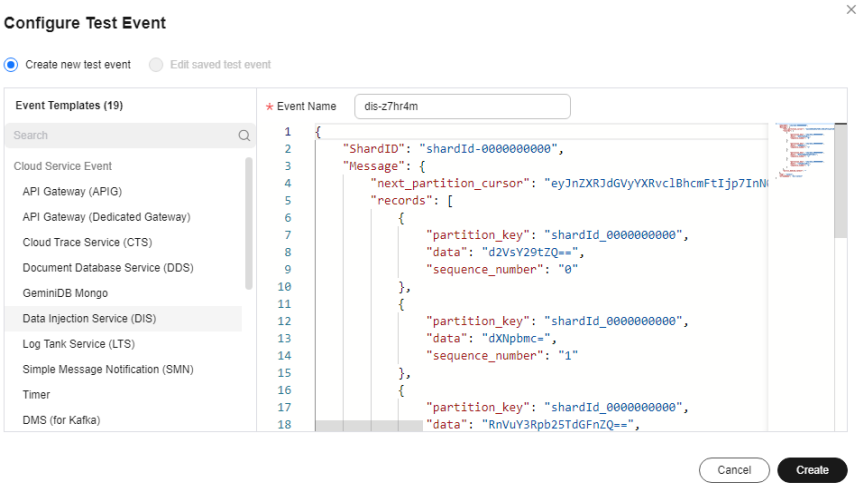
**Figure 3-22** Configuring a test event




**Step 2** In the **Configure Test Event** dialog box, set the test event information, as shown in **Figure 3-23**.

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

**Figure 3-23** Test event




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

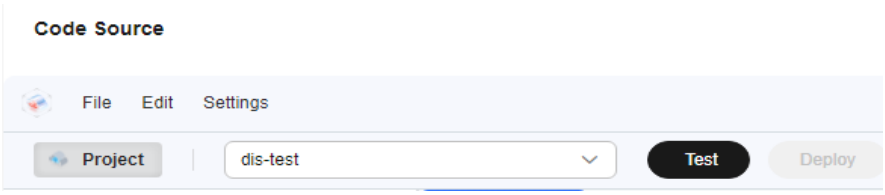
**----End**

## 3.3.5 Processing Data

Perform the following procedure to process simulated stream data:

**Step 1** On the **DISDemo** page, select test event **dis-test**, and click **Test** to test the function, as shown in **Figure 3-24**.

**Figure 3-24** Configuring a test event



**Step 2** After the function is executed successfully, check the logs shown in **Figure 3-25**. For all logs of the function, go to the **Logs** tab page.

**Figure 3-25** Function execution result


**----End**

# 3.4 Uploading Files Using APIs in FunctionGraph

## 3.4.1 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.

## 3.4.2 Resource Planning

**Table 3-8** Resource planning

| Product | Configuration Example |
| --- | --- |
| APIG | • Region: AP-Singapore<br>• Specification: dedicated gateway |
| FunctionGraph | • Region: AP-Singapore<br>• Billing mode: pay-per-use |

## 3.4.3 Procedure

This solution includes the following steps:

1. Create a function to receive and parse uploaded files.
2. Buy a dedicated gateway and bind an APIG trigger to the function for E2E testing.

### 3.4.3.1 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 select **API Gateway (Dedicated)** for **Trigger Type**.
   - **API Instance**: Select a gateway. If no gateway is available, click **Create API Instance**.
   - **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 for your own services.
   - **Protocol**: Select **HTTPS**.
   - **Method**: Select **ANY**.
   - **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 3-26** Example


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

**----End**

### 3.4.3.2 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 select **API Gateway (Dedicated)** for **Trigger Type**.
   - **API Instance**: Select a gateway. If no gateway is available, click **Create API Instance**.
   - **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 for your own services.
   - **Protocol**: Select **HTTPS**.
   - **Method**: Select **ANY**.
   - **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
  ```

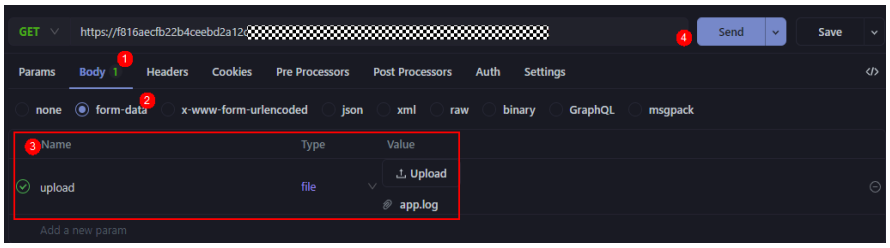
  **Figure 3-27** Example

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

**Figure 3-28** Example



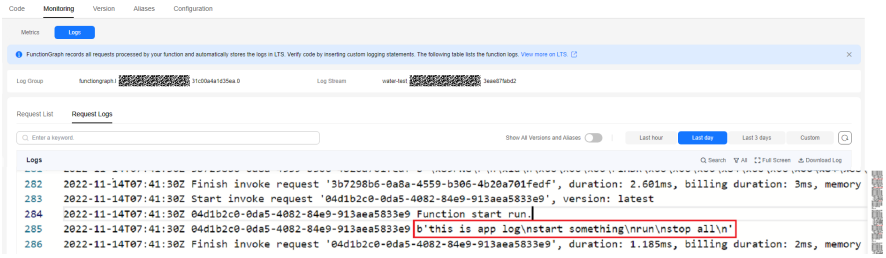

**Name**: Select **upload**.

**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 3-29** View logs




**----End**

# 3.5 Converting Device Coordinate Data in IoTDA

## 3.5.1 Introduction

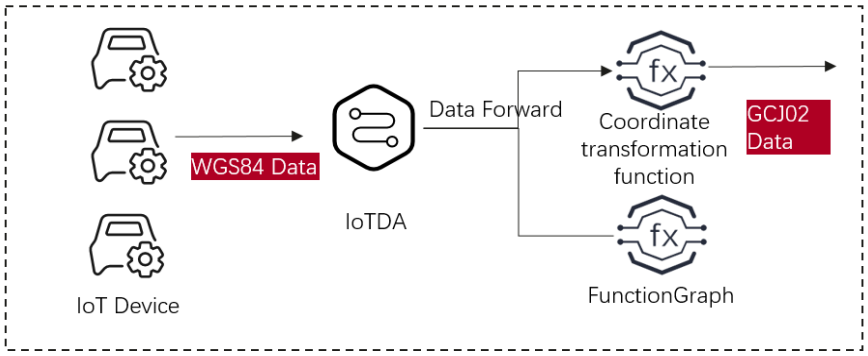
### Scenarios

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.

# 3.5.2 Preparation

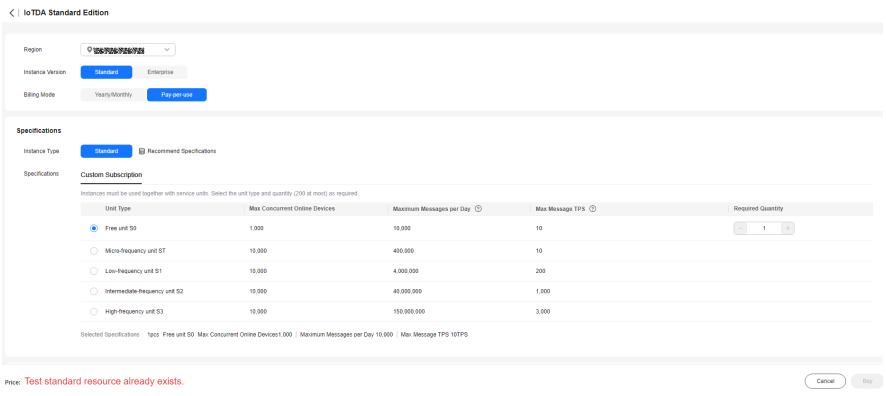
Before creating a forwarding rule, create an IoTDA instance as well as products and devices. In this best practice, we only create an instance for testing.

## 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 3-30** 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**.

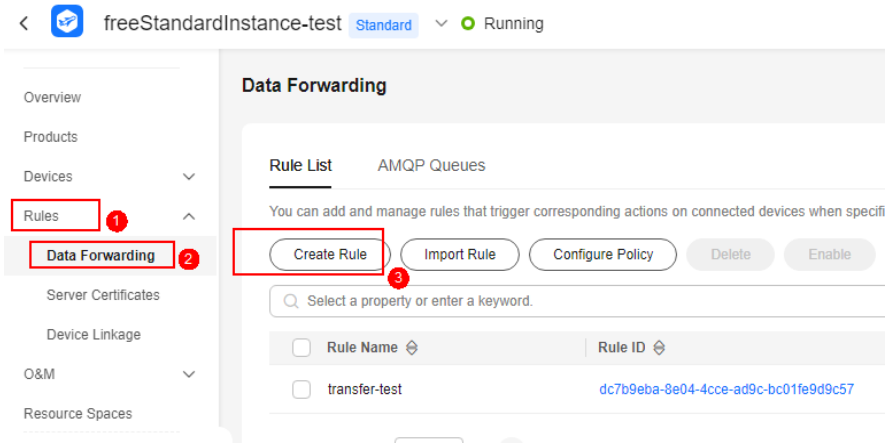
**----End**

## 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 3-31** 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.

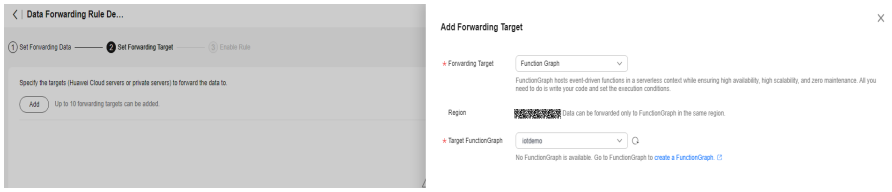
**Step 3** To set the forwarding target, click **Add**, and select **FunctionGraph**.

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

**Step 5** Select function **iotdemo**.

**Figure 3-32** Adding a forwarding target




**Step 6** Start the rule.

**----End**

# 3.5.3 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 3-33** 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 3-34** Connectivity test result


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

**Figure 3-35** Viewing logs

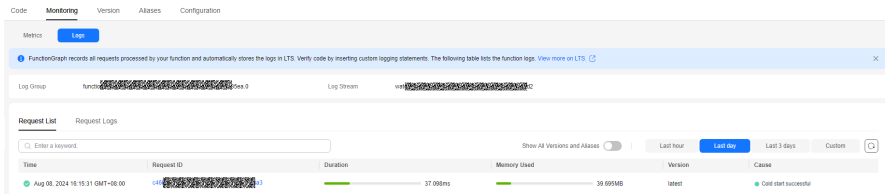
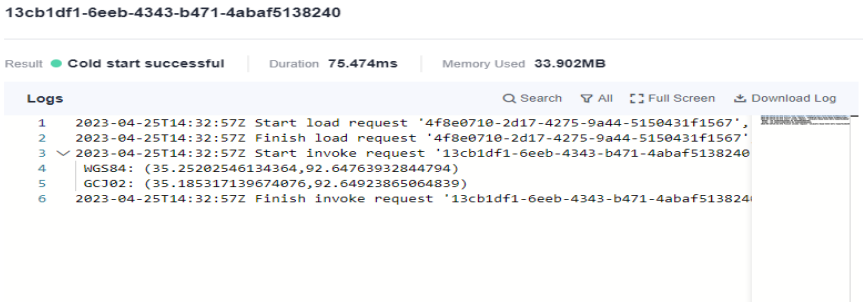



**Figure 3-36** Viewing request details




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

**----End**

# 3.6 Using FunctionGraph to Encrypt and Decrypt Files in OBS

## 3.6.1 Introduction

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

caused by network attacks and data reduction. This chapter describes how to use FunctionGraph and DEW to encrypt and decrypt files.

## Scenarios

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

  NOTE

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

## Procedure

- Create two buckets on the OBS console.
- Create a function and set the **OBS Application Service** trigger. (**OBS Application Service** trigger is available only in **CN North-Beijing4**, **CN North-Ulanqab1**, and **CN East-Shanghai1**.)
- Upload files to one of the buckets.
- Trigger the function to encrypt and decrypt the files.
- The function uploads the processed files to the other bucket.

  NOTE

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

  1. Two OBS buckets (for storing uploaded and processed files respectively)
  2. A file encryption/decryption function
  3. An **OBS Application Service** trigger, which is used to associate a function with an OBS bucket.

# 3.6.2 Preparation

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

Create an agency to delegate FunctionGraph to access OBS resources.

## Creating OBS Buckets

CAUTION

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

**Procedure**

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

**Step 2** On the **Create Bucket** page, set the bucket information.

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

Click **Create Now**.

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

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

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

**----End**

## Creating a DEW Key

⚠ CAUTION

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

**Procedure**

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

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

**Step 3** Record the master key ID.

**----End**

## Creating an Agency

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

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

**Step 3** Set the agency information.

- For **Agency Name**: Enter 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 **OBS Administrator** and **DEW KeypairFullAccess**.

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

**----End**

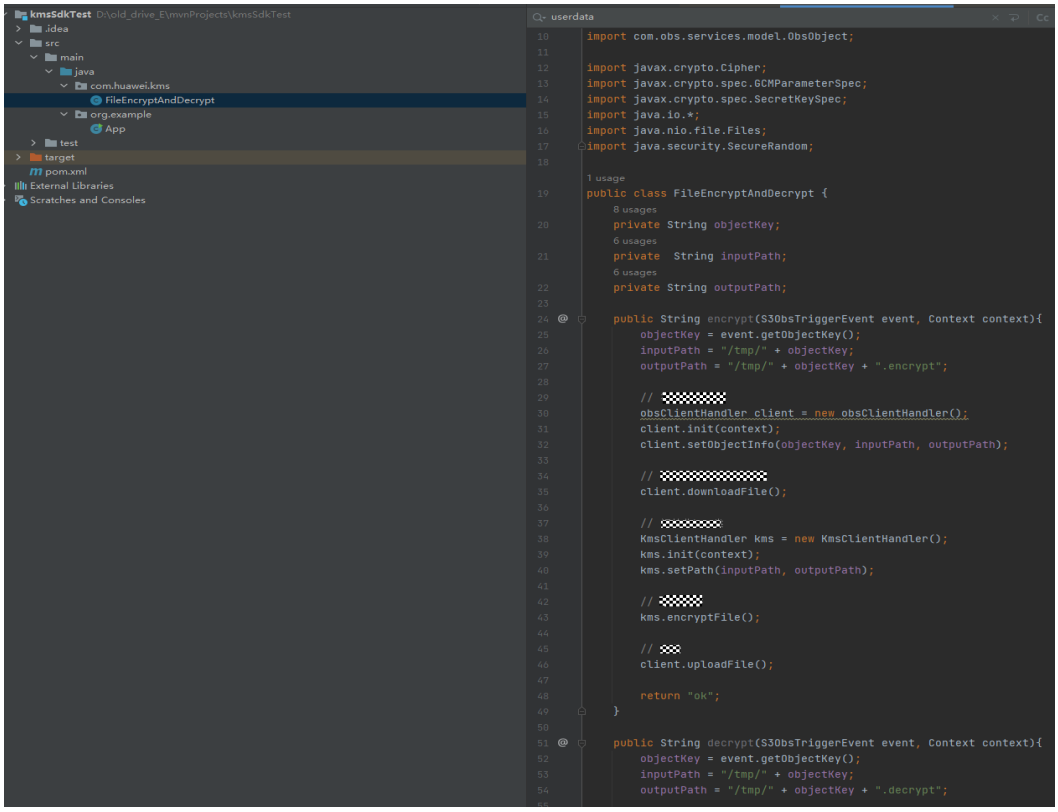
## 3.6.3 Building a Program

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

### Creating a Deployment Package

This example uses a Java 8 function to encrypt/decrypt files. For details about function development, see **Developing Functions in Java**. **Figure 3-37** shows the sample code directory. The service code is not described.

**Figure 3-37** Sample code directory




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

```
package com.huawei.kms;
import com.huawei.services.runtime.Context;
```

```
import com.huawei.services.runtime.entity.s3obs.S3ObsTriggerEvent;
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.kms.v1.KmsClient;
import com.huaweicloud.sdk.kms.v1.model.*;
import com.obs.services.ObsClient;
import com.obs.services.exception.ObsException;
import com.obs.services.model.ObsObject;
import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.nio.file.Files;
import java.security.SecureRandom;
public class FileEncryptAndDecrypt {
    private String objectKey;
    private  String inputPath;
    private String outputPath;
    public String encrypt(S3ObsTriggerEvent event, Context context){
        objectKey = event.getObjectKey();
        inputPath = "/tmp/" + objectKey;
        outputPath = "/tmp/" + objectKey + ".encrypt";
        // Initialize OBS class.
        obsClientHandler client = new obsClientHandler();
        client.init(context);
        client.setObjectInfo(objectKey, inputPath, outputPath);
        // Download files from the specified OBS bucket.
        client.downloadFile();
        // Initialize KMS class.
        KmsClientHandler kms = new KmsClientHandler();
        kms.init(context);
        kms.setPath(inputPath, outputPath);
        // Encrypt files.
        kms.encryptFile();
        // Upload files.
        client.uploadFile();
        return "ok";
    }
    public String decrypt(S3ObsTriggerEvent event, Context context){
        objectKey = event.getObjectKey();
        inputPath = "/tmp/" + objectKey;
        outputPath = "/tmp/" + objectKey + ".decrypt";
        // Initialize OBS class.
        obsClientHandler client = new obsClientHandler();
        client.init(context);
        client.setObjectInfo(objectKey, inputPath, outputPath);
        // Download files from the specified OBS bucket.
        client.downloadFile();
        // Initialize KMS class.
        KmsClientHandler kms = new KmsClientHandler();
        kms.init(context);
        kms.setPath(inputPath, outputPath);
        // Encrypt files.
        kms.decryptFile();
        // Upload files.
        client.uploadFile();
        return "ok";
    }
    static class KmsClientHandler {
        // DEW API version. Currently fixed to v1.0.
        private static final String KMS_INTERFACE_VERSION = "v1.0";
        private static final String AES_KEY_BIT_LENGTH = "256";
        private static final String AES_KEY_BYTE_LENGTH = "32";
        private static final String AES_ALG = "AES/GCM/PKCS5Padding";
        private static final String AES_FLAG = "AES";
        private static final int GCM_TAG_LENGTH = 16;
        private static final int GCM_IV_LENGTH = 12;
        private String ACCESS_KEY;
        private String SECRET_ACCESS_KEY;
        private String SECURITY_TOKEN;
```

```
    private String PROJECT_ID;
    private String KMS_ENDPOINT;
    private String keyId;
    private String cipherText;
    private String inputPath;
    private String outputPath;
    private Context context;
    private KmsClient kmsClient = null;
    void init(Context context) {
        this.context = context;
    }
    void initKmsClient() {
        if (kmsClient == null) {
            ACCESS_KEY = context.getSecurityAccessKey();
            SECRET_ACCESS_KEY = context.SecuritygetSecretKey();
            SECURITY_TOKEN = context.getSecurityToken();
            PROJECT_ID = context.getProjectID();
            KMS_ENDPOINT = context.getUserData("kms_endpoint");
            keyId = context.getUserData("kms_key_id");
            cipherText = context.getUserData("cipher_text");
            final BasicCredentials auth = new
BasicCredentials().withAk(ACCESS_KEY).withSk(SECRET_ACCESS_KEY).withSecurityToken(SECURITY_TOKEN).
withProjectId(PROJECT_ID);
            kmsClient = kmsClient.newBuilder().withCredential(auth).withEndpoint(KMS_ENDPOINT).build();
        }
    }
    byte[] getEncryptPlainKey() {
        final CreateDatakeyRequest createDatakeyRequest = new
CreateDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
                .withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));
        final CreateDatakeyResponse createDatakeyResponse =
kmsClient.createDatakey(createDatakeyRequest);
        final String cipherText = createDatakeyResponse.getCipherText();
        return hexToBytes(createDatakeyResponse.getPlainText());
    }
    byte[] hexToBytes(String hexString) {
        final int stringLength = hexString.length();
        assert stringLength > 0;
        final byte[] result = new byte[stringLength / 2];
        int j = 0;
        for (int i = 0; i < stringLength; i += 2) {
            result[j++] = (byte) Integer.parseInt(hexString.substring(i, i + 2), 16);
        }
        return result;
    }
    public void setPath(String inputPath, String outputPath) {
        this.inputPath = inputPath;
        this.outputPath = outputPath;
    }
    public void  encryptFile() {
        final File outEncryptFile = new File(outputPath);
        final File inFile = new File(inputPath);
        final byte[] iv = new byte[GCM_IV_LENGTH];
        final SecureRandom secureRandom = new SecureRandom();
        secureRandom.nextBytes(iv);
        doFileFinal(Cipher.ENCRYPT_MODE, inFile, outEncryptFile, getEncryptPlainKey(), iv);
    }
    byte[] getDecryptPlainKey() {
final CreateDatakeyRequest createDatakeyRequest = new
CreateDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
    .withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));
// Create a data key.
final CreateDatakeyResponse createDatakeyResponse = kmsClient.createDatakey(createDatakeyRequest);
        final DecryptDatakeyRequest decryptDatakeyRequest  = new
DecryptDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
                .withBody(new
DecryptDatakeyRequestBody().withKeyId(keyId).withCipherText(createDatakeyResponse.getCipherText()
```

```
).withDatakeyCipherLength(AES_KEY_BYTE_LENGTH));
            return hexToBytes(kmsClient.decryptDatakey(decryptDatakeyRequest).getDataKey());
        }
        public void  decryptFile() {
            final File outEncryptFile = new File(outputPath);
            final File inFile = new File(inputPath);
            final byte[] iv = new byte[GCM_IV_LENGTH];
            final SecureRandom secureRandom = new SecureRandom();
            secureRandom.nextBytes(iv);
            doFileFinal(Cipher.DECRYPT_MODE, inFile, outEncryptFile, getDecryptPlainKey(), iv);
        }
        /**         //     * Encrypt/Decrypt files.         //     *         //     * @param cipherMode Encryption mode.
Options: Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.         //     * @param infile     Files before
encryption/decryption.         //     * @param outFile    Files after encryption/decryption.         //     *
@param keyPlain   Plaintext key         //     * @param iv         Initialize vector.         //     */         void
doFileFinal(int cipherMode, File infile, File outFile, byte[] keyPlain, byte[] iv) {
            try (BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(infile.toPath()));
                 BufferedOutputStream bos = new
BufferedOutputStream(Files.newOutputStream(outFile.toPath()))) {
                final byte[] bytIn = new byte[(int) infile.length()];
                final int fileLength = bis.read(bytIn);
                assert fileLength > 0;
                final SecretKeySpec secretKeySpec = new SecretKeySpec(keyPlain, AES_FLAG);
                final Cipher cipher = Cipher.getInstance(AES_ALG);
                final GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(GCM_TAG_LENGTH *
Byte.SIZE, iv);
                cipher.init(cipherMode, secretKeySpec, gcmParameterSpec);
                final byte[] bytOut = cipher.doFinal(bytIn);
                bos.write(bytOut);
            } catch (Exception e) {
                throw new RuntimeException(e.getMessage());
            }
        }
    }
    static class obsClientHandler {
        private ObsClient obsClient = null;
        private String inputBucketName;
        private String outputBucketName;
        private String objectKey;
        private Context context;
        private String localInPath;
        private String localOutPath;
        public void init(Context context) {
            this.context = context;
        }
        void initObsclient() {
            if (obsClient == null) {
                inputBucketName = context.getUserData("input_bucket");
                outputBucketName = context.getUserData("output_bucket");
                String SECURITY_ACCESS_KEY = context.getSecurityAccessKey();
                String SECURITY_SECRET_KEY = context.getSecuritySecretKey();
                String SECURITY_TOKEN = context.getSecurityToken();
                String OBS_ENDPOINT = context.getUserData("obs_endpoint");
                obsClient = new ObsClient(SECURITY_ACCESS_KEY, SECURITY_SECRET_KEY, SECURITY_TOKEN,
OBS_ENDPOINT);
            }
        }
        public  void setObjectInfo(String objectKey, String inPath, String outPath) {
            this.objectKey = objectKey;
            localInPath = inPath;
            localOutPath = outPath;
        }
        public void downloadFile() {
            initObsclient();
            try {
                ObsObject obsObject = obsClient.getObject(inputBucketName, objectKey);
                InputStream inputStream = obsObject.getObjectContent();
                byte[] b = new byte[1024];
                int len;
```

```
            FileOutputStream fileOutputStream = new FileOutputStream("/tmp/" + objectKey);
            while ((len = inputStream.read(b)) != -1) {
                fileOutputStream.write(b);
            }
            inputStream.close();
            fileOutputStream.close();
        } catch (ObsException ex) {
            ex.printStackTrace();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    public void uploadFile() {
        try {
            // Local path of the files to upload. File names must be specified.
            FileInputStream fis = new FileInputStream(new File("/tmp/" + objectKey + ".encrypt"));
            obsClient.putObject(outputBucketName, objectKey, fis);
            fis.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
  }
}
```

## Creating a Function

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

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

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

**Step 3** Click **Create from scratch** and configure 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, **fss_examples_dew**.
- For **Agency**, select **serverless_trust**.
- For **Runtime**, select **Java 8**.

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

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

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

   **dew_endpoint**: DEW endpoint

   **dew_key_id**: Master key ID

   **input_bucket**: OBS bucket for storing uploaded files

   **output_bucket**: OBS bucket for storing encrypted/decrypted files

   **obs_endpoint**: OBS endpoint

   **Table 3-9** Environment variables

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

**----End**

## 3.6.4 Adding an Event Source

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

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

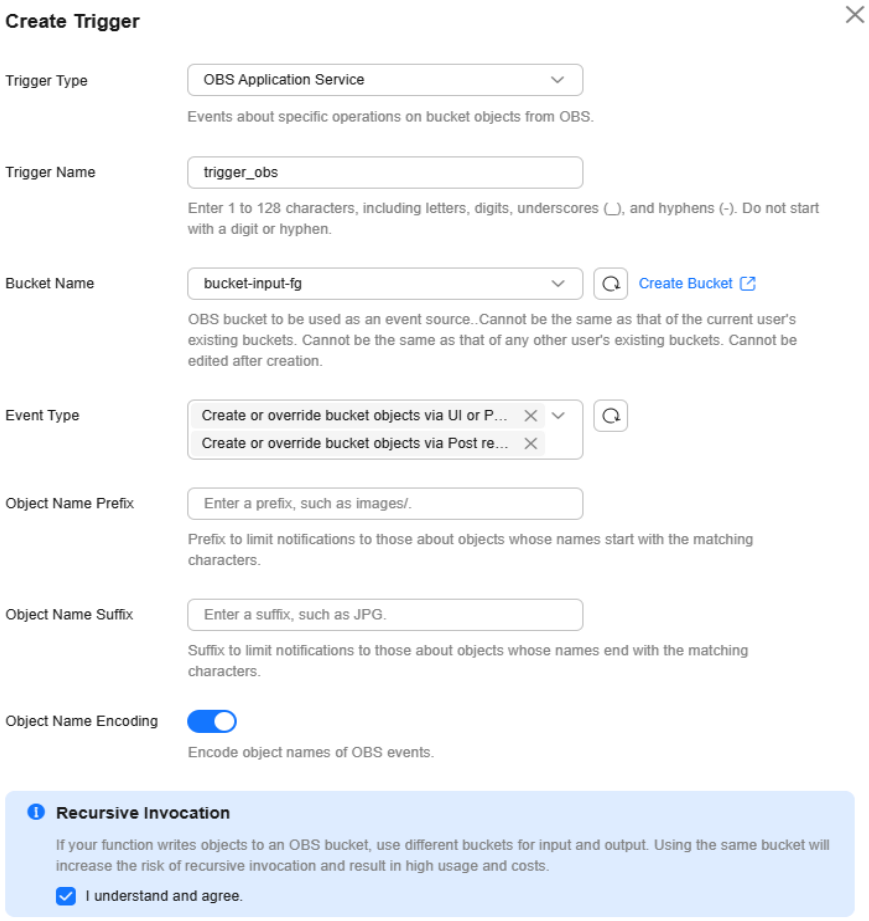
**Step 2** Select **OBS Application Service** for **Trigger Type**, and set the trigger information, as shown in **Figure 3-38**.

Select the created **dew-bucket-input** bucket.

Select **Create or override bucket objects via UI or Put request** or **Create or override bucket objects via Post request**. for **Event Type**.

**Figure 3-38** Creating an OBS Application Service trigger




**Step 3** Click **OK**.

NOTE

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

**----End**

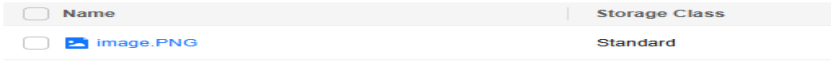
## 3.6.5 Processing Files

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

### Uploading a File to Generate an Event

Log in to the **OBS console**, go to the object page of the **dew-bucket-input** bucket, and upload the **image.png** file, as shown in **Figure 3-39**.

**Figure 3-39** Uploading a file



### Triggering the Function

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

The **Objects** page of the bucket **dew-bucket-output** displays the processed file **image.encrypt.png**, as shown in **Figure 3-40**. In the **Operation** column, click **Download** to download the file.

**Figure 3-40** Output file


# 3.7 Identifying Abnormal Service Logs in LTS and Storing Them in OBS

## 3.7.1 Introduction

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.

### Scenarios

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

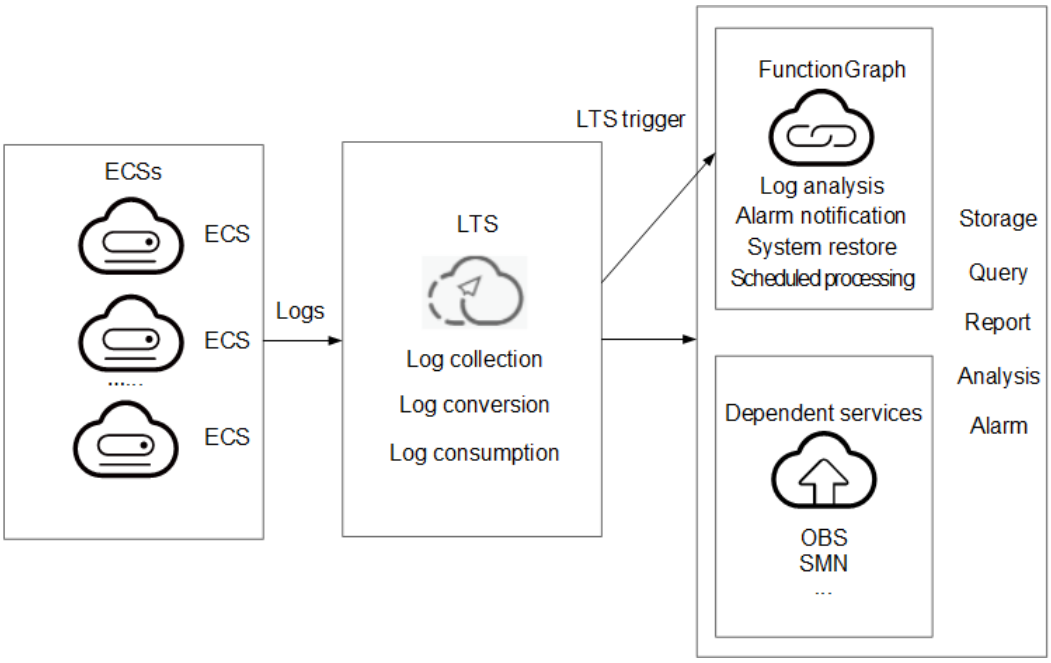
Obtain log data based on an LTS trigger created on FunctionGraph, analyze and process key information in the logs by using a customized function, and then filter alarm logs.

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 3-41**.

**Figure 3-41** 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.

## Extended Applications

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.

# 3.7.2 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.

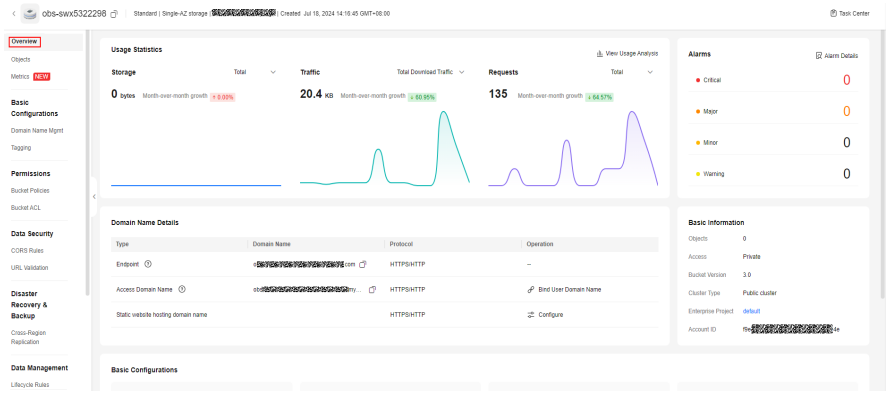
  In this example, when log events trigger the specified function through an LTS trigger, the function filters alarm logs and pushes alarm message to the subscription endpoints.

## 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 3-42**. For details, see **Creating a Bucket**.

   **Figure 3-42** Creating a bucket

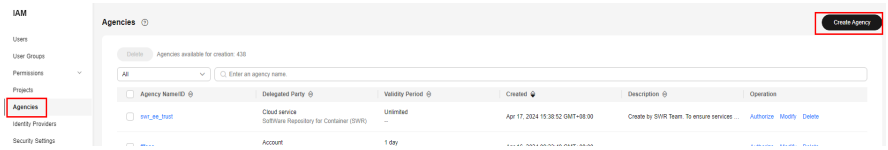
   **NOTE**

   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 3-43** 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.

## 3.7.3 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 3-10**.

**Table 3-10** Environment variables

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

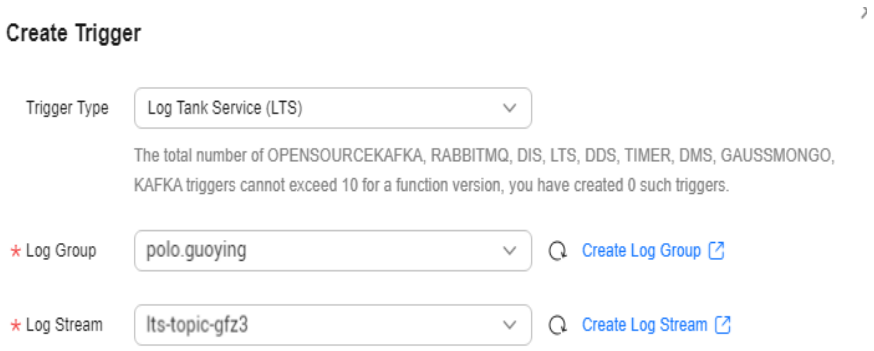
# 3.7.4 Adding an Event Source

## Creating an LTS trigger

Create an LTS trigger by using the log group and log topic created in **Preparation**. Configure the trigger information according to **Figure 3-44**.

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.

**Figure 3-44** Creating an LTS trigger




# 3.7.5 Processing Log Data

## Handling Alarms

Email notifications will be received from SMN if alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR** are generated, as shown in **Figure 3-45**. You can also view details of the alarm logs by opening the **log.txt** file in the specified bucket, as shown in **Figure 3-46**.

**Figure 3-45** Email notification

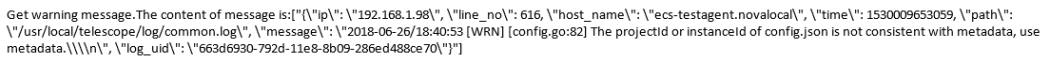
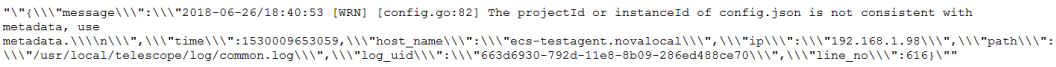


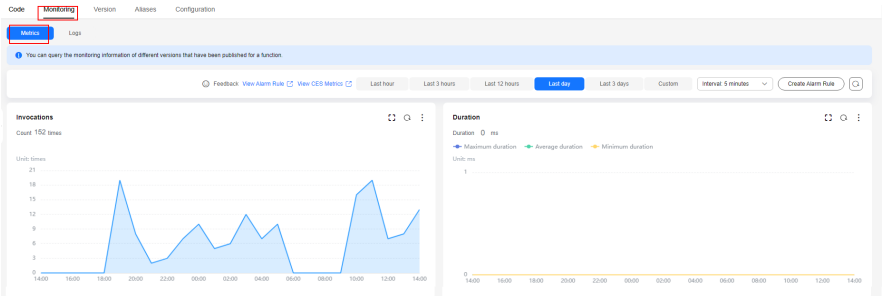

**Figure 3-46** Alarm log details




On the **Monitoring** tab page of the function, check the number of invocations, as shown in **Figure 3-47**.

**Figure 3-47** Function metrics




# 3.8 Using FunctionGraph to Filter Logs in LTS in Real Time

## 3.8.1 Introduction

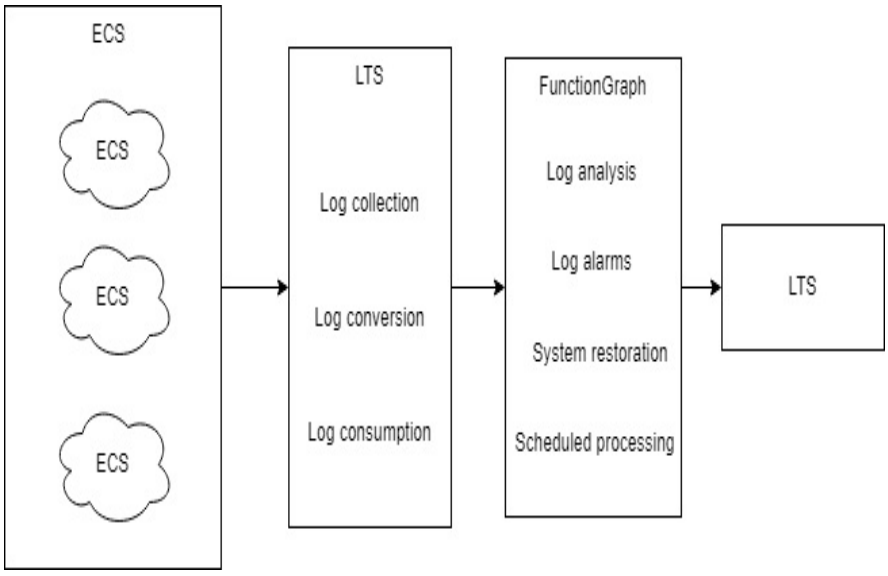
This practice shows how to use FunctionGraph and LTS to process logs and transfer messages to LTS.

### 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 3-48** shows this process.

**Figure 3-48** 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.

### Extended Applications

Use FunctionGraph and LTS to periodically analyze and process log data with a timer trigger to delete redundant logs and save space and costs.

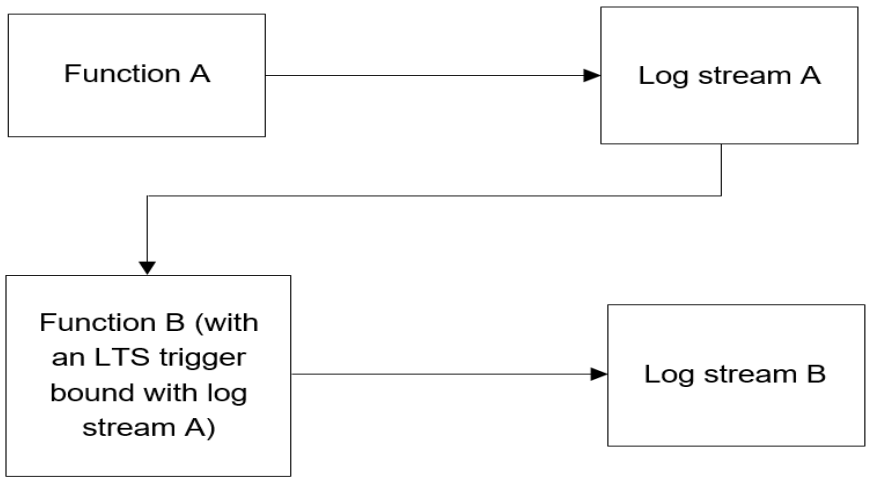
## 3.8.2 Preparation

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 3-49** Flowchart



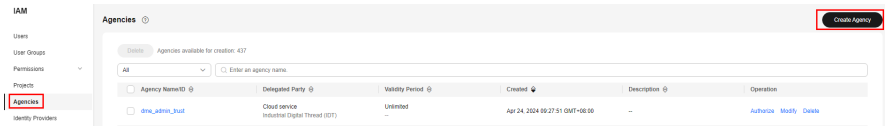

  

## Creating an Agency

**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 3-50**.

**Figure 3-50** 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**

# 3.8.3 Building a Program

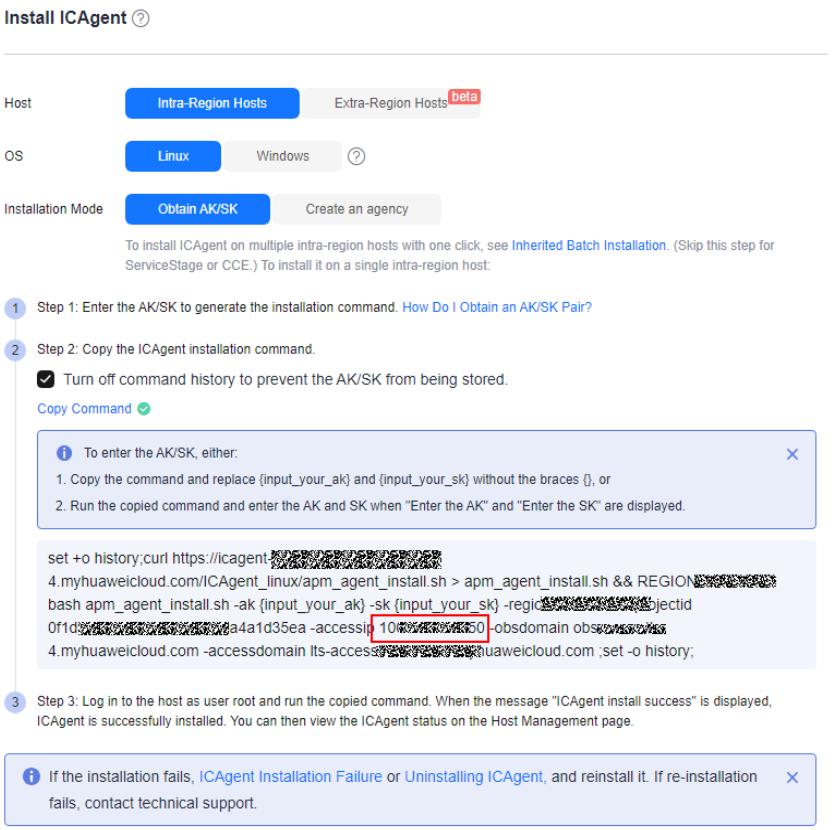
## Prerequisites

(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 3-51** 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.

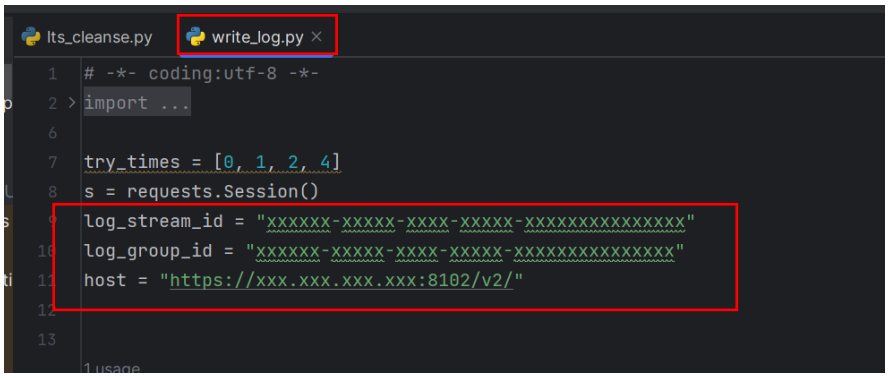
## Creating a Function

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 3-52**.

**Figure 3-52** write_log.py




Create function **B**. For the sample code of this function, see the **lts_cleanse.py** file. In the code of function **B**, replace **host**, **log_group_id**, and **log_stream_id** with the access point and the IDs of log group **test-1121** and log stream **test-1121**, and add the **huaweicloudsdklts** dependency to this function, as shown in **Figure 3-53** and **Figure 3-54**.

**Figure 3-53** lts_cleanse.py

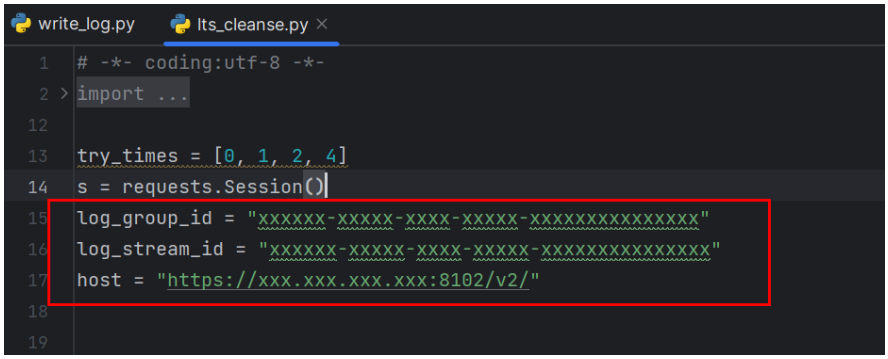
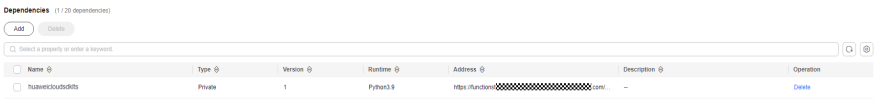



**Figure 3-54** Adding a dependency for function B




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.

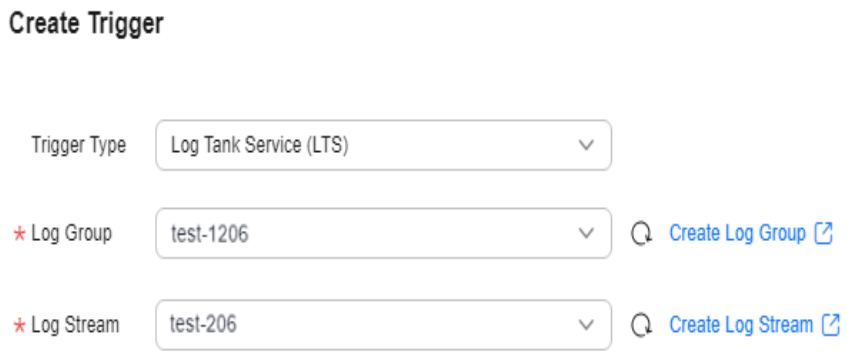
## 3.8.4 Adding an Event Source

### Creating an LTS trigger

Create an LTS trigger by using the log group and log stream created in **Preparation**, and configure the trigger information according to **Figure 3-55**.

**Figure 3-55** Creating an LTS trigger




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

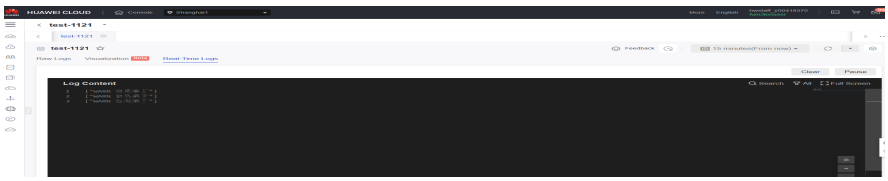
## 3.8.5 Processing Results

### Handling Alarms

Filter alarm logs containing keyword **WRN**, **WARN**, **ERR**, or **ERROR**, and transfer them to a specified log stream. **Figure 3-56** and **Figure 3-57** show the real-time logs before and after filtering, respectively.

**Figure 3-56** Logs before filtering

**Figure 3-57** Logs after filtering



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

**Figure 3-58** Function metrics (1)

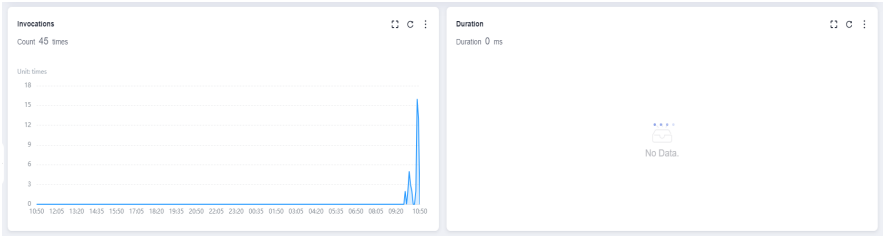


**Figure 3-59** Function metrics (2)

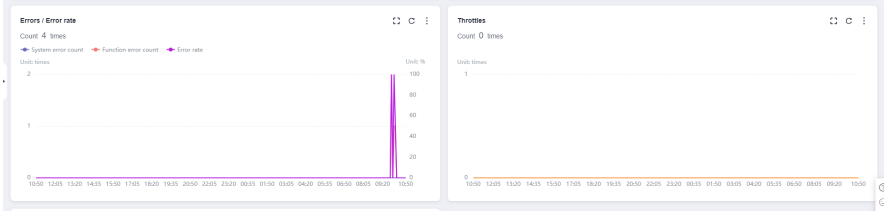



**Figure 3-60** Function metrics (3)


# 3.9 Using FunctionGraph to Rotate Images Stored in OBS

## 3.9.1 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**)

# 3.9.2 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**

## 3.9.3 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 3-61** shows the sample code directory. The service code is not described.

**Figure 3-61** Sample code directory

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

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

## Creating a Function

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

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

Click **Create Function**.

Set the function information.

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

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

- For **Runtime**, select **Go 1.x**.

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

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

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

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

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

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

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

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

**----End**

**Table 3-11** 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**

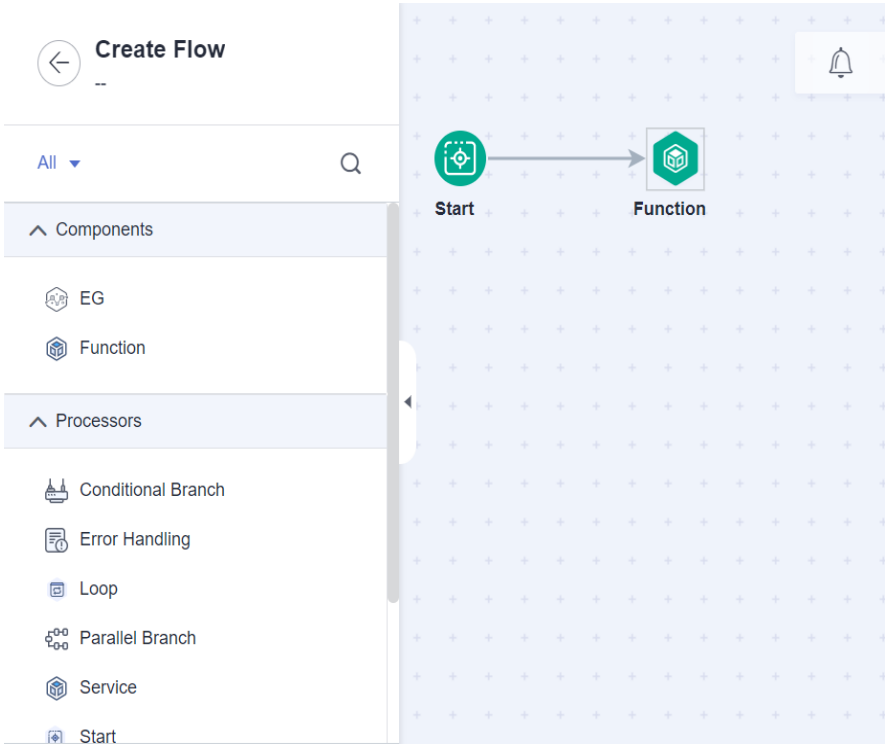
  

## Creating a Flow

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

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

**Figure 3-62** Creating an express flow

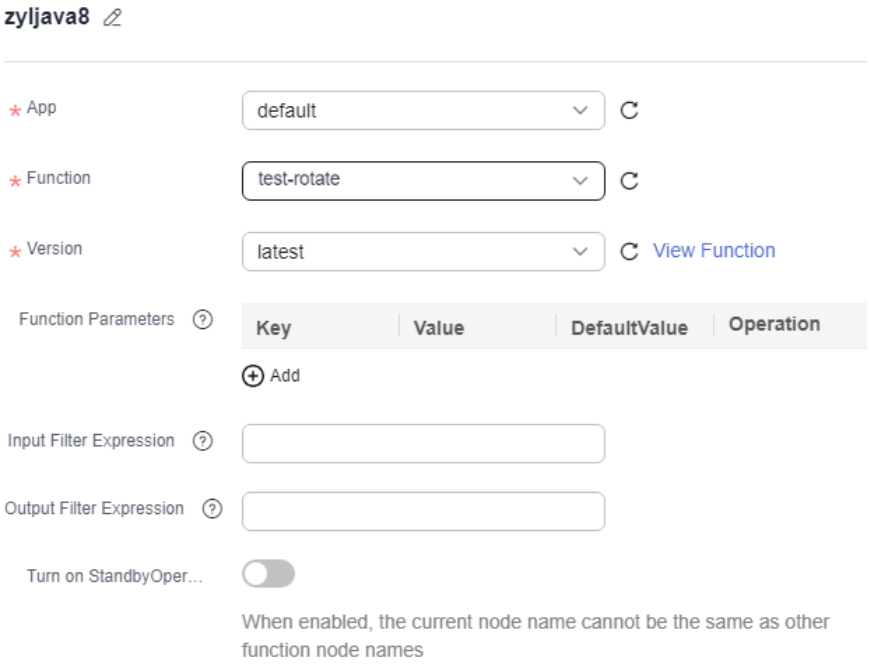



**Step 2** Drag a function node and click it to configure parameters.

- **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 3-63** Configuring function node



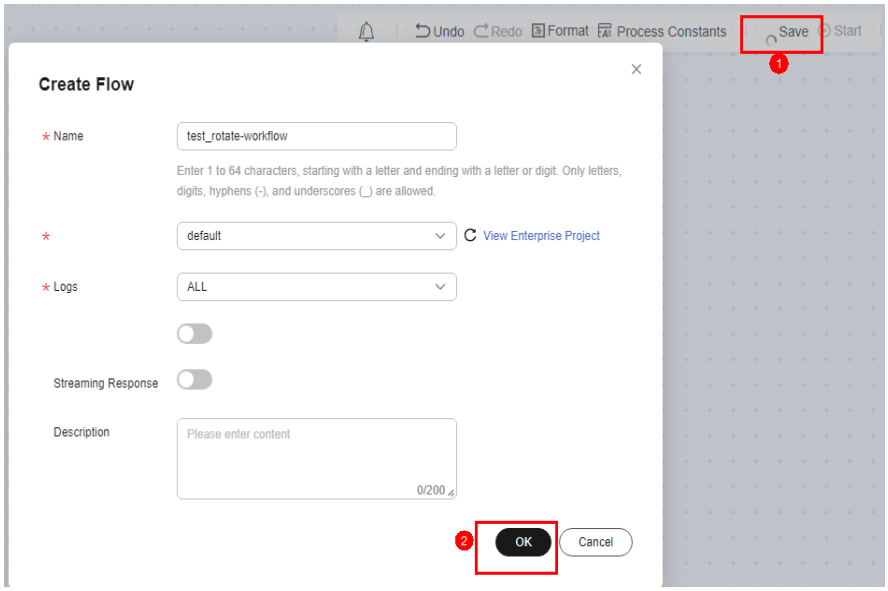

Click **OK** after the parameters are configured.

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

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

Retain the default values for other parameters.

**Figure 3-64** Saving a flow




**----End**

# 3.9.4 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 3-65**. **Figure 3-66** shows the uploaded image.

**Figure 3-65** Example

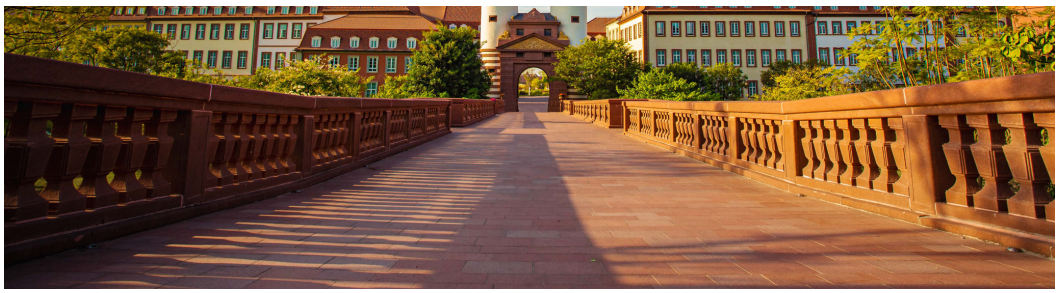
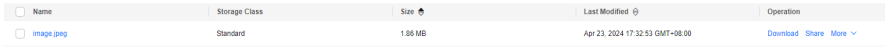
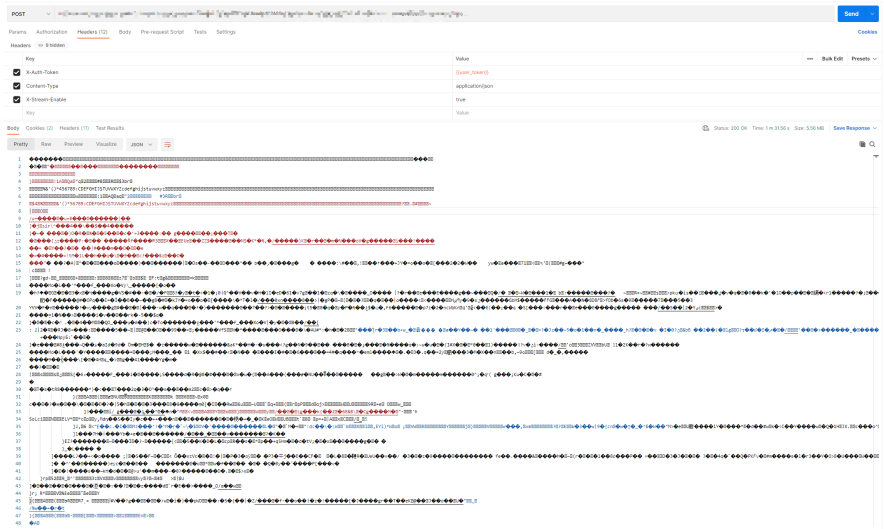


**Figure 3-66** Uploaded image




## Using Postman to Trigger the Function Flow



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

# 3.10 Using FunctionGraph to Compress and Watermark Images

This section describes how to use FunctionGraph to process large stream files. Create an express flow that meets your service requirements.

## Background and Value

Serverless workflows feature orchestration, state management, persistence, visualized monitoring, error handling, and cloud service integration. They are suitable for many scenarios, including:

- Complex, abstract services, such as order management and CRM
- Services that require automatic interruption and recovery when manual intervention is involved among tasks, such as manual review and pipeline deployment
- Services that require manual interruption and recovery, such as data backup and restoration
- Task status monitoring
- Stream processing, such as log analysis and image/video processing

  Nowadays, most serverless workflow platforms focus more on control process orchestration rather than data flow orchestration and transmission. In scenarios similar to , the data flow is simple and well supported by various platforms. However, they do not have a solution for ultra-large data stream processing scenarios, such as file transcoding. For these scenarios, Huawei Cloud FunctionGraph provides the serverless streaming solution, which responds to process files within milliseconds.

## Principles

Huawei Cloud FunctionGraph provides the serverless streaming solution for file processing via orchestration. Steps are driven by data flows, which is easier to understand. This section uses image processing as an example to describe how this solution works.

A workflow system needs to process two parts:

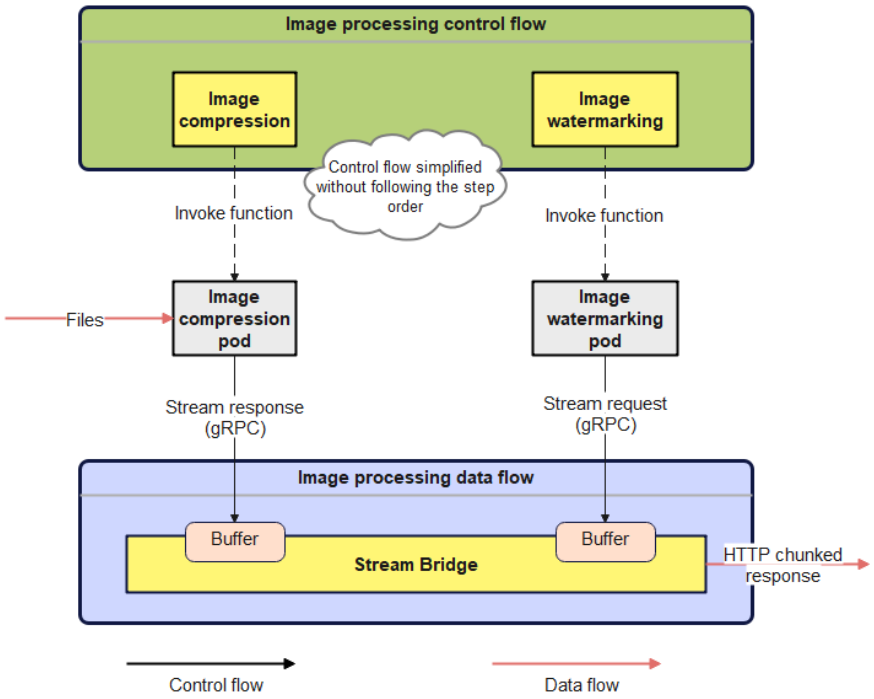
- Control flow: Controls the flow between steps and the execution of serverless functions in the steps. Ensure that steps are performed in sequence.
- Data flow: Controls the data flowing through the entire workflow. Generally, the output of a previous step serves as the input of the next step. For

example, in the preceding image processing workflow, the image compression result is the input of the watermarking step.

In common service orchestration, the execution sequence of each service needs to be precisely controlled, so the control flow is the core of a workflow. However, streaming processing scenarios such as file processing do not require more on the control flow. For example, in the image processing scenario, large images can be processed by block. Image compression and watermarking are not necessarily completed in a specific sequence.

The following figure shows the architecture of Huawei Cloud FunctionGraph's serverless streaming solution.




Serverless streaming allows steps to be executed in parallel rather than in a specific sequence. Steps interact with each through data flows, which are controlled by the Stream Bridge component. The function SDKs include a streaming data API, which writes data into Stream Bridge in a gRPC stream. Stream Bridge then distributes the data flow into the function pod in the next step.

## Procedure

**Step 1** Create an image compression function, which uses **ctx.Write()** to return results as streaming data.

NOTE

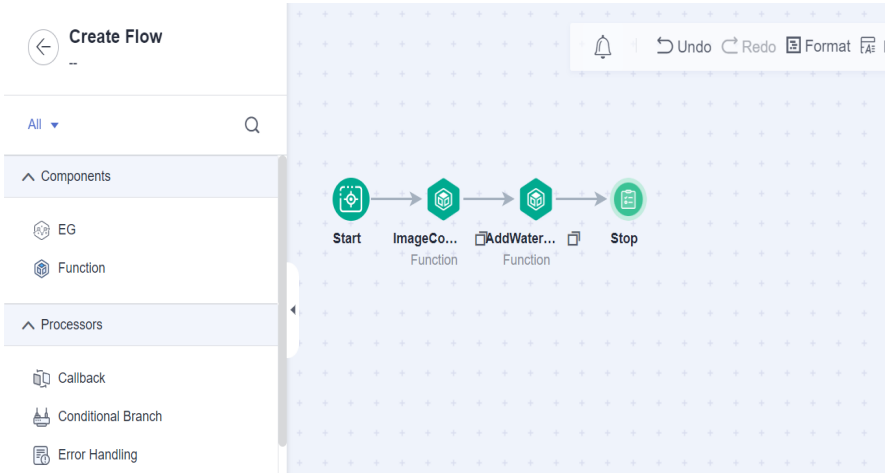
Currently, only Go functions are supported.

```
func ImageTransform(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    reader, writer := io.Pipe()
    file, err := downloadOriginFile(ctx)
    if err != nil {
        return "nok", err
    }

    go func() {
        defer writer.Close()
        encodeImageFile(writer, file, ctx)
    }()

    defer reader.Close()
    // Write result stream back to client
    err = ctx.Write(reader)
    return "ok", err
}

// downloadOriginFile used to download the origin video file
func downloadOriginFile(ctx context.RuntimeContext) (*os.File, error) {
}

// encodeImageFile used to encode the origin image file to target resolution, result output will writes to writer stream
func encodeImageFile(writer io.Writer, originFile *os.File, ctx context.RuntimeContext)  {
}
```

FunctionGraph supports streaming response with **ctx.Write()**. Instead of focusing on network transmission, you only need to return the final results in a stream.

**Step 2** Create a workflow on the FunctionGraph console.




**Step 3** Invoke the synchronous flow execution API to obtain the file stream. The data is returned to the client in chunked streaming mode.

**----End**

# 4 Functional Application Practices

## 4.1 Using FunctionGraph and CTS to Identify Login and Logout Operations from Invalid IP Addresses

### 4.1.1 Introduction

#### Scenarios

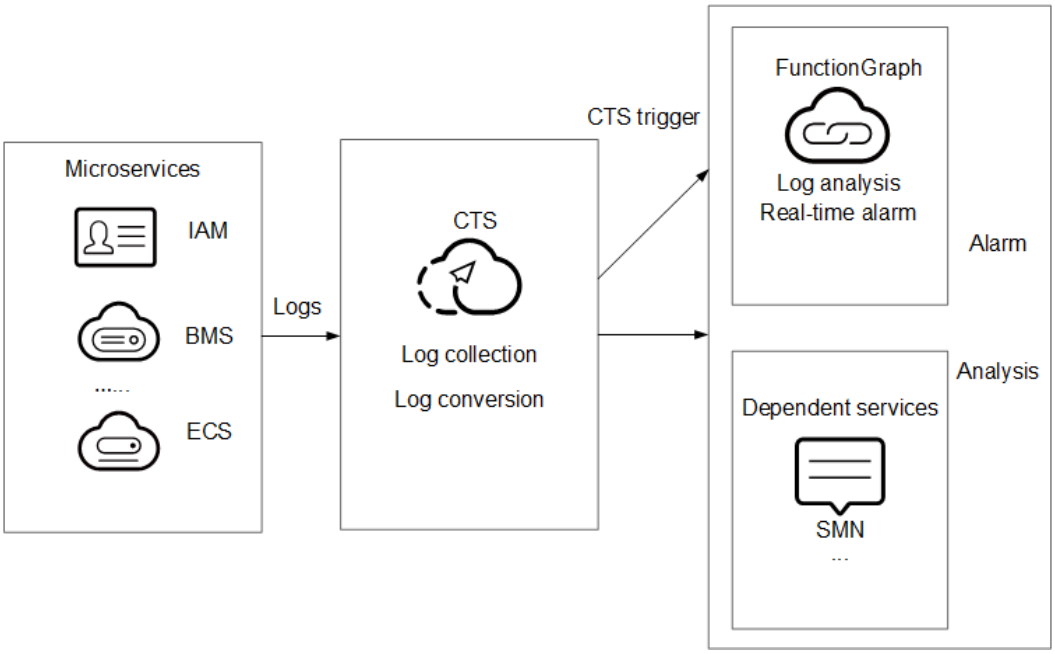
Use Cloud Trace Service (CTS) to collect real-time records of cloud resource operations.

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.

## Notes and Constraints

This practice involves operations on CTS and SMN. For details about the constraints, see the following sections:

- **Data Trackers**
- **Creating a Key Event Notification**

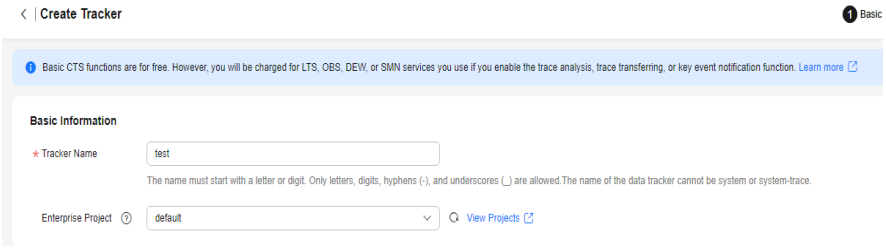
# 4.1.2 Preparation

## 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




## 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

For details about the configuration process and restrictions, see **Creating a Key Event Notification**.

- 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 can be sent by 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.

## 4.1.3 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. |

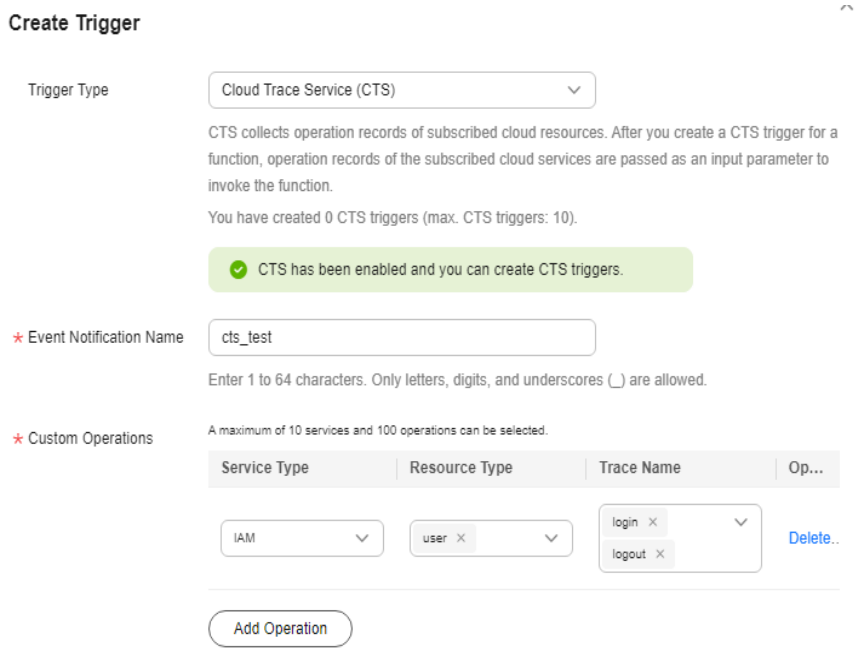
Set the environment variables by following the procedure in **Environment Variables**.

## 4.1.4 Adding an Event Source

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

**Figure 4-3** Creating a CTS trigger

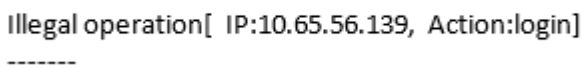



CTS records the logins and logouts of users on IAM.

## 4.1.5 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**.

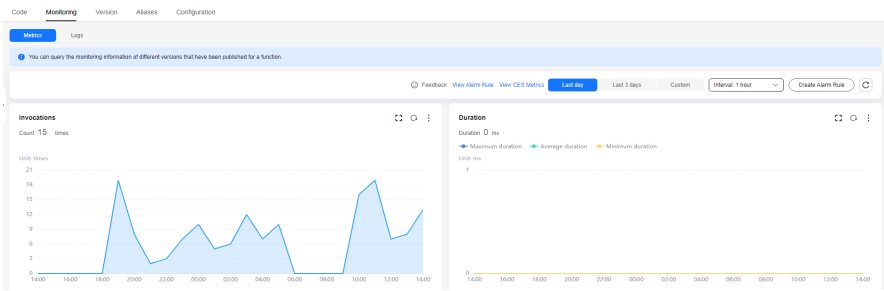
**Figure 4-4** Email notification




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



# 4.2 Using FunctionGraph Functions As the Backend to Implement APIG Custom Authorizers

## 4.2.1 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 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

## 4.2.2 Resource Planning

Ensure that the following resources are in the same region.

**Table 4-2** Resource planning

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

# 4.2.3 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**.

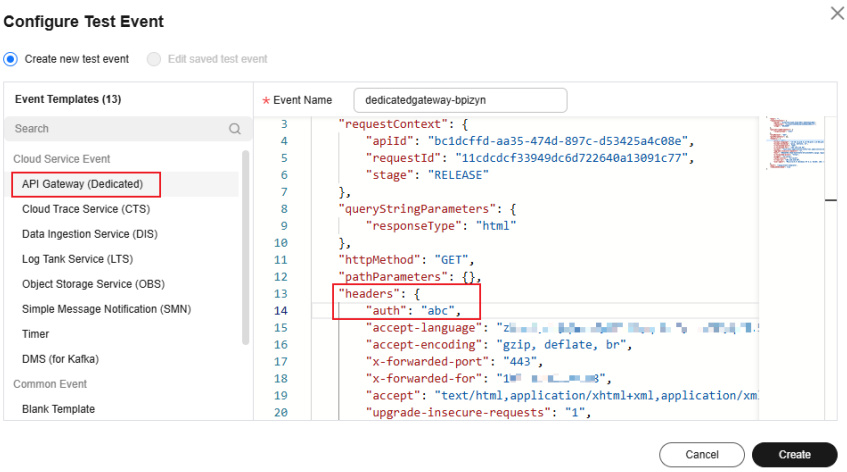
- **Template**: Select **Create from scratch**.
- **Function Type**: Select **Event Function**.
- **Function Name**: Enter a function name, for example, **apig-test**.
- **Agency**: Select **Use no agency**.
- **Runtime**: Select **Python 2.7**.

**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"**.

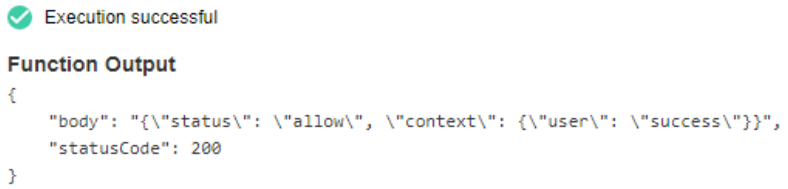
  

**Figure 4-6** Configuring a test event




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

**Figure 4-7** Viewing the execution result




**----End**

## Creating a Custom Authorizer

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

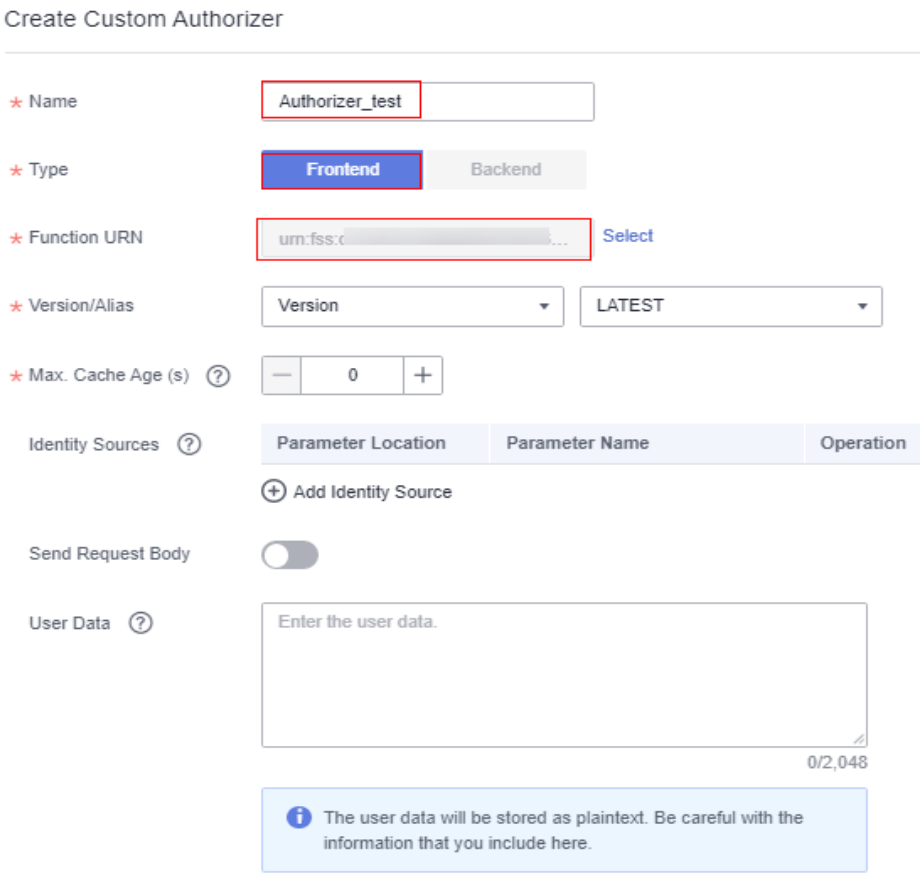
**Step 1** In the left navigation pane of the management console, choose **Middleware** > **API Gateway** to go to the APIG console. In the navigation pane, choose **API Management** > **API Policies**. On the **Custom Authorizers** tab, click **Create Custom Authorizer**.

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

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

**Figure 4-8** Creating a custom authorizer




**Step 3** Click **OK**.

**----End**

## Creating a Backend Service Function

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

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

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

```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    body = "<html><title>Functiongraph Demo</title><body><p>Hello, FunctionGraph!</p></body></html>"
    print(body)
    return {
        "statusCode":200,
        "body":body,
        "headers": {
            "Content-Type": "text/html",
        },
```

```
    "isBase64Encoded": False
}
```

**----End**

## Request Parameter Code Example

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

The function must have a clear API definition. Example:

**def handler (event, context)**

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

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

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

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

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

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

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

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

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

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

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

### Example Header Parameter

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

### Example Query String

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

### Example User Data

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

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

# 4.2.4 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 4-9** and **Figure 4-10**.

- **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 4-9** Configuring frontend definition

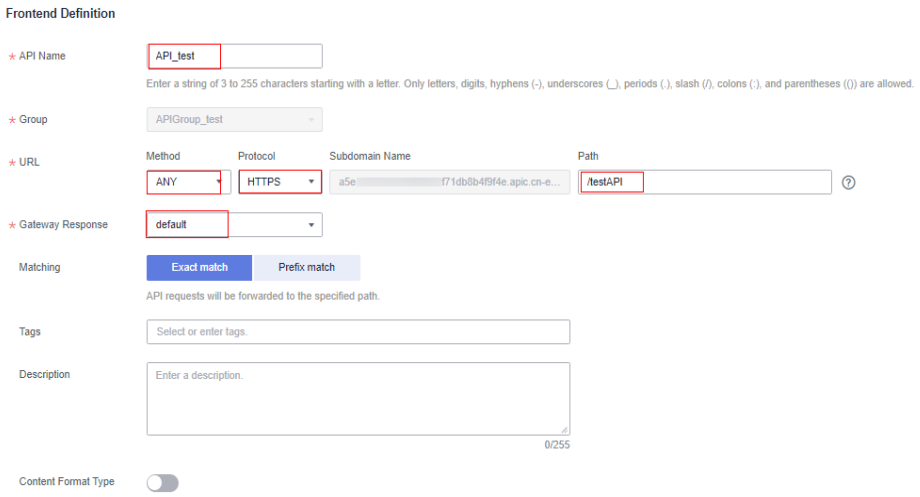
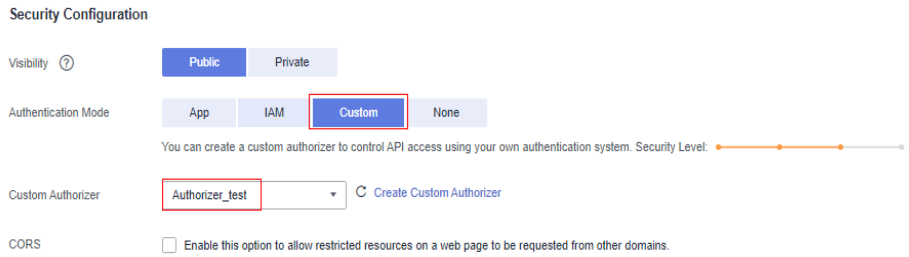


**Figure 4-10** Configuring security settings



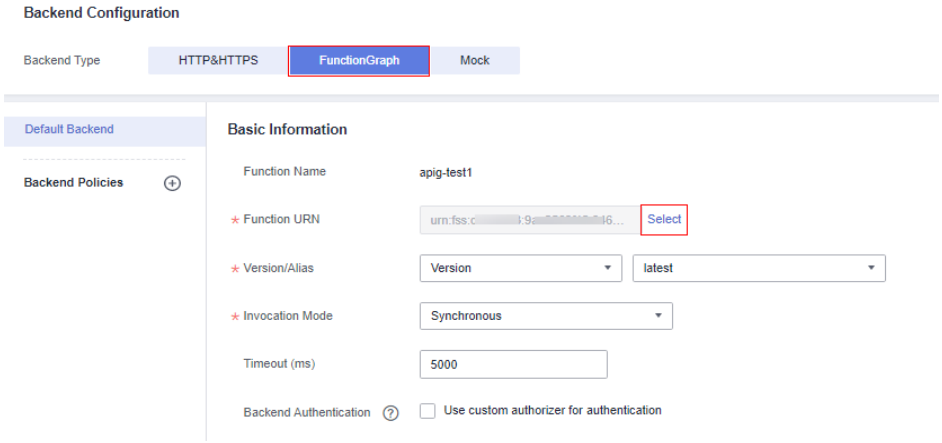

NOTE

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

**Step 3** Click **Next** to configure the backend service according to **Figure 4-11**.

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

**Figure 4-11** Configuring the backend service




**Step 4** Click **Finish**.

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

**Figure 4-12** Publishing an API


**----End**

## 4.2.5 Debugging and Calling the API

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

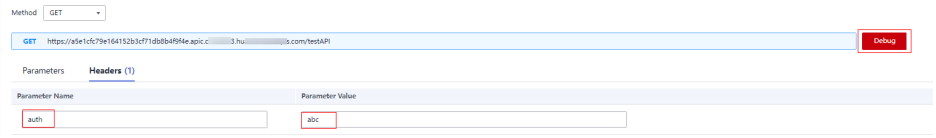
**Step 1** Log in to the APIG console. In the navigation pane, choose **API Management** > **APIs**. Then click **API_test**, and click **Debug**.

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

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

**Figure 4-13** Adding a header




**Step 3** Check whether the API response contains the content you have defined in the service function. See **Figure 4-14**.

**Figure 4-14** API response

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

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

**----End**

# 4.3 Using FunctionGraph HTTP Functions to Process gRPC Requests

## Introduction

This section describes how to 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
```

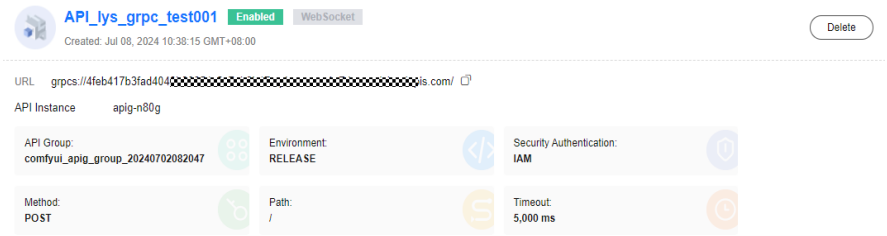
3. **Creating an HTTP function and uploading code**

   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 4-15** APIG trigger

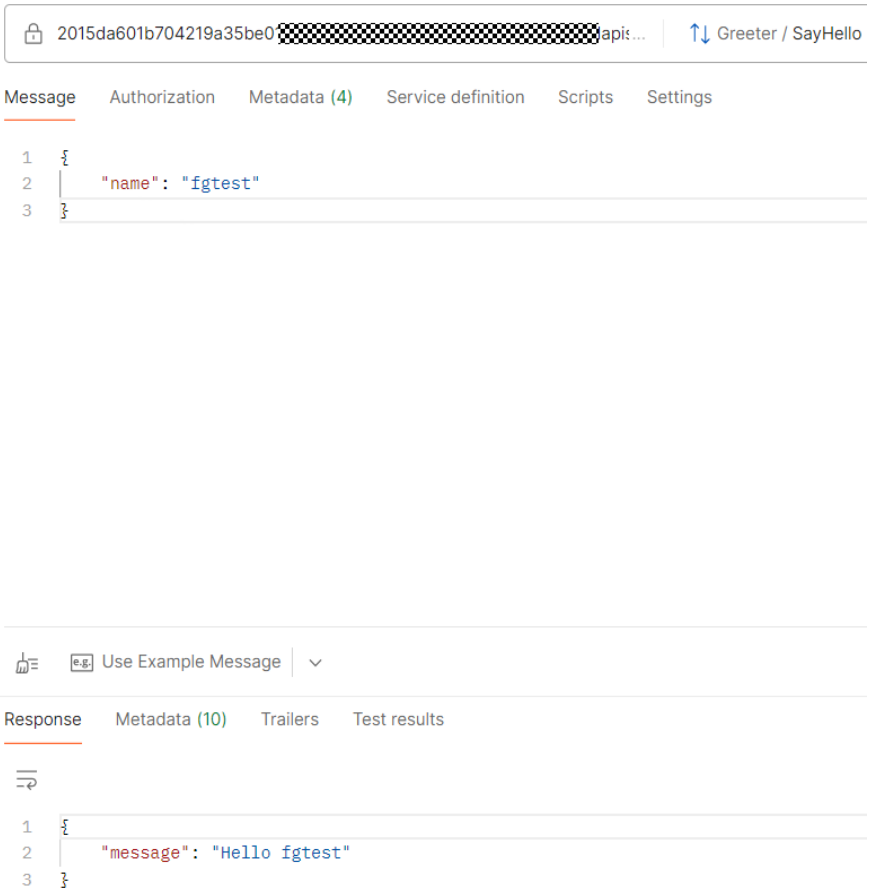



5. **Invocation test**

   Use Postman to debug gRPC requests.

**Figure 4-16** gRPC request result




# 4.4 Using a Java Function and Log4j2 to Print Logs

## Introduction

FunctionGraph supports Log4j2 for Java functions. This section describes how to use functions and Log4j2 to print logs.

## Step 1: Download a Package

In this example, Java is used to implement log printing. You can directly download the sample code **Log_demo.jar** without any modification.

The key sample code is as follows.

```
package org.example;

import com.huawei.services.runtime.Context;
import lombok.extern.slf4j.Slf4j;
import org.apache.logging.log4j.core.config.Configurator;
import org.apache.logging.log4j.util.LoaderUtil;

import java.net.URISyntaxException;
import java.util.Objects;

@Slf4j
```

```
public class LogTest {

    public void init(Context context) {
        try {

Configurator.reconfigure(Objects.requireNonNull(LoaderUtil.getThreadContextClassLoader().getResource("log4j2-custom.xml")).toURI());
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
    }

    public void handler(String event, Context context) {
        log.debug("debug log");
        log.info("info log");
        log.warn("warn log");
        log.error("info log");
    }
}
```

The following code is added to the initializer:

```
Configurator.reconfigure(Objects.requireNonNull(LoaderUtil.getThreadContextClassLoader().getResource("log4j2-custom.xml")).toURI());
```

## Step 2: Creating a Function

**Step 1** Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane. In the upper right corner, click **Create Function**.

**Step 2** Click **Create from scratch** and configure the function information.

- **Function Type**: Select **Event Function**.
- **Region**: Select a region based on site requirements.
- **Function Name**: Enter a custom name.
- **Runtime**: Select **Java 8**.

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

**Step 3** Upload the function code.

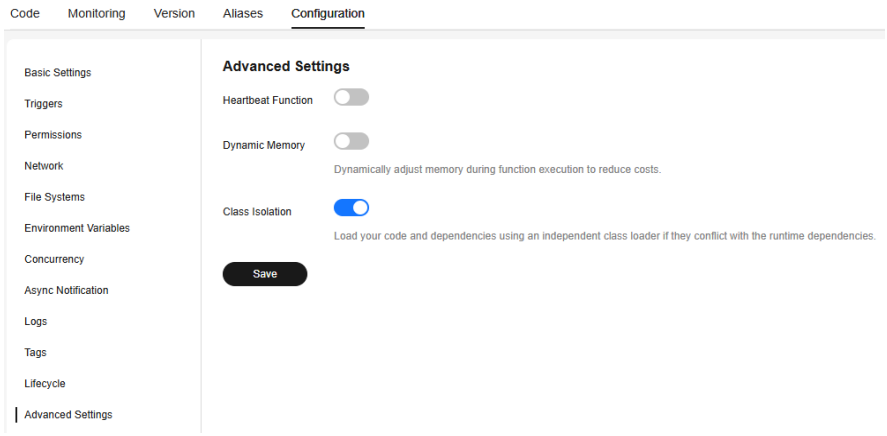
After the function is created, go to the function details page, click the **Code** tab, choose **Upload** > **Local ZIP**, and add the ZIP file downloaded in **Step 1: Download a Package**.

**Step 4** Enable class isolation.

After the code package is successfully deployed, choose **Configuration** > **Advanced Settings**, enable **Class Isolation**, and click **Save** as shown in **Figure 4-17**.

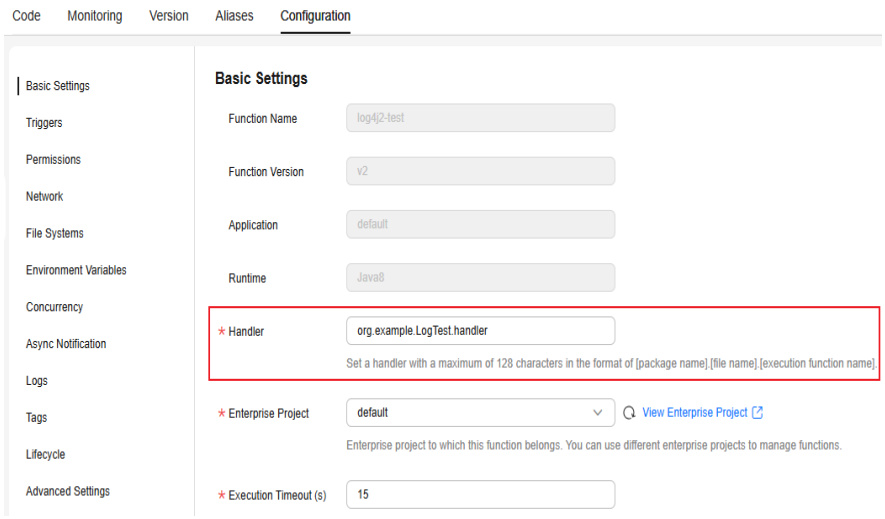
**Figure 4-17** Enabling class isolation




**Step 5** Set the function handler.

As shown in **Figure 4-18**, choose **Configuration** > **Basic Settings**, set **Handler** to **org.example.LogTest.handler**, and click **Save**.

**Figure 4-18** Setting the handler

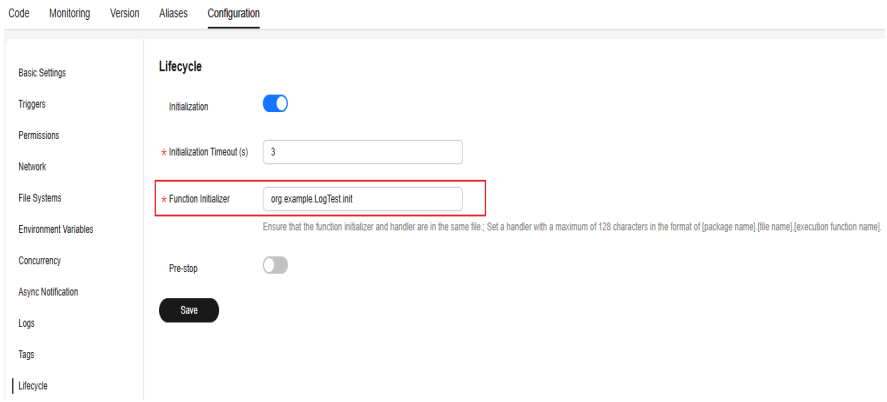



**Step 6** Set the function initializer.

As shown in **Figure 4-19**, choose **Configuration** > **Lifecycle**, enable **Initialization**. On the displayed page, set **Function Initializer** to **org.example.LogTest.init**, and click **Save**.

**Figure 4-19** Setting the function initializer

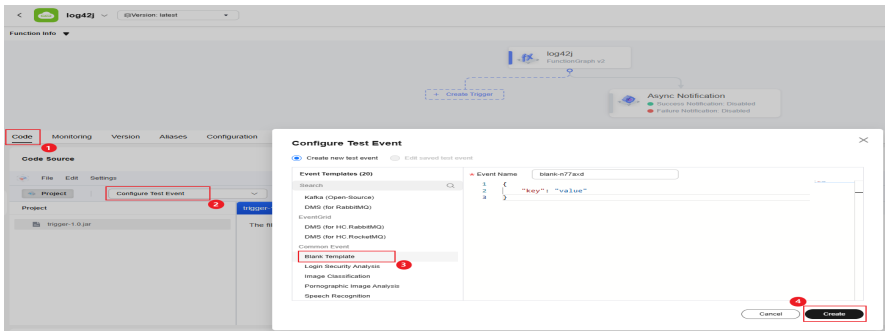



**----End**

## Step 3: Testing the Function

**Step 1** After all parameters are configured, as shown in **Figure 4-20**, click the **Code** tab and click **Configure Test Event**, select **Blank Template**, and click **Create**.

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




**Step 2** Select the created test event and click **Test**. The test result is displayed in **Execution Result** tab on the right.

**----End**

# 4.5 Using FunctionGraph to Deploy Stable Diffusion for AI Drawing

## 4.5.1 Introduction

### Stable Diffusion Scenarios

Stable Diffusion is an open-source text-to-image generation model. It can generate high-quality and unique images based on user prompts, providing a wide range of creative possibilities. With Stable Diffusion WebUI, you can see the image generation process.

You can deploy Stable Diffusion applications in FunctionGraph **Applications** page as required.

- **Quick deployment**: You can use the default model and temporary domain name to quickly deploy the Stable Diffusion model without having a deep technical background.
- **Custom domain name**: You can bind a custom domain name for your Stable Diffusion application to enable private and public access.
- **Custom model**: You can mount a file system to an application and upload a custom model. Different model capabilities can be used to generate more personalized images.
- **Advanced usage**: For more scenarios, see **(Advanced) Mounting an SFS File System to Multiple Users**, **(Advanced) Using ECS as an NFS Server to Isolate Resources of Multiple Users**, **(Advanced) Accessing Applications Using APIs**, and **(Advanced) Enabling WebUI Authentication**.

### Advantages

- Easy to deploy

  The deployment process is simple. With the serverless solution, you can experience Stable Diffusion without server management and O&M.
- Open-source and customizable

  You can easily implement personalized AI painting based on multiple customized and advanced application scenarios.

### Constraints

Currently, Stable Diffusion application is only available in the **CN East-Shanghai1** region. Ensure that all related resources are deployed in this region.

### 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 potential violations. You are advised to use Stable-Diffusion with Huawei Cloud Moderation to review the generated result. For details, see **Image Moderation (V3)**.

## 4.5.2 Resource and Cost Planning

Plan resources and costs based on your requirements. For details, see **Table 4-3**.

**Table 4-3** Resource and cost planning

| Resource | Description | Billing | Mandatory |
| --- | --- | --- | --- |
| Function Graph | • Function type: container image-based HTTP function<br>• Region: **CN East-Shanghai1**<br>• Quantity: 2 (functions are automatically generated after the application is created) | • Billing mode: Pay-per-use<br>• The first 1 million invocations are free of charge in a month. For details about the billing items, see **Pay-per-Use Billing**. | Yes |
| API Gateway (APIG) | • Version: dedicated gateway<br>• Region: **CN East-Shanghai1**<br>• Quantity: 1 | • Billing mode: Select yearly/monthly or pay-per-use based on service requirements.<br>• For details about the billing modes and standards, see **APIG Billing Modes**. | Yes |
| Domain Name Service (DNS) | Public domain name resolution | Free | Mandatory for public domain name resolution with DNS |
| Virtual Private Cloud (VPC) | • Region: **CN East-Shanghai1**<br>• Number of subnets: 1<br>• Quantity: 1 | • VPC: free<br>• Subnet: free | Mandatory when a custom model is used or there are multiple users. |

| Resource | Description | Billing | Mandatory |
| --- | --- | --- | --- |
| Scalable File Service (SFS) | • Region: **CN East-Shanghai1**<br>• File system type: Available SFS Turbo file system<br>• Type: 250 MB/s/TiB<br>• Capacity: 1.2 TB<br>• Quantity: 1 | • Billing mode: Pay-per-use<br>• For details about the billing items, see **SFS Pay-per-Use Billing**. | Mandatory for uploading and using a custom model |
| Elastic Cloud Server (ECS) | • Region: **CN East-Shanghai1**<br>• OS: Public image EulerOS 2.5 64bit (40 GiB)<br>• Number of security groups: 1<br>• Quantity: 1 | • Billing mode: Pay-per-use<br>• Creating a security group: Free<br>• The instance type, storage specifications, and whether to enable public access are selected based on service requirements. For details about the billing items and standards, see the **ECS Pay-per-Use Billing**. | Mandatory when an ECS is used as the NFS server |

## 4.5.3 Procedure

Table 4-4 describes the procedure for deploying the Stable Diffusion application using FunctionGraph. The advanced process is oriented to specific service scenarios.

**Table 4-4** Procedure for deploying the Stable Diffusion application

| Step | Description |
| --- | --- |
| **Deploying and Using the Stable Diffusion Application** | You can create an application using the Stable Diffusion template in FunctionGraph Applications page to use the default model and temporary domain names. |

| Step | Description |
| --- | --- |
| **(Optional) Binding a Custom Domain Name** | To bind a custom domain name to access the application, perform the following operations:<br>1. Prepare a custom domain name.<br>2. Configure domain name resolution.<br>3. Bind the custom domain name. |
| **(Optional) Uploading a Custom Model** | To use a custom model, perform the following operations:<br>1. Create a VPC and subnet.<br>2. Create an SFS Turbo file system.<br>3. Initialize the file system mounted to the custom model.<br>4. Upload and load a custom model. |
| **(Advanced) Using ECS as an NFS Server to Isolate Resources of Multiple Users** | To isolate resources among multiple users, you can mount an ECS as the source of a file system. You can set a shared NFS path to manage resources of multiple users. The operations are as follows:<br>1. Buy an ECS.<br>2. Configure NFS sharing for an ECS.<br>3. Mount an ECS to the Stable Diffusion application function.<br>4. Upload and load a model. |
| **(Advanced) Mounting an SFS File System to Multiple Users** | To share model resources among multiple users, you can mount the same SFS Turbo file system to each user. In addition, you can set the path for saving the results of individual applications to isolate the inference results. To do so, perform the following operations:<br>1. Create a multi-user configuration file.<br>2. Modify environment variables and use the new configuration file.<br>3. Change the path for saving the result. |
| **(Advanced) Enabling WebUI Authentication** | To enhance application security, you can configure function environment variables to enable WebUI authentication. When accessing the WebUI, you need to enter the username and password to perform drawing operations. |
| **(Advanced) Accessing Applications Using APIs** | To access applications using APIs, you can enable and configure concurrency parameters by configuring function environment variables. |

# 4.5.4 Deploying and Using the Stable Diffusion Application

Use the Stable-Diffusion template to create an application and configure the required agency. After the application is created, you can use the built-in default models and temporary domain names.

## Step 1: Creating a Cloud Service Agency for FunctionGraph

To deploy a Stable Diffusion application, you need to configure an agency for FunctionGraph to allow it to use other cloud services.

**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 **serverless_trust**.
- **Agency Type**: Select **Cloud service**.
- **Cloud Service**: Select **FunctionGraph**.
- **Validity Period**: Select **Unlimited**.
- (Optional) **Description**: Enter **Stable Diffusion**.

**Step 3** Click **OK**. The system displays a message indicating that the creation is successful. Click **Authorize**.

**Step 4** On the **Select Policy/Role** page, search for the policies listed in **Table 4-5** as required, select them, and click **Next**.

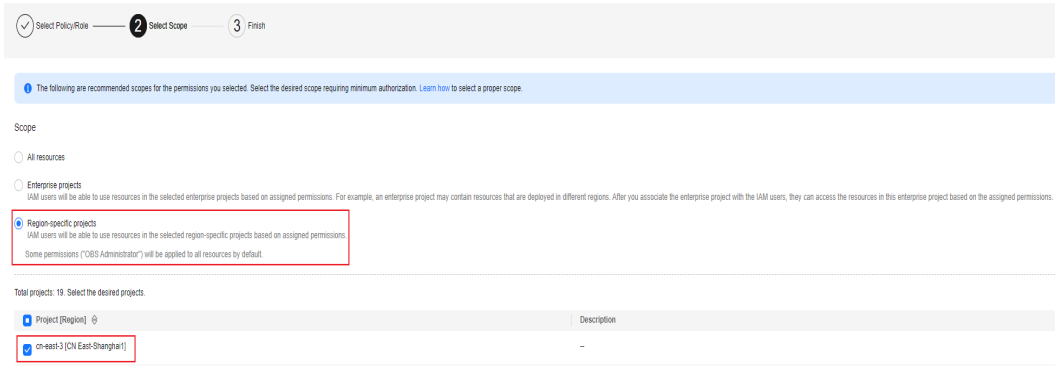
**Table 4-5** Policies and related description

| Policy | Description | Mandatory |
| --- | --- | --- |
| SWR Admin | SoftWare Repository for Container (SWR) administrator with full permissions. | Yes |
| VPC Administrator<br>(The system will select the Server Administrator system role that the system role depends on. You do not need to manually cancel the selection.) | VPC Administrator: administrator of the VPC service.<br>Server Administrator: server administrator. | Mandatory for uploading and using a custom model. |
| SFS FullAccess | Full permissions for SFS | Mandatory for mounting an SFS file system. |
| SFS Turbo FullAccess | Full permissions for SFS Turbo. | Mandatory for mounting an SFS file system. |

**Step 5** On the **Select Scope** page shown in **Figure 4-21**, select **Region-specific projects**, select **cn-east-3 [CN East-Shanghai1]**, and click **OK**.

**Figure 4-21** Region-specific projects




**Step 6** After the authorization is successful, click **Finish** to view the authorization record.

**----End**

## Step 2: Buying a Dedicated Gateway

Buy a dedicated gateway based on your service requirements. For details, see **Creating a Gateway**. Pay attention to the following configuration during the purchase:

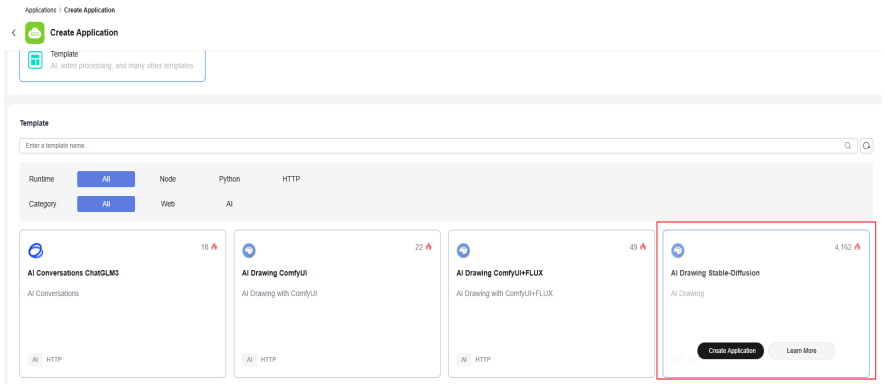
- Region: **CN East-Shanghai1**
- **AZ**: The AZ must be the same as that of the created subnet. In this example, **AZ 1** is used.
- **Public Inbound Access**: In this example, enable the public inbound access. Select the bandwidth as required.

## Step 3: Creating an Application Using the Stable Diffusion Template

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

**Step 2** As shown in **Figure 4-22**, find the **AI Drawing Stable-Diffusion** template and click **Create Application**. Read the displayed description carefully, select the check box, and click **Agree and Continue**.

**Figure 4-22** Selecting the AI Drawing Stable-Diffusion template



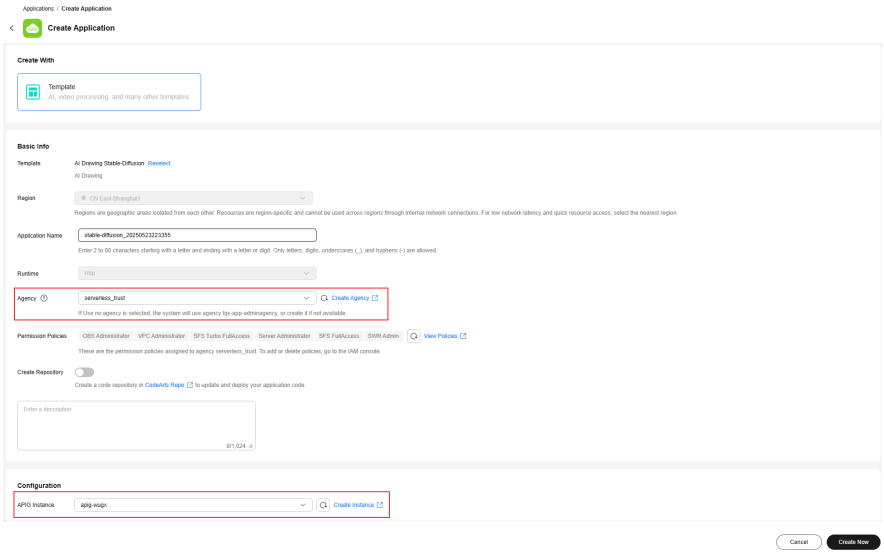

  

> **NOTE**
>
> If the **Service Subscription** dialog box is displayed, read the description and click **Enable Now**.

**Step 3** On the **Configure Application** page, set application parameters as shown in **Figure 4-23**. After setting the parameters, click **Create Now** in the lower right corner of the page.

- **Application Name**: Enter a custom name or use the default name.
- **Agency**: Select the **serverless_trust** agency created in **Step 1: Creating a Cloud Service Agency for FunctionGraph**.
- **APIG Instance**: Select the gateway created in **Step 2: Buying a Dedicated Gateway**.

**Figure 4-23** Application configuration




**Step 4** Wait until the application is created. The application contains resources such as functions, API gateways, and triggers, as shown in **Figure 4-24**. For details about the key resources of FunctionGraph, see **Table 4-6**.

**To make it easier for you to experience the feature, a temporary domain name is allocated to you. The temporary domain name can be used only for testing and is valid for 30 days.** To allow long-term access to applications, bind a custom domain name. For details, see **(Optional) Binding a Custom Domain Name**.

**Figure 4-24** Application created

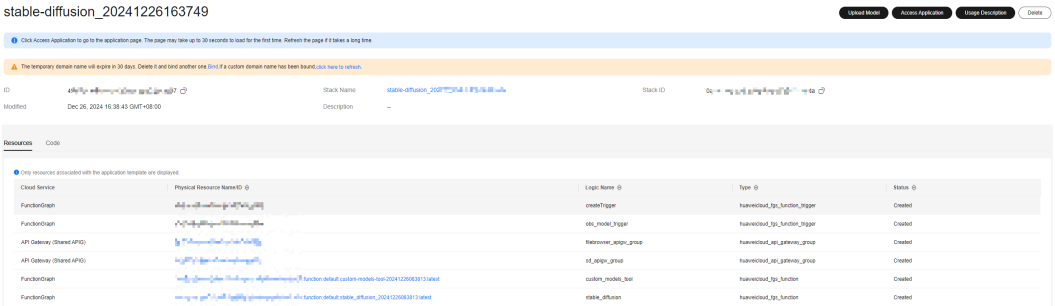


**Table 4-6** Key function services

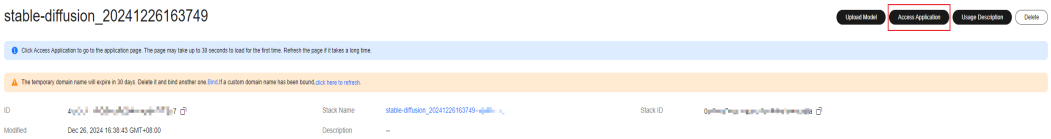
| Resource Logical Name | Description |
| --- | --- |
| stable_diffusion | Accesses the Stable Diffusion WebUI through the APIG trigger. |
| custom_models_tool | Manages Stable Diffusion resources (uploading models and plug-ins and downloading images) via the APIG trigger. |

**----End**

## Step 3: Using the Default Model and Temporary Domain Name for AI Drawing

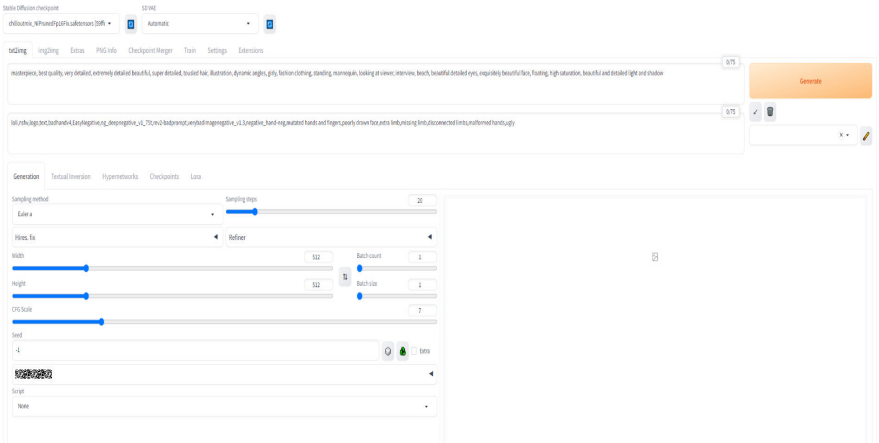
**Step 1** On the application details page shown in **Figure 4-25**, click **Access Application**. The Stable Diffusion WebUI page is displayed. The first cold start takes about 30 seconds. If the loading times out, refresh the page.

**Figure 4-25** Accessing the application




**Step 2** On the **txt2img** tab page, enter the corresponding prompt word and reverse prompt (in both Chinese and English), and click **Generate** on the right. An image that matches the prompt word description is generated.

**Figure 4-26** Stable Diffusion WebUI



**NOTICE**

The application created in the preceding steps can only use the default model built in the application for AI drawing. If you want to use more custom models, you need to mount an external file system to the application. For details, see **(Optional) Uploading a Custom Model**.

**----End**

## 4.5.5 (Optional) Binding a Custom Domain Name

To enable private and public access to the Stable Diffusion application, bind a custom domain name to the application.

### Prerequisites

The Stable Diffusion application has been deployed. For details, see **Deploying and Using the Stable Diffusion Application**.

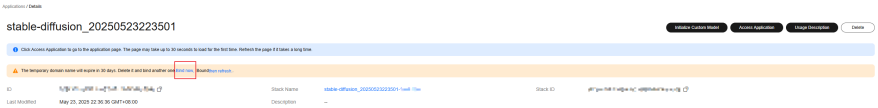
### Step 1: Preparing a Custom Domain Name

Apply for a public network domain name through the domain name registrar. Ensure that the domain name is available.

### Step 2: Configuring Domain Name Resolution

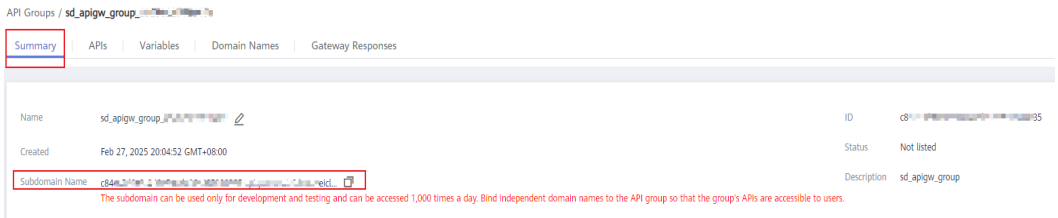
**Step 1** After an application is created, click **Bind now** in the prompt, as shown in **Figure 4-27**. The APIG console is displayed.

**Figure 4-27** Clicking the Bind now button




**Step 2** On the displayed APIG console, click the **Summary** tab and copy the subdomain name as shown in **Figure 4-28**.

**Figure 4-28** Copying the subdomain name



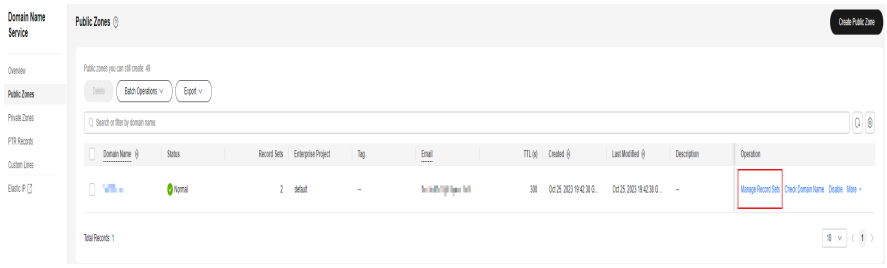

**Step 3** Log in to the DNS console, choose **Public Zones**, and click **Manage Record Set** on the right of the purchased domain name as shown in **Figure 4-29**.

**NOTE**

If you use a domain name not registered with Huawei Cloud, see **Creating a Public Domain Name**.

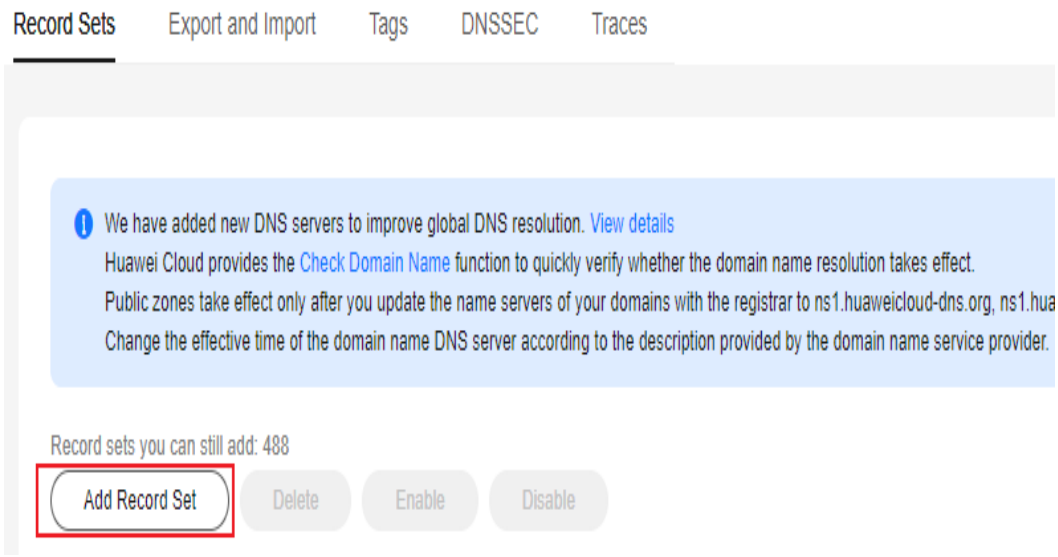
  

**Figure 4-29** Clicking Manage Record Sets




**Step 4** On the **Record Sets** tab, click **Add Record Set**, as shown in **Figure 4-30**.

**Figure 4-30** Adding a record set



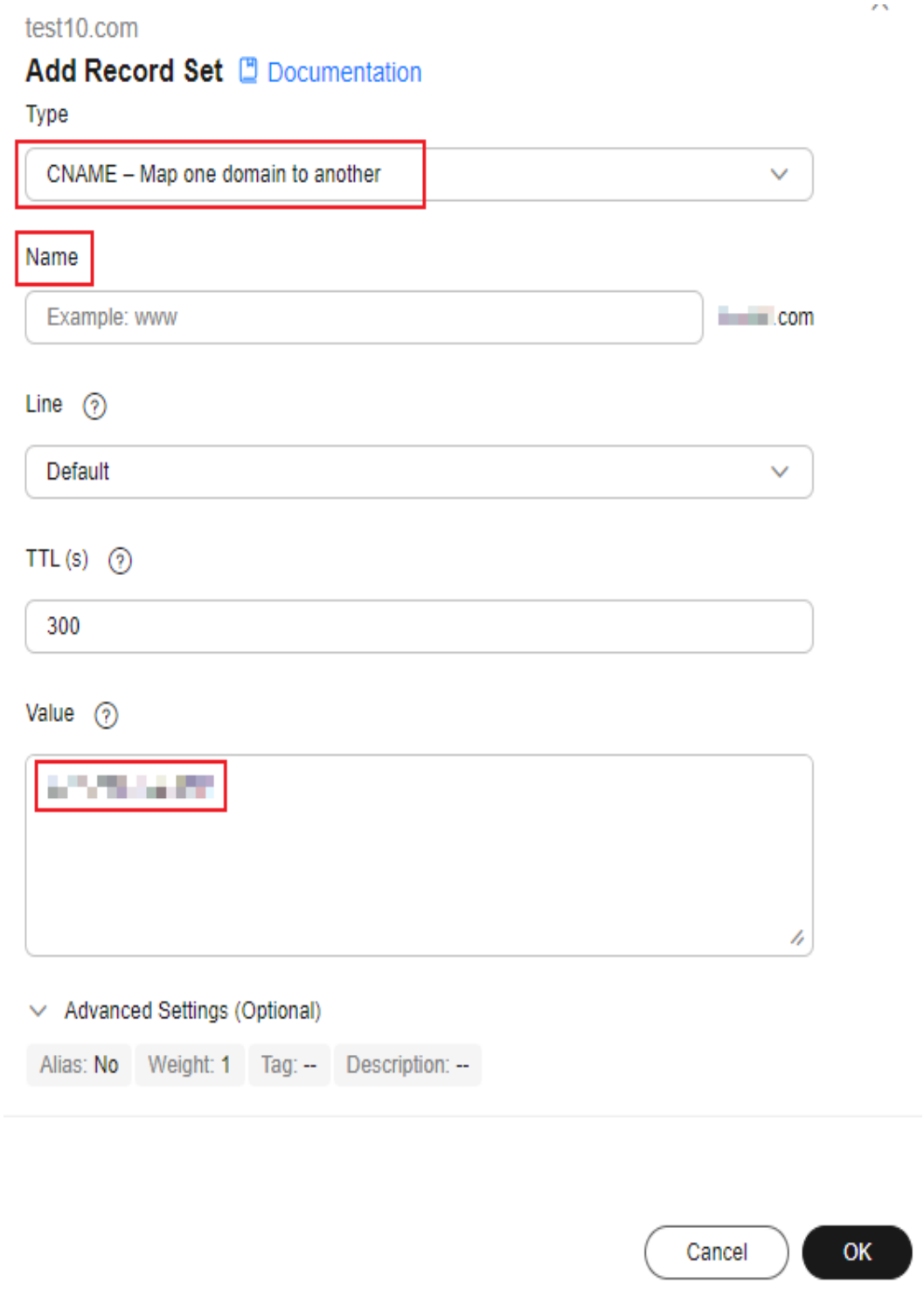

**Step 5** In the displayed **Add Record Set** dialog box, configure the information.

- **Type**: Select **CNAME – Map one domain to another**.
- Name: Set it according to **Adding a CNAME Record Set**.
- Value: Enter the subdomain name copied in **Step 2**.

Retain the default values for other parameters and click **OK** to complete domain name resolution as shown in **Figure 4-31**.

**Figure 4-31** Configuring the record set




**----End**

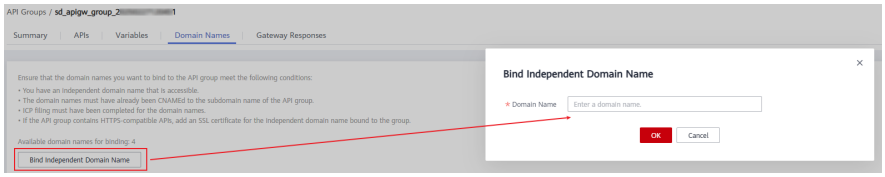
## Step 3: Bind a Custom Domain Name

**Step 1** Return to the APIG console, as shown in **Figure 4-32**. On the **Domain Names** tab page, click **Bind Independent Domain Name**, enter a custom domain name, and

click **OK**. After the binding is successful, you can unbind the temporary domain name.

**Figure 4-32** Binding an independent domain name




**Step 2** Return to the application details page on the FunctionGraph console, as shown in Figure 4-33. Click **then refresh** to access the application using the custom domain name.

**Figure 4-33** Refreshing the bound domain name


**----End**

# 4.5.6 (Optional) Uploading a Custom Model

The Stable Diffusion application has default models. To use more custom models, initialize the file system mounted to the custom model in the created application and upload the custom models.

## Prerequisites

1. FunctionGraph cloud service agencies contain the **SWR Admin**, **VPC Administrator**, **Server Administrator**, **SFS FullAccess**, and **SFS Turbo FullAccess** permissions.
2. The Stable Diffusion application has been deployed. For details, see **Deploying and Using the Stable Diffusion Application**.

## Step 1: Creating a VPC and Subnet

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

**Step 2** On the **Create VPC** page, set parameters by referring to **Table 4-7** and retain the default values for other parameters.

**Table 4-7** Configuring the VPC and subnet

| Category | Parameter | Description | Example Value |
| --- | --- | --- | --- |
| Basic Information | Region | Mandatory<br>Region where the VPC and its subnets are deployed. Currently, the Stable Diffusion application can be deployed only in **CN East-Shanghai1**. | CN East-Shanghai1 |
| | Name | Mandatory.<br>VPC name. The following requirements must be met:<br>● Must contain 1 to 64 characters.<br>● Can contain letters, numbers, underscores (_), hyphens (-), and periods (.). | vpc-fg |
| | IPv4 CIDR Block | Mandatory.<br>Set the IPv4 CIDR block of the VPC based on the suggestions on the page. When selecting the VPC CIDR block, consider the following two points:<br>● Number of IP addresses: Reserve sufficient IP addresses for subsequent business growth.<br>● IP address ranges: Avoid IP address conflicts if you need to connect a VPC to an on-premises data center or connect two VPCs. | 192.168.x.x/16 |
| | Enterprise Project | Mandatory.<br>An enterprise project facilitates project-level management and grouping of cloud resources and users. The default project is **default**. | default |
| Subnet Setting 1 | Subnet Name | Mandatory.<br>The subnet name. The following requirements must be met:<br>● Must contain 1 to 64 characters.<br>● Can contain letters, numbers, underscores (_), hyphens (-), and periods (.). | subnet-fg |

| Category | Parameter | Description | Example Value |
| --- | --- | --- | --- |
| | AZ | Mandatory.<br>AZs in the same VPC can communicate with each other over the intranet. AZs are physically isolated from each other. If the service requirements are high, you are advised to select multiple AZs. In this example, one AZ is selected. | AZ1 (center) |
| | IPv4 CIDR Block | Mandatory.<br>The IPv4 CIDR block of the subnet, which must be within the VPC CIDR block. The mask length of the subnet CIDR block ranges from the mask length of the VPC CIDR block to 29. You can select a value as required. | 192.168.x.x/24 |

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

**----End**

## Step 2: 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** On the **Create File System** page, set parameters by referring to **Table 4-8**. Retain the default values for other parameters. For details about other parameters, see **Creating an SFS Turbo File System**.

**Table 4-8** File system parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Billing Mode | Mandatory.<br>• Pay-per-use is suitable for flexible usage.<br>• Yearly/Monthly is ideal when your resource usage duration is predictable. | Select **Pay-per-use**. |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Region | Mandatory.<br>Region where the file system is deployed. Currently, the AI drawing application can be deployed only in **CN East-Shanghai1** and must be in the same VPC as the created VPC. | CN East-Shanghai1 |
| Project | Mandatory.<br>Region where the project is deployed. Select the default synchronization settings based on the region. | CN East-Shanghai1 (default) |
| AZ | Mandatory.<br>The value must be the same as the AZ of the created subnet. | Select **AZ1**. |
| Type | Mandatory.<br>Select the file system type and performance based on the recommended scenario and actual situation. In this example, all file system types are supported. You are advised to select the 250 MB/s/TiB type that is suitable for most application scenarios. | 250 MB/s/TiB |
| Capacity | Mandatory.<br>Maximum capacity of a single file system. Select a value based on the site requirements. The value must be an integer multiple of 1.2 and in the range from 1.2 to 1023.6. | 1.2 |
| Enterprise Project | Mandatory.<br>The value must be the same as that selected when the VPC is created. | default |
| VPC | Mandatory.<br>VPC and subnet to which the file system belongs. Select the VPC and subnet created in **Step 1: Creating a VPC and Subnet**. | vpc-fg;<br>subnet-fg(192.168.x.x/24) |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Name | Mandatory.<br>Name of the file system. The following requirements must be met:<br>• Must start with a letter and contain 4 to 64 characters.<br>• Only letters, digits, underscores (_), and hyphens (-) are allowed. | sfs-turbo-fg |

**Step 3** After the parameters are configured, click **Create Now**. Confirm the information and click **Submit**. Wait until the file system creation task is submitted successfully.

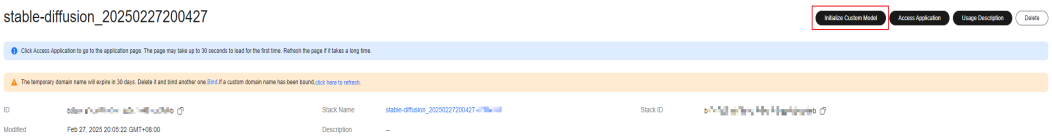
**----End**

## Step 3: Initializing a Custom Model

**Step 1** Log in to the **FunctionGraph console** and select **CN East-Shanghai1** region. In the navigation pane on the left, choose **Applications**. Click the name of the application that is successfully created and needs to be initialized.

**Step 2** On the application details page, click **Initialize Custom Model** as shown in **Figure 4-34**, read the description in the displayed dialog box, select the check box, and click **OK**.

**Figure 4-34** Initializing a custom model



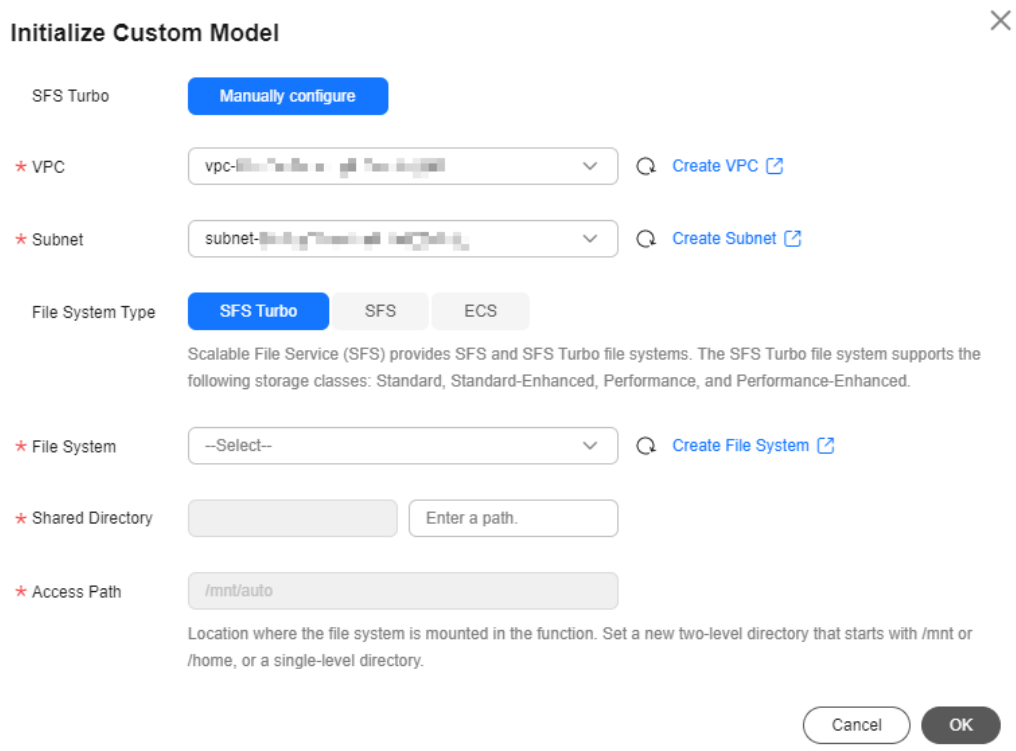

**Step 3** In the **Initialize Custom Model** dialog box, set the following parameters:

- **VPC**: Select **vpc-fg (192.168.x.x/16)**.
- Subnet: Select **subnet-fg (192.168.x.x/24)**.
- **File System Type**: Select **SFS Turbo**.
- **File System**: Select **sfs-turbo-fg**.

Retain the default values for other parameters, as shown in **Figure 4-35**. After the configuration is complete, click **OK**.

**Figure 4-35** Initializing a custom model




**Step 4** Return to the application details page. If the **Initialize Custom Model** button changes to **Upload Model**, the initialization is successful. Click **Access Application** to go to the WebUI. The system automatically creates the directories and files required for application deployment in the file system.

NOTE

You do not need to perform any operation after logging in to the Stable Diffusion WebUI. This operation is used to load directories and files in the file system for uploading custom models.

**----End**

## Step 4: Uploading and Loading a Custom Model

**Step 1** Return to the application details page and click **Upload Model**. The file management page is displayed. The default username and password are **admin**. Change the password on the setting page after login to ensure data security.

**Step 2** Table 4-9 lists some key directories related to the upload of custom models. You can upload model files to the corresponding directories.

**Table 4-9** Key directory path

| Path | Description |
| --- | --- |
| sd/models/Stable-diffusion | Stores Checkpoint model files. |
| sd/models/VAE | Stores VAE files. |

| Path | Description |
| --- | --- |
| sd/models/Lora | Stores Lora files. |
| sd/extensions | Stores plug-ins. |
| sd/outputs | Stores outputs. |

**Step 3** After the upload is complete, return to the Stable Diffusion WebUI. Click the refresh button to load the model or open the WebUI again. After the loading is successful, you can view and select the new model for AI drawing. The loading may take a long time.

**----End**

# 4.5.7 (Advanced) Using ECS as an NFS Server to Isolate Resources of Multiple Users

## Application Scenario

FunctionGraph functions can be mounted to SFS file systems or NFS shared paths shared by ECSs. In multi-user scenarios, ECSs can be used to effectively manage resources of multiple users. You can configure specific permissions to meet the requirements of strong isolation between users.

## Prerequisites

1. Each user must have the **SWR Admin**, **VPC Administrator**, **Server Administrator**, **SFS FullAccess**, and **SFS Turbo FullAccess** permissions in their agencies.
2. Each user must complete the steps in **Deploying and Using the Stable Diffusion Application** to create an application.
3. You have completed steps in **Step 1: Creating a VPC and Subnet**.

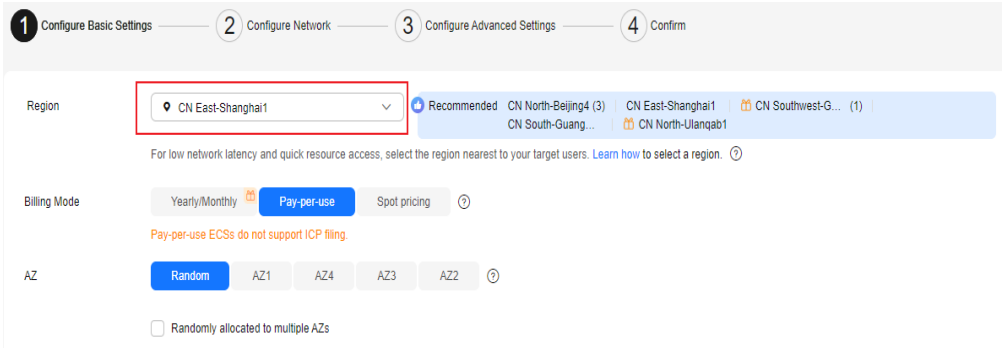
## Step 1: Buying an ECS

Go to the **Buy ECS** page. The instance type and whether to enable public access can be selected based on service requirements. Follow the instructions below. For details about other parameters, see **Setting ECS Purchase Parameters**.

- **Basic Configuration**: See **Figure 4-36**. In this example, the **Pay-per-use** billing mode is used, and the region is **CN East-Shanghai1**.

**Figure 4-36** Basic configuration




- **OS: EulerOS 2.5 64bit(40 GiB)** is used in this example.

  NOTE

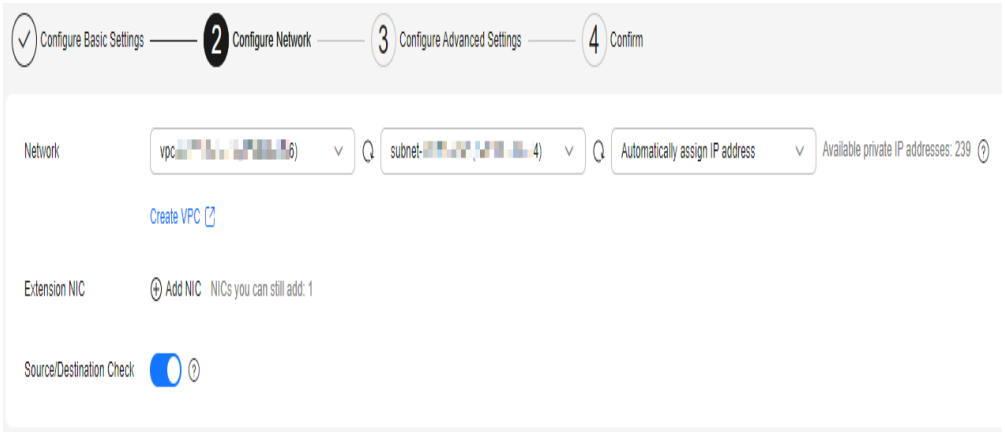
  Some Linux commands may vary with the image version.

- **Storage & Backup**: The size of most model files ranges from 1 GB to 10 GB. You are advised to select the system disk capacity based on the actual requirements and add data disks by referring to **Figure 4-37**.

**Figure 4-37** Configuring a system disk


- **Network**: Select the VPC and subnet created in **Step 1: Creating a VPC and Subnet**, as shown in **Figure 4-38**.

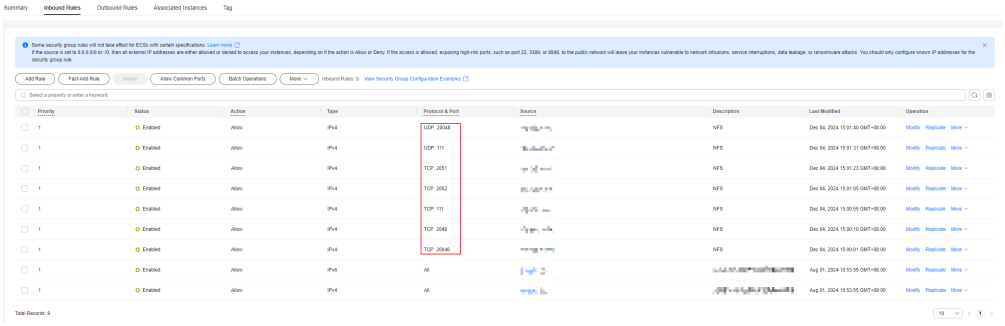
**Figure 4-38** Configuring network




- **Security Group**: Create a security group by referring to **Figure 4-39**. The inbound rule allows IP addresses in the subnet to access ports **111**, **2049**, **2051**, **2052**, and **20048** to support the NFS service. Other ports, such as port **22** for SSH and SFTP and port **21** for FTP, can be configured based on the actual protocol, port, and source address.

**Figure 4-39** Configuring a security group



## Step 2: Configuring NFS Sharing for the ECS

After the ECS is purchased, you can configure NFS sharing. The following uses **user1** and **user2** as an example. You can add or delete users as required.

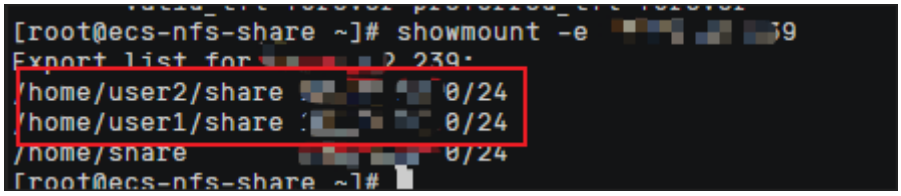
1. Add **user1** and **user2** and create the **home** directory.
   ```
   useradd -m user1 && useradd -m user2
   ```
2. Change the passwords of **user1** and **user2**.
   ```
   passwd user1
   passwd user2
   ```
3. Create a shared directory for users and change the operation permission on the shared 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 used as a subdirectory of the **home** directory to restrict other users' operations and ensure that the function has the operation permission after being mounted to the directory. Therefore, setting the permission to **777** does not cause excessive permissions.
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) // Enter the created subnet CIDR block.
   /home/user2/share xx.xx.xx.xx/xx (rw) // Enter the created subnet CIDR block.
   ```
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 information shown in **Figure 4-40** is displayed, the NFS sharing is successfully configured.
   ```
   showmount -e xx.xx.xx.xx //Replace IP address with the private IP address of the server host.
   ```

**Figure 4-40** Checking sharing information



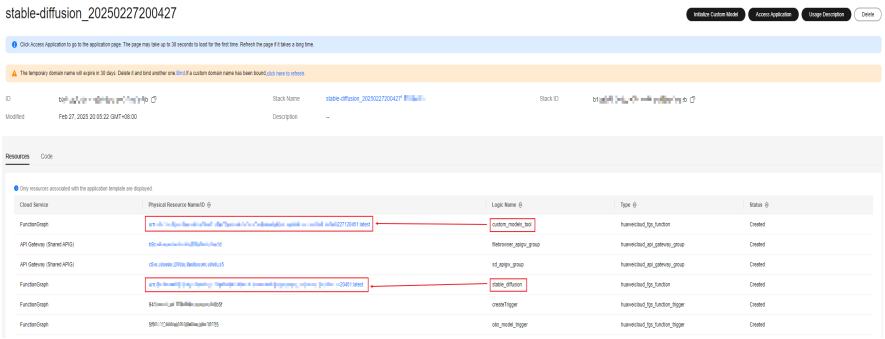

## Step 3: Mounting an ECS to the Stable Diffusion Application

Log in to the **FunctionGraph console** and go to the application center. **user1** and **user2** have created a Stable Diffusion application under their respective accounts. The following uses **user1** as an example. The operations for **user2** are the same.

**Step 1** Go to the application details page, find the function resources whose **Logic Name** are **stable_diffusion** and **custom_models_tool** in the **Resources** tab. Click the links to go to the function details page, as shown in **Figure 4-41**.

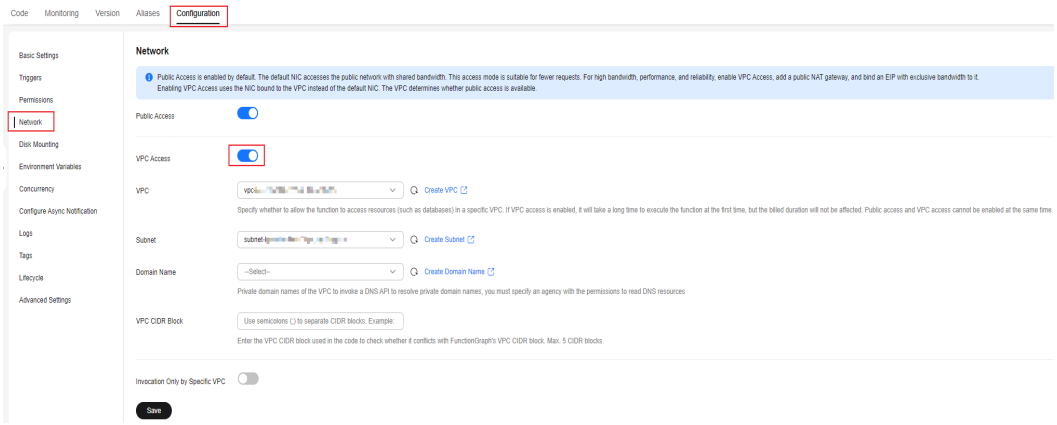
The operations of the two functions are the same. The **stable_diffusion** function is used as an example.

**Figure 4-41** Functions of user1




**Step 2** On the function details page shown in **Figure 4-42**, choose **Configuration** > **Network**, enable **VPC Access**, select the VPC and subnet used in **Step 1: Buying an ECS**, and click **Save**.

**Figure 4-42** Configuring network




**Step 3** In the navigation tree on the left of the **Configuration** page, choose **File Systems** > **Mount File System**. After the configuration is complete, click **OK**.

- **File System Type**: Select **ECS**.
- **ECS**: Select the ECS created in **Step 1: Buying an ECS**.
- **Shared Directory**: Enter **/home/user1/share** (e**/home/user2/share** for user2).
- **Access Path**: Enter **/mnt/auto**.

NOTE

If an SFS Turbo file system has been mounted, unmount the SFS Turbo file system after the ECS is successfully mounted and release the SFS Turbo file system resources in a timely manner to avoid continuous charging.

**Step 4** Set the **custom_models_tool** function by referring to **Step 1** to **Step 3**.

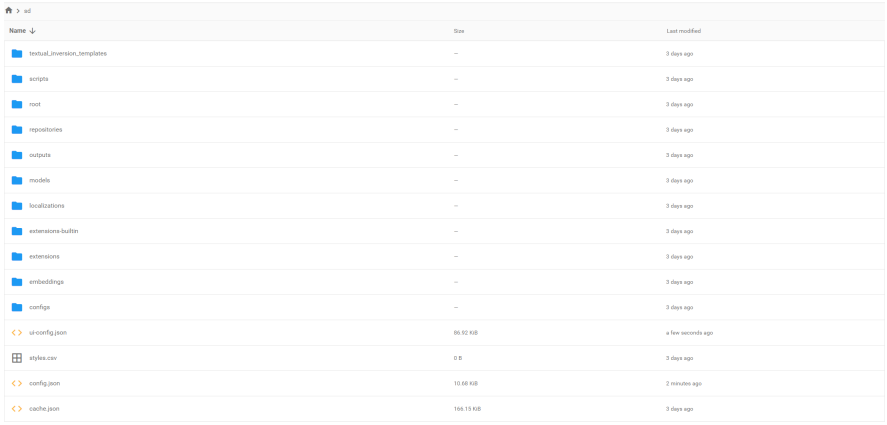
**----End**

## Step 4: Uploading and Loading a Model

**Step 1** Return to the application details page and click **Access Application** to go to the WebUI. The function automatically creates the directories and files required by the application in the mounting directory.

**Step 2** After the page is successfully loaded, return to the application details page and click **Upload Model** to open the file management tool. The default username and password are both **admin**. After login, change the password on the setting page to ensure data security. **Figure 4-43** shows the **sd** directory related to the application.

**Figure 4-43** File management tool



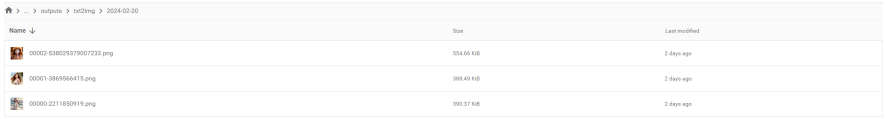

**Step 3** Upload the model and plugin files to the corresponding directories. For details about some key directories, see **Table 4-9**.

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

**Step 5** Click **Generate** in the upper right corner to start AI drawing. The result image is automatically saved to the **/home/user1/share/sd/outputs/txt2img/202x-xx-xx** directory, as shown in **Figure 4-44**.

**Figure 4-44** Directory for saving images



**----End**

# 4.5.8 (Advanced) Mounting an SFS File System to Multiple Users

## Application Scenario

Storing model files for each user uses too much space. Mount the same SFS file system to users' applications so that they can share model file resources. To avoid inference results of different users from affecting each other, you can change the result save path on the Stable Diffusion WebUI.

## Prerequisites

1. Each user must have the **SWR Admin**, **VPC Administrator**, **Server Administrator**, **SFS FullAccess**, and **SFS Turbo FullAccess** permissions in their agencies.
2. Each user must complete the steps in **Deploying and Using the Stable Diffusion Application** to create an application.
3. Each user must use the same SFS file system to initialize the mounted file system of the custom model. For details, see **Step 3: Initializing a Custom Model**. The **sd** directory already exists in the mounted SFS file system.

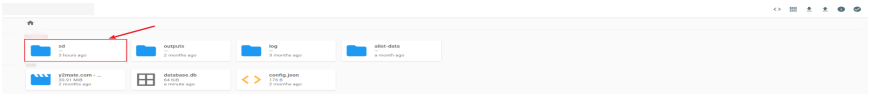
## Step 1: Creating a Multi-user Configuration File

This example uses two users **user1** and **user2**. You can add or delete users as required.

**Step 1** Select a user, go to the details page of the Stable Diffusion application whose custom model has been initialized, click **Upload Model**, and log in to the file system.

**Step 2** Go to the **sd** directory, as shown in **Figure 4-45**.

**Figure 4-45** Entering the sd directory



**Step 3** As shown in **Figure 4-46**, find the **config.json** file, select it, and copy it to any directory. In this case, the file is still stored in the **sd** directory and renamed **config_user1.json**. As shown in **Figure 4-47**, copy the file for user2 and rename it **config_user2.json**.

**Figure 4-46** Copying the config.json File

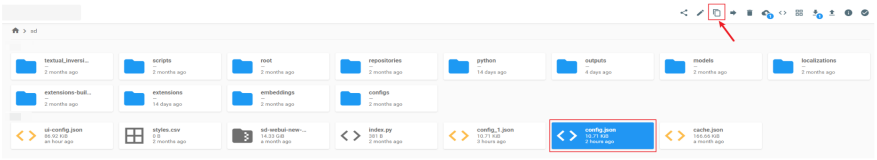
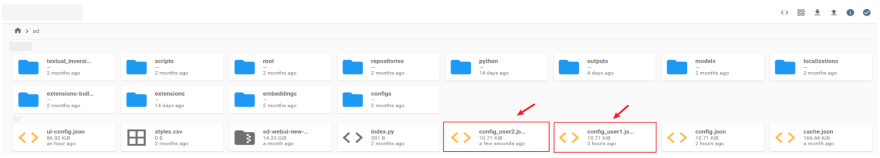


**Figure 4-47** Creating the config_user1.json and config_user2.json files



**----End**

## Step 2: Modifying Environment Variables and Using the New Configuration File

**Step 1** Go to the Stable Diffusion application details page, find the function resource whose logic name is **stable_diffusion** in the **Resource** tab, and click the link to go to the function details page.

**Step 2** On the **Configuration** > **Environment Variables** tab page, click **Edit Environment Variable**. In the dialog box that is displayed, click **Add**. Set the environment variable for **user1** based on **Table 4-10** and **user2** based on **Table 4-11** and click **OK**.

**Table 4-10** Environment variables of the new configuration file used by user1

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

**Table 4-11** Environment variables of the new configuration file used by user2

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

**----End**

NOTE

To enable **(Advanced) Enabling WebUI Authentication** and **(Advanced) Accessing Applications Using APIs** at the same time, you can set environment variables at the same time. For details, see **Table 4-14**.

### Step 3: Changing the Result Save Path

After the preceding configurations are complete, the two users can share the model files in the same SFS file system. To further isolate the inference results of different users, you can change the result save path in the settings.

## 4.5.9 (Advanced) Enabling WebUI Authentication

WebUI authentication is disabled by default for Stable Diffusion applications deployed in the application center. To prevent functions from being abused due to domain name leakage, configure function environment variables to enable WebUI authentication. For details, see **Configuring Environment Variables**.

### Enabling WebUI Authentication

**Step 1** Go to the Stable Diffusion application details page, find the function resource whose logic name is **stable_diffusion** in the **Resource** tab, and click the link to go to the function details page.

**Step 2** Choose **Configuration** > **Environment Variables**, click **Edit Environment Variable**. In the dialog box that is displayed, click **Add**, add the information in the following table, and click **OK**.

**Table 4-12** Environment variables for enabling WebUI authentication

| Key | Value | Description |
| --- | --- | --- |
| EXTRA_ARGS | --gradio-auth user1:password1 | Enter the username in **user1** and the password in **password1**. |

**Step 3** After the setting is complete, you need to enter the username and password to access the WebUI.

**----End**

## 4.5.10 (Advanced) Accessing Applications Using APIs

API access is disabled by default for the Stable Diffusion application deployed in the application center. You can enable API access by **Configuring Function Environment Variables**.

### Accessing the Application Using an API

**Step 1** Go to the Stable Diffusion application details page, find the function resource whose logic name is **stable_diffusion** in the **Resource** tab, and click the link to go to the function details page.

**Step 2** Choose **Configuration** > **Environment Variables**, click **Edit Environment Variable**. In the dialog box that is displayed, click **Add**, add the information in the following table, and click **OK**.

**Table 4-13** Environment variables for accessing applications using APIs

| Key | Value | Description |
| --- | --- | --- |
| EXTRA_ARGS | --api --api-auth username1:password1,username2:password2 --nowebui | Enter the usernames in **username1** and **username2**, and enter the passwords in **password1** and **password2**. Use commas (,) to separate the usernames and passwords of multiple users. |

**Step 3** After the configuration is complete, you need to enter the username and password when accessing the application using APIs.

**----End**

## Configuring Concurrency Parameters

Configure concurrency parameters by referring to **Configuring Concurrency Parameters**. The recommended parameters in WebUI and API modes are as follows:

- WebUI mode
  - Set the value of **Max. Requests per Instance** to 100 or higher. According to testing results, in single-user scenarios, you are advised to set the **Max. Requests per Instance** to around 15. For multi-user scenarios, the value should be configured to 100 or higher.
  - Set the value of **Max. Instances per Function** to 1. In WebUI mode, the task progress is continuously monitored during image generation. If there are multiple instances, requests may be disordered, which may cause obstacles in progress display and final result display.
- API mode
  - Set the value of **Max. Requests per Instance** between 1 and 5 to ensure that a single instance does not process too many requests. When the concurrent requests reach the upper limit, new instances are triggered to ensure image generation efficiency.
  - Set the value of **Max. Instances per Function** to **400** (default). The value can be changed as needed.

## Accessing an Application Using APIs and Enabling WebUI Authentication

To access an application using APIs and enable WebUI authentication, you can set environment variables by referring to **Table 4-14**.

**Table 4-14** Accessing an application using APIs and enabling WebUI authentication

| Key | Value | Description |
| --- | --- | --- |
| EXTRA_ARGS | --api --api-auth user1:password1 --gradio-auth user1:password1 | Enter the usernames in **username1** and **username2**, and enter the passwords in **password1** and **password2**. Use commas (,) to separate the usernames and passwords of multiple users. |

## 4.5.11 Disclaimer

1. The Stable-Diffusion, **Stable-Diffusion-WebUI**, and **image build project** used by this application are open-source projects in the community. Huawei Cloud only provides computing power support. For any question, seek help from the open-source community or solve the problems by yourself.
2. This practice is only an example for your reference and learning. If you want to use it in the production environment, optimize it according to the image build project. If you have any questions, **submit a service ticket**.
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.

# 4.6 Deploying an MCP Server Using FunctionGraph

## What Is MCP Server?

Model Context Protocol (MCP) is an open-source protocol that aims to provide context information to large language models (LLMs) in a standardized manner. The MCP server runs based on the model context protocol and can seamlessly integrate LLMs with external data sources and tools. By using standardized interactions, the MCP server helps LLMs obtain abundant context information. MCP servers are widely used. For example, the file system server can assist AI in analyzing project files, and the web search server can help AI obtain the latest information.

## Solution Overview

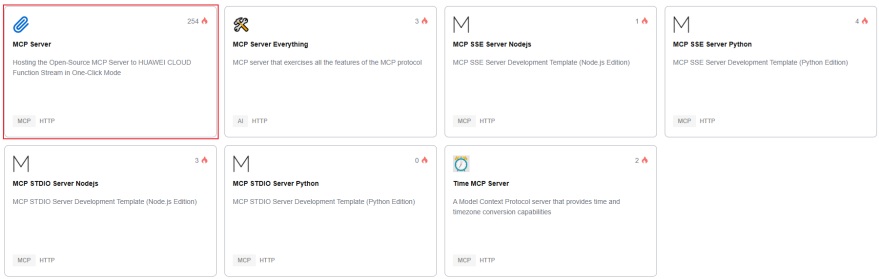
During digital transformation, if you use the traditional cloud server deployment model, you need to estimate the peak traffic and plan resources in advance. However, the static resource configuration is not suitable for services with unpredictable traffic, which may lead to underutilization of servers and waste of resources, affecting cost-effectiveness. FunctionGraph offers an efficient, flexible, and reliable solution for hosting MCP servers. With the serverless architecture, FunctionGraph can automatically adjust resource allocation based on actual

traffic, improving resource utilization, reducing resource idleness, and optimizing costs.

FunctionGraph provides application templates for deploying popular open-source MCP servers in one click. You can use API Gateway (APIG) to provide services externally. FunctionGraph not only simplifies the deployment process, but also automatically processes logs and monitoring data, allowing you to focus on core service logic development.

**Figure 4-48** MCP server template in the FunctionGraph application center




## Notes and Constraints

- Currently, the MCP server application can be deployed only in the **CN North-Beijing4** region. Ensure that all resources used in this practice are deployed in the region.
- Each application can run only one MCP server instance.

## Resource and Cost Planning

Table 1 describes the resources and costs required for deploying the MCP server application.

**Table 4-15** Resource and cost planning

| Resource | Description | Billing |
| --- | --- | --- |
| FunctionGraph | • Function type: HTTP function<br>• Example region: **CN North-Beijing4**<br>• Quantity: 1 | • Billing mode: Pay-per-use<br>• The first 1 million invocations are free of charge in a month. For details about the billing items, see **Pay-per-Use Billing**. |
| API Gateway (APIG) | • Version: dedicated gateway<br>• Example region: **CN North-Beijing4**<br>• Public inbound access: enabled<br>• Quantity: 1 | • Billing mode: Pay-per-use<br>• Select the instance specifications and bandwidth based on your service requirements. For details about the billing items and standards, see **APIG Billing Overview**. |

## Step 1: Creating a Dedicated Gateway

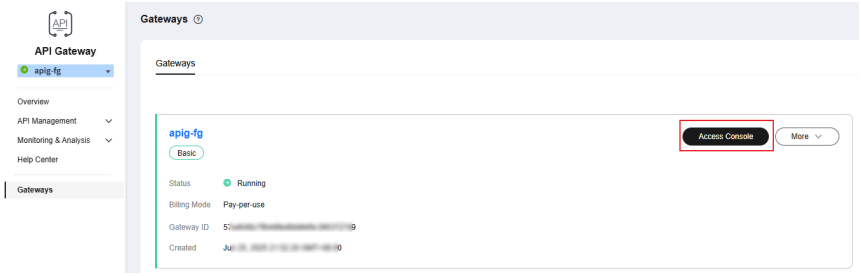
1. On the APIG console, purchase a dedicated gateway on the **Buy Gateway** page by referring to **Creating a Gateway**.

   Create a dedicated gateway named **apig-fg** by referring to the following parameters:

   - **Region**: **CN North-Beijing4**.
   - **Public Inbound Access**: Enable the public inbound access and select the bandwidth as required.

2. In the navigation pane on the left of the APIG console, choose **Gateways**. On the displayed page, click **Access Console**.

   **Figure 4-49** Viewing the gateway console

3. On the **Parameters** tab page, click **Modify** on the right of the **sse_strategy** parameter, change the parameter value to **On**, and click **Save** to enable the SSE transmission policy.

## Step 2: Create an MCP Server Application

1. Log in to the **FunctionGraph console** and select the **CN North-Beijing4** region.
2. In the navigation pane on the left, choose **Applications**. In the upper right corner, click **Create**. The page for selecting a template is displayed.
3. Find the **MCP Server** template and click **Create Application**. The page for creating an application is displayed.
4. Set application parameters by referring to **Table 4-16**. After the configuration is complete, click **Create Now**.

**Figure 4-50** Creating an MCP Server application

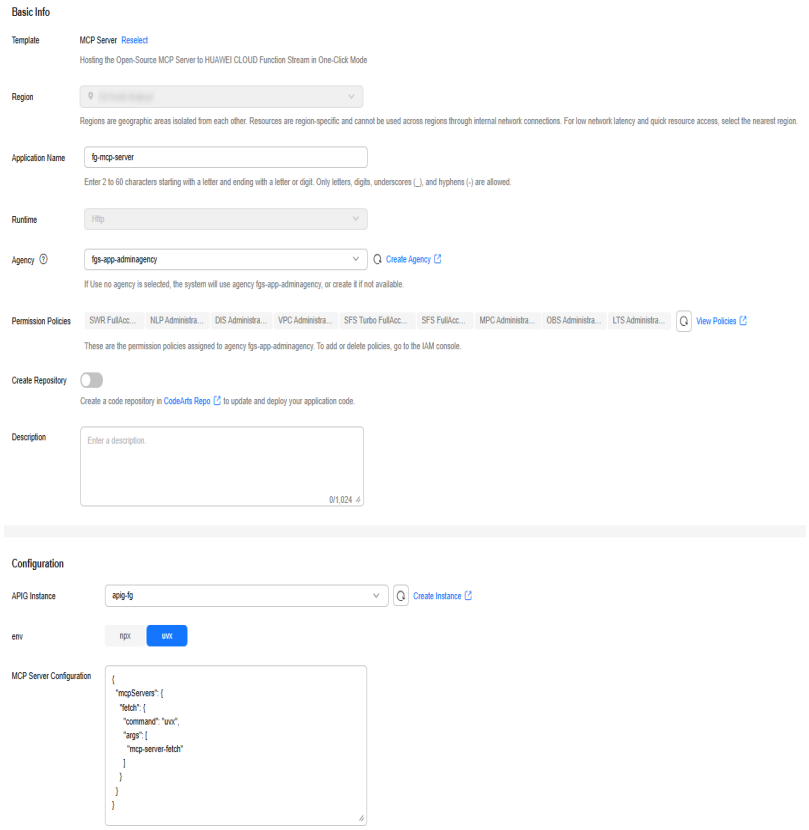



**Table 4-16** Parameters for creating an MCP server application

| Parameter | Example Value | Description |
| --- | --- | --- |
| Template | MCP Server | The selected function template is displayed by default. To change it, click **Reselect**. |
| Region | **CN North-Beijing4** | Select a region to create the application. This application can be created in the **CN North-Beijing4** region.<br>Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. Select a region near you to ensure the lowest latency possible. |
| Application Name | fg-mcp-server | Enter a custom application name,<br>which contains 2 to 60 characters, including letters, digits, underscores (_), and hyphens (-). It must start with a letter and end with a letter or digit. |

| Parameter | Example Value | Description |
| --- | --- | --- |
| Runtime | HTTP | The default runtime of the template is displayed and cannot be changed. |
| Agency | fgs-app-adminagency | Select the agency of the function to access other cloud services.<br>If the default agency **fgs-app-adminagency** has not been created, select **Use no agency**, complete other configurations, and click **Create Now**. The system prompts you to create an agency named **fgs-app-adminagency** to ensure that the application can be created. |
| Create Repository | Disabled | When enabled, you can create a code repository in CodeArts Repo to update and deploy your application code. |
| (Optional) **Description** | - | Enter a description of the application.<br>It can contain up to 1,024 characters. |
| APIG Instance | apig-fg | Select the APIG gateway created in **Step 1: Creating a Dedicated Gateway**. |
| env | uvx | You can select either of the following running environments based on your requirements:<br>• **npx**: Temporarily invokes npm packages to execute commands. It is Node.js-based and no global dependency is required.<br>• **uvx**: temporarily installs and runs the command line tool provided by the Python package in an isolated environment. |

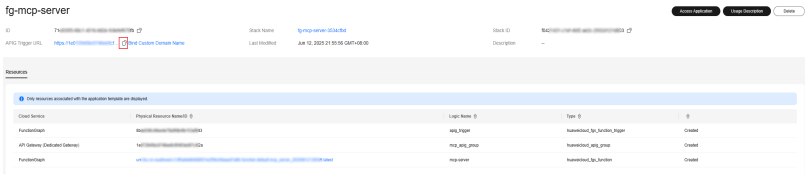
  

| Parameter | Example Value | Description |
| --- | --- | --- |
| MCP Service Configuration | {<br>"mcpServers": {<br>"fetch": {<br>"command": "uvx",<br>"args": [<br>"mcp-server-fetch"<br>]<br>}<br>}<br>} | Enter the MCP service configuration in JSON format. You can customize the configuration as needed.<br>The JSON configuration file defines how to obtain data from the MCP server. The following describes the parameters in the JSON configuration file in this example:<br>• **mcpServers**: main object of the configuration file, indicating the configuration related to the MCP server.<br>• **fetch**: configuration related to the fetch operation, which defines how to obtain data from the MCP server.<br>• **command**: name of the command or tool used to perform the fetch operation. In this example, the **uvx** environment is used.<br>• **args**: parameter list provided for the **command** command, which is used to notify the specific task to be executed. |

5. After the application is created, copy the **APIG Trigger URL**.

   By default, a temporary domain name is provided for 30 days in the test environment. In the actual production environment, prepare a custom domain name.

**Figure 4-51** MCP server application



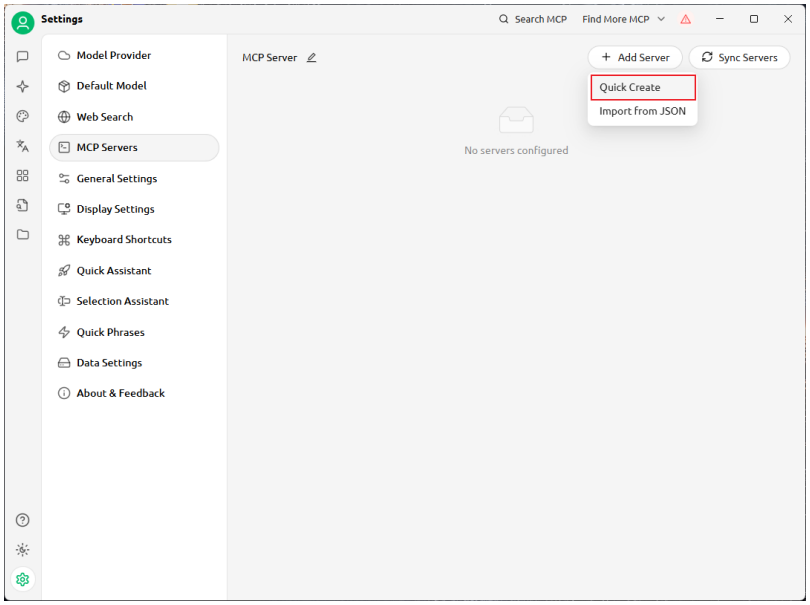

## Step 3: Configuring a Client to Start an AI Dialog

In this example, CherryStudio is used as the client for AI dialog. Install the CherryStudio client applicable to your device and ensure that a proper model is configured for CherryStudio to perform basic dialogs.

1. Open the CherryStudio client, go to the settings page, click **MCP Servers**, and choose **Add Server > Quick Create**.

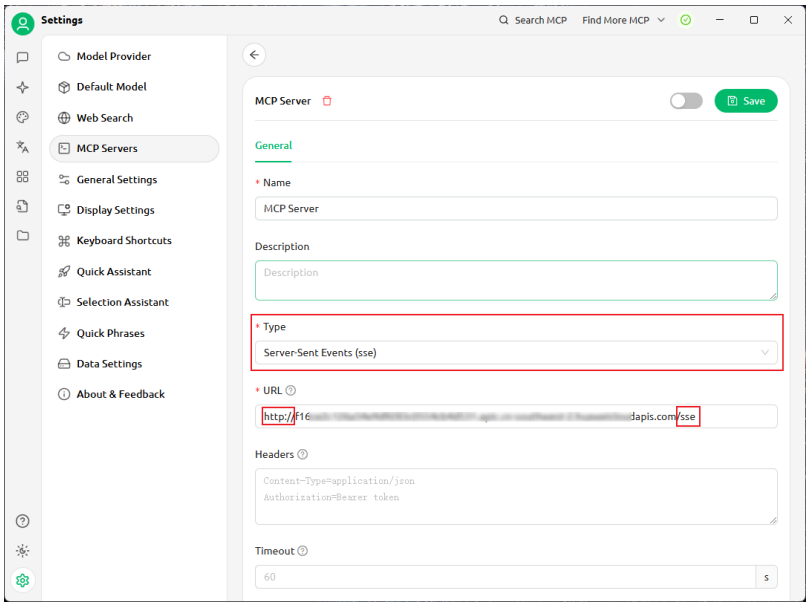
  

**Figure 4-52** Adding a server




2. Set **Type** to **Server-Sent Events (sse)**, enter the URL copied in **5** in the **URL** text box, change *https* to *http*, and add *sse* to the end of the URL. The configuration is complete, as shown in **Figure 4-53**. Click **Save**.

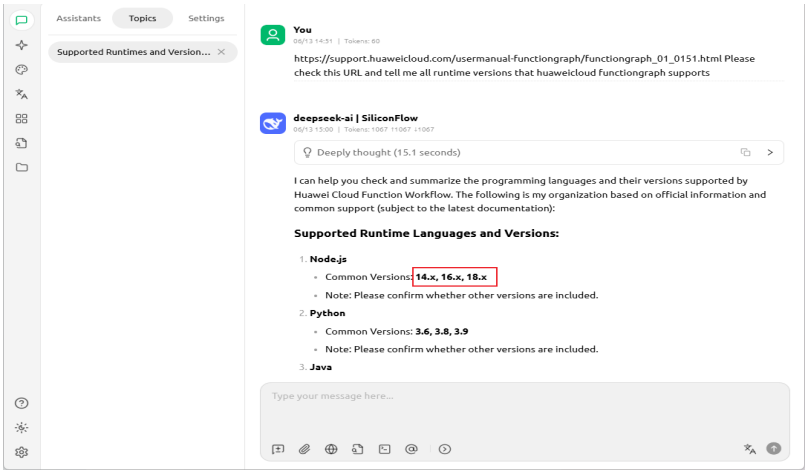
**Figure 4-53** Configuring the MCP Server




3. After the configuration is complete, you can access the assistant page to start AI dialogs. As shown in **Figure 4-54**, the MCP server is not enabled for model dialog, and the model does not provide a normal answer.

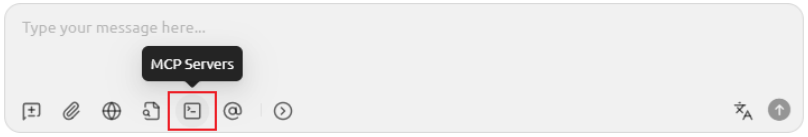
  

**Figure 4-54** Dialog with MCP server disabled



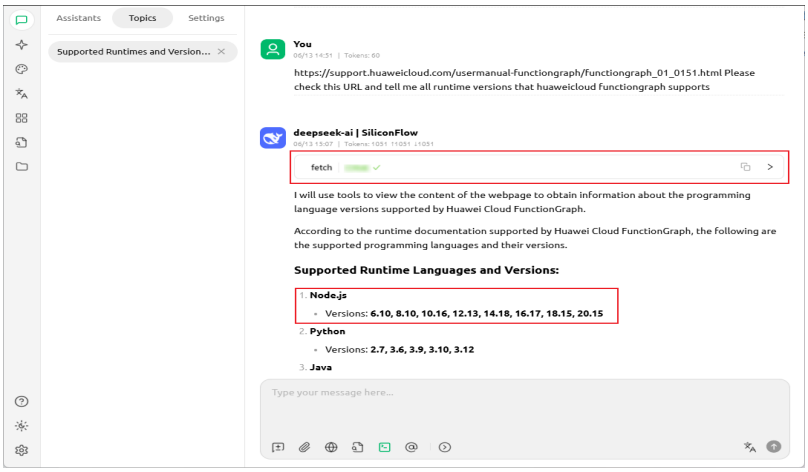

4. Click the MCP server button in the chat box and select the MCP server configured in 2.

**Figure 4-55** Configuring the MCP server




5. After the MCP server is enabled, the model invokes the fetch tool to read the official document link and answers the question correctly, as shown in **Figure 4-56**.

**Figure 4-56** Dialog with MCP server enabled

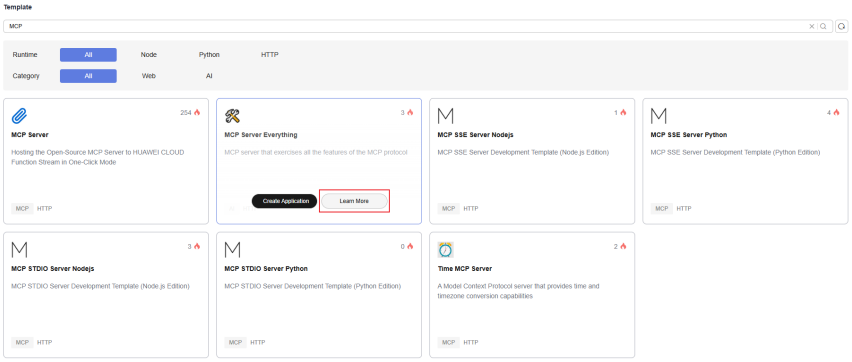



## More MCP Templates

FunctionGraph provides multiple popular MCP application templates to help you quickly develop MCP services. You can click **Learn More** of a template to view the usage description.

**Figure 4-57** MCP-related templates




## Disclaimer

- This practice uses open-source projects (**supergateway**, **mcp-proxy**, and **image build project**). Huawei Cloud is not responsible for these projects. For open-source issues, contact their official communities. Huawei Cloud only provides computing support.
- This practice is for reference only. If you want to use it in a production environment, test it thoroughly and evaluate the costs. For any questions, **submit a service ticket**.

# 5 Function Building Practices

## 5.1 Building an HTTP Function Using an Existing Spring Boot Project

### Introduction

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

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

### Procedure

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

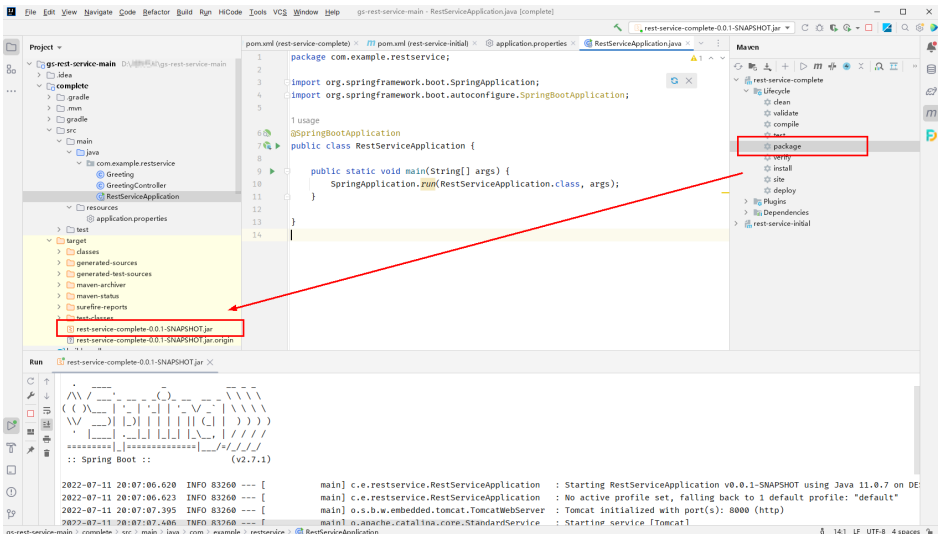
In this example, a Maven project created using IntelliJ IDEA is used.

**Building a Code Package**

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

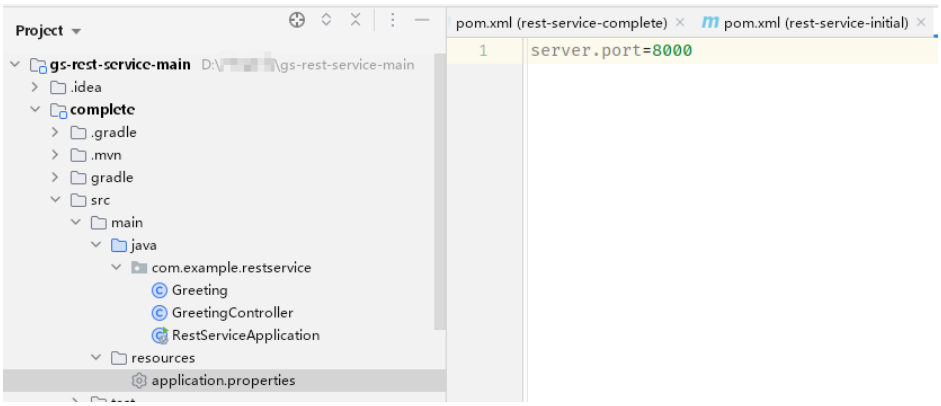
  

**Figure 5-1** Generating a JAR file



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

**Figure 5-2** Configuring port 8000




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

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

NOTE

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

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

### Creating an HTTP Function and Uploading Code

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

### Verifying the Result

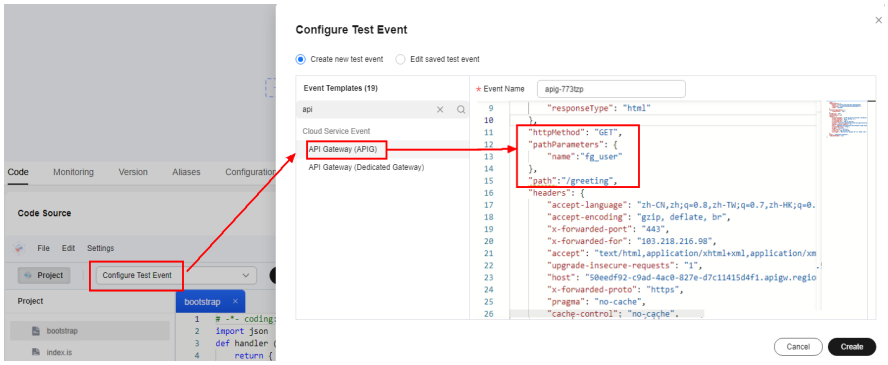
- Using a test event

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

b. On the **Configure Test Event** page, select the event template, and modify the **path** and **pathParameters** parameters in the template to construct a simple GET request.

**Figure 5-3** Configuring a test event

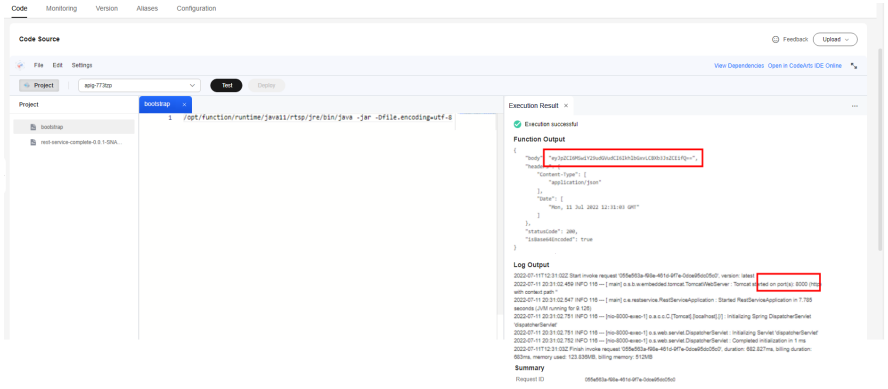



c. Click **Create**.

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 5-4** Viewing the returned result

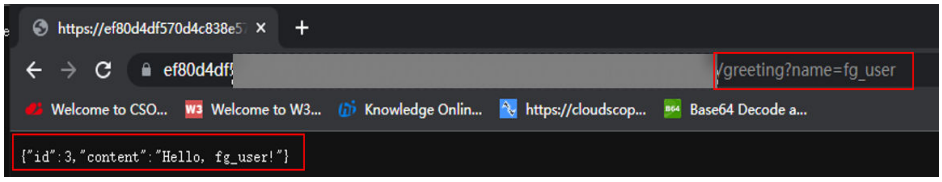



- Using an APIG trigger

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

**Figure 5-5** 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 5-6** Modifying the mapping address

- 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?

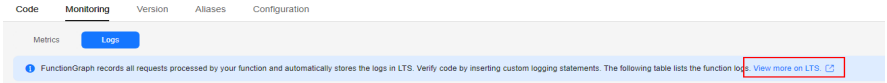
   Function instances that have not received any requests during a specific period of time will be deleted together with their local logs. You will be unable to view the function logs during function running. Therefore, in addition to writing logs to your local host, output logs to the console by setting 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 5-7** 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>
```

# 5.2 Building an HTTP Function Using 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**

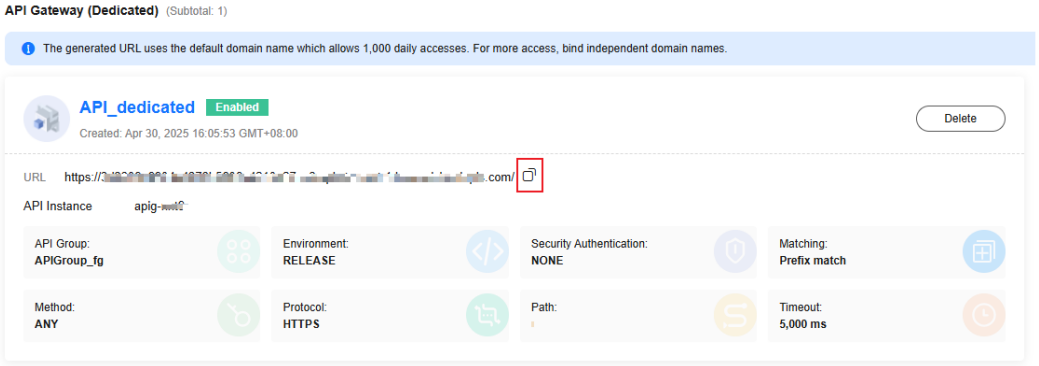
Create an HTTP function and upload the **xxx.zip** package. For details, see **Creating an HTTP Function**.

**Creating an APIG (Dedicated) trigger**

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

**Figure 5-8** 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 5-9** Request result


# 5.3 Using FunctionGraph to Access RDS for MySQL

## 5.3.1 Introduction

### Scenario

In FunctionGraph, different function instances do not share states. Databases can persistently store structured data, which achieves state sharing. FunctionGraph can be used to access cloud databases to query and insert data.

This section describes how to access RDS for MySQL from FunctionGraph and query data, and provides **sample code** for testing. The sample uses a **database connection pool and retry mechanism** to improve performance and reliability, demonstrating a secure and efficient way to work with RDS for MySQL in FunctionGraph.

### Resource and Cost Planning

**Table 1** describes the resources required for the practice of accessing RDS for MySQL from FunctionGraph.

**Table 5-1** Resource and cost planning

| Resource | Description | Billing |
| --- | --- | --- |
| FunctionGraph | • Function type: **Event Function**<br>• Region: **CN East-Shanghai1**<br>• Quantity: 1 | • Billing mode: Pay-per-use<br>• The first 1 million invocations are free of charge in a month. For details about the billing items, see **Pay-per-Use Billing**. |
| RDS | • Region: **CN East-Shanghai1**<br>• **Engine Options**: **MySQL**<br>• Quantity: 1 | • Billing mode: Pay-per-use<br>• Select specifications based on service requirements. For details about billing items and standards, see **RDS for MySQL Pay-per-Use Billing**. |
| VPC | • Region: **CN East-Shanghai1**<br>• Number of subnets: 1<br>• Number of security groups: 1<br>• Quantity: 1 | • VPC: free<br>• Subnet: free<br>• Security group: free |

## Procedure

This following describes the procedure for using a FunctionGraph function to access RDS for MySQL. For details, see **Procedure**.

**Table 5-2** Procedure for accessing RDS for MySQL from a function

| Step | Description |
| --- | --- |
| **Prerequisites** | Before starting this practice, ensure you have a VPC network environment, an RDS for MySQL instance with its database and tables, and a function agency with **VPC Administrator** permission already created. |
| **Step 1: Creating Function Dependencies** | In this practice, Python sample code is used to implement database connection and access. The code depends on the **pymysql** and **DBUtils** packages, which must be uploaded to the FunctionGraph console for subsequent invocations. |
| **Step 2: Creating a Function** | On the FunctionGraph console, create a function for accessing RDS for MySQL. |
| **Step 3: Configuring the Function** | On the function details page, configure the function code, dependencies, and function settings. |

| Step | Description |
| --- | --- |
| Step 4: Testing the Function | Test whether the function can access records in the database table of the RDS for MySQL instance. |

## 5.3.2 Procedure

### Prerequisites

- You have **created a VPC and a default subnet**.
- You have **purchased an RDS for MySQL DB instance**, **created a database**, and ensured that the database contains available tables. The RDS instance must be in the same region as the function. This practice uses **CN East-Shanghai1** as an example.
- You have **created an agency** with the **VPC Administrator** permission.

### Step 1: Creating Function Dependencies

**Table 5-3** Downloading dependency packages

| Dependency Package | Description | Link |
| --- | --- | --- |
| pymysql | MySQL database connector compiled using Python, which enables Python programs to communicate with MySQL databases. | **pymysql_py36-1.zip** (**SHA-256 verification file**) |
| DBUtils | Database connection pool tool package, which can manage and reuse database connections. | **dbutils_py36-1.zip** (**SHA-256 verification file**) |

1. Log in to the **FunctionGraph console** and select **CN East-Shanghai1** region.
2. Log in to the FunctionGraph console, and choose **Functions** > **Dependencies** in the navigation pane.
3. Click **Create Dependency**. On the displayed page, create the **pymysql_py36** and **dbutils_py36** dependencies by referring to **Table 5-4**.

   The **pymysql** dependency is used as an example, as shown in **Figure 5-10**. The parameters for the **dbutils** dependency are the same.

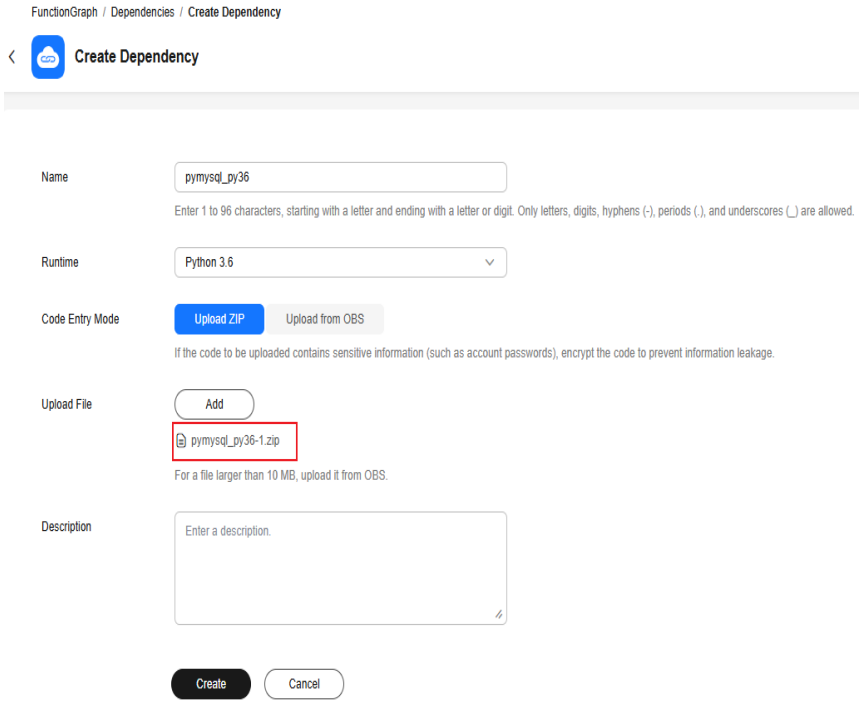
  

**Table 5-4** Parameters for creating a dependency

| Parameter | Value | Description |
| --- | --- | --- |
| Name | pymysql_py36 | Dependency name.<br>The value can contain a maximum of 96 characters, including letters, digits, underscores (_), periods (.), and hyphens (-). Start with a letter and end with a letter or digit. |
| Runtime | Python 3.6 | Runtime language of the dependency. |
| Code Entry Mode | Upload ZIP | You can upload a ZIP file or upload a ZIP file from OBS.<br>• **Upload ZIP**: Click **Add** to upload a ZIP file. **The maximum file size is 10 MB.**<br>• **Upload from OBS**: Specify an OBS link URL. For details about how to upload an object to OBS, see **Uploading an Object**. For details about how to obtain the OBS link URL, see **Accessing an Object Using a URL**. |
| Upload File | Click **Add** and upload the ZIP package downloaded in **Table 5-3**. | This parameter is displayed after you select **Upload ZIP**. |
| Description | - | Enter a description of the dependency. |

**Figure 5-10** Creating a dependency




## Step 2: Creating a Function

1. Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**, and click **Create Function**.
2. Set the basic information about the function by referring to **Table 5-5**, as shown in **Figure 5-11**.

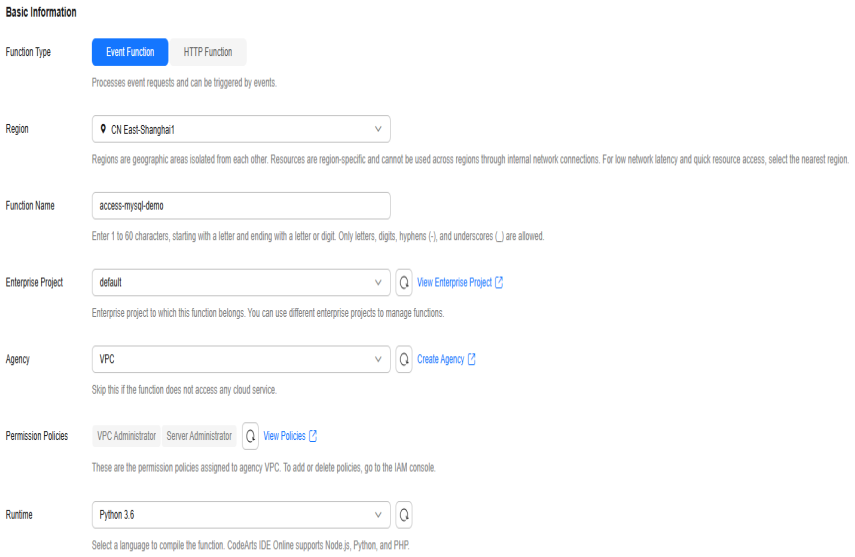
**Table 5-5** Basic information for creating a function

| Parameter | Example Value | Description |
| --- | --- | --- |
| Function Type | Event Function | An event function is triggered by a specific event, which is usually a request event in JSON format. |
| Region | CN East-Shanghai1 | Select the region where the function is located. This practice uses the **CN East-Shanghai1** region as an example. |
| Function Name | access-mysql-demo | Enter a function name. The naming rules are as follows:<br>• Consists of 1 to 60 characters, and can contain letters, digits, hyphens (-), and underscores (_).<br>• Starts with a letter and end with a letter or digit. |

| Parameter | Example Value | Description |
| --- | --- | --- |
| Enterprise Project | default | Select the enterprise project to which the function belongs. Enterprise projects let you manage cloud resources and users by project.<br>The default value is **default**. You can select the created enterprise project.<br>**If the Enterprise Management service is not enabled, this parameter is unavailable. For details, see Enabling the Enterprise Project Function.** |
| Agency | VPC | Select an agency for the function. An agency is used to authorize FunctionGraph to access other cloud services.<br>In this practice, the function needs to access resources in a VPC. Therefore, you need to select an agency with the **VPC Administrator** permission. |
| Runtime | Python 3.6 | Select a runtime to compile the function. This practice uses Python 3.6 as an example. |

**Figure 5-11** Basic information for creating a function



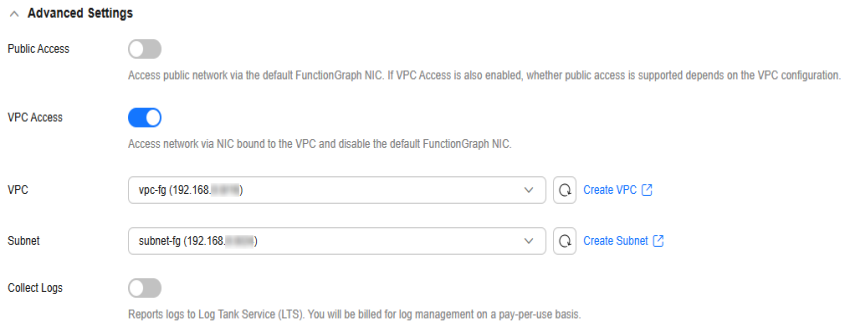

3. Configure the advanced settings by referring to **Table 5-6**, and click **Create Now**, as shown in **Figure 5-12**.

**Table 5-6** Advanced settings for creating a function

| Parameter | Example Value | Description |
| --- | --- | --- |
| Public Access | Disabled | If this feature is enabled, functions can access services on the public network through the default NIC. The public network access bandwidth is shared among users and applies only to test scenarios. |
| VPC Access | Enabled<br>• VPC: **vpc-fg (192.168.x.x/x)**<br>• Subnet: **subnet-fg (192.168.x.x/x)** | After this feature is enabled, you can select the VPC and subnet that the function needs to access. **Select the same VPC as the created RDS instance.**<br>If this feature is enabled, functions will use the NIC bound to the configured VPC for network access, and the default NIC of FunctionGraph will be disabled. That is, the **Public Access** parameter does not take effect. |
| Collect Logs | Disabled | After it is enabled, logs generated during function execution will be reported to LTS.<br>LTS will be billed on a pay-per-use basis. For details, see **LTS Pricing Details**. |

**Figure 5-12** Advanced settings for creating a function




## Step 3: Configuring the Function

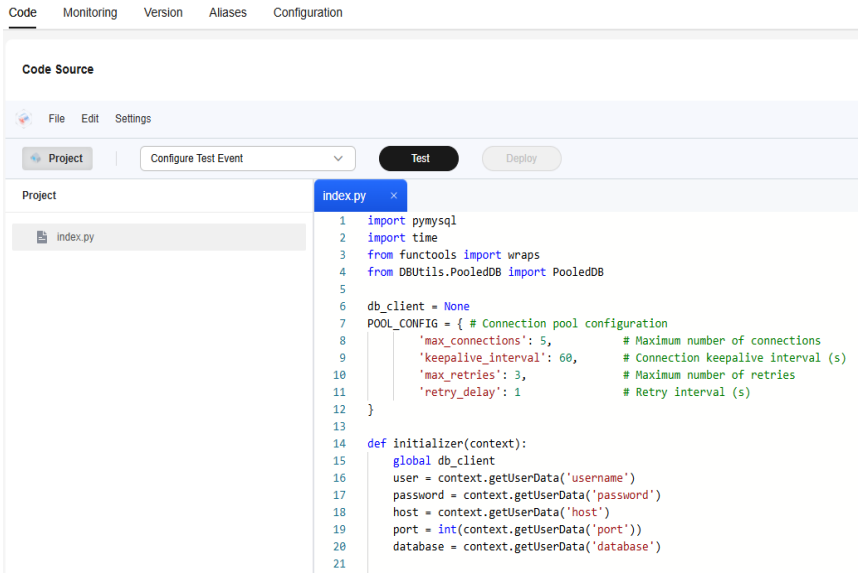
1. On the function details page, copy **Sample Code for Accessing RDS for MySQL**, as shown in **Figure 5-13**, paste it to the inline editor to overwrite the code in the **index.py** file, and click **Deploy**.

**NOTICE**

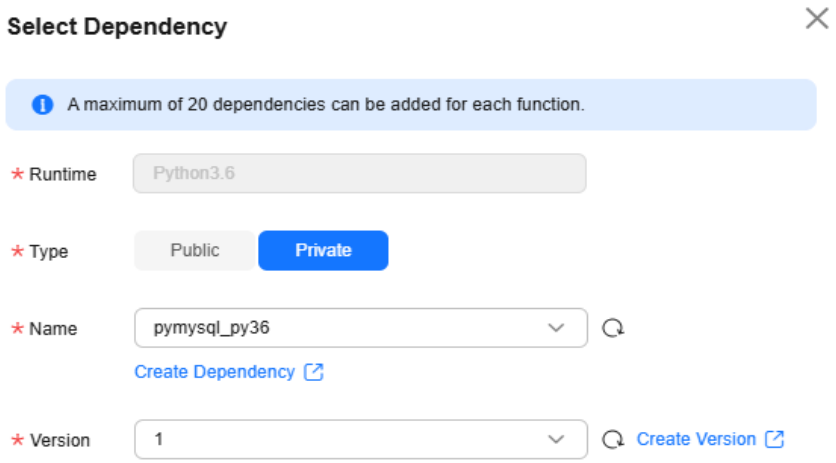
The sample code in this practice queries the first 10 records from the user table in the RDS for MySQL database. Modify the code according to the actual table name in your database.

**Figure 5-13** Deploying code




2. On the **Code** tab page, scroll to the **Dependencies** area at the bottom of the page and click **Add**.
3. In the **Select Dependency** dialog box, set **Type** to **Private**, and add the **pymysql_py36** and **dbutils_py36** dependencies created in **Step 1: Creating Function Dependencies**, as shown in **Figure 5-14**.

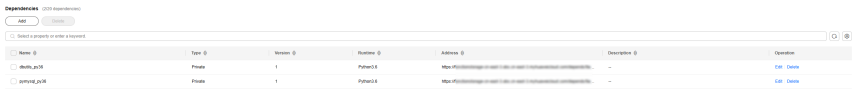
**Figure 5-14** Selecting dependencies




4. After the dependencies are added, the page shown in **Figure 5-15** is displayed.

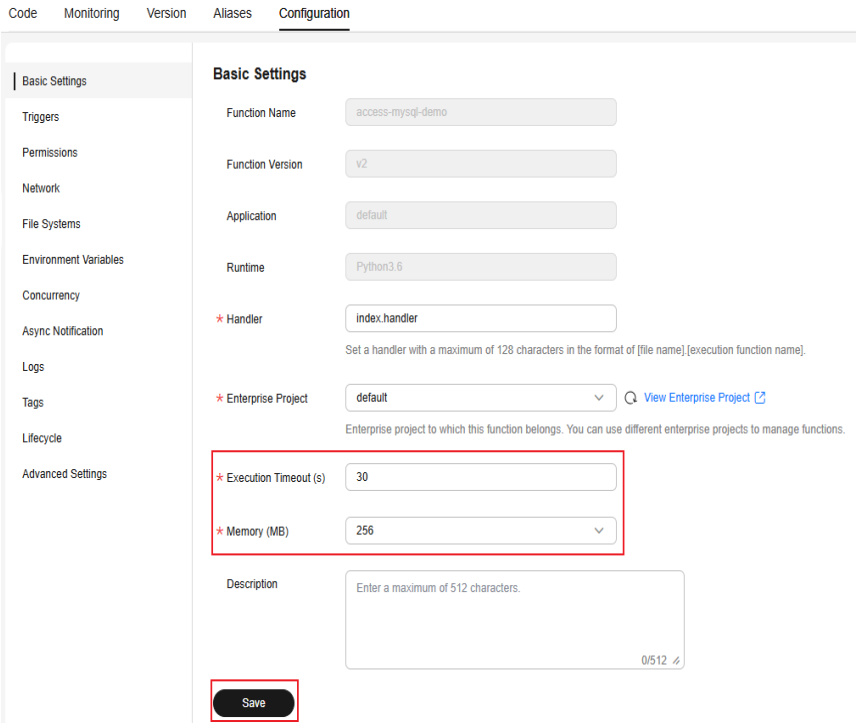
  

**Figure 5-15** Dependencies



5. Choose **Configuration** > **Basic Settings**, set **Execution Timeout (s)** to **30** and **Memory (MB)** to **256**, and click **Save**.

**Figure 5-16** Configuring basic settings



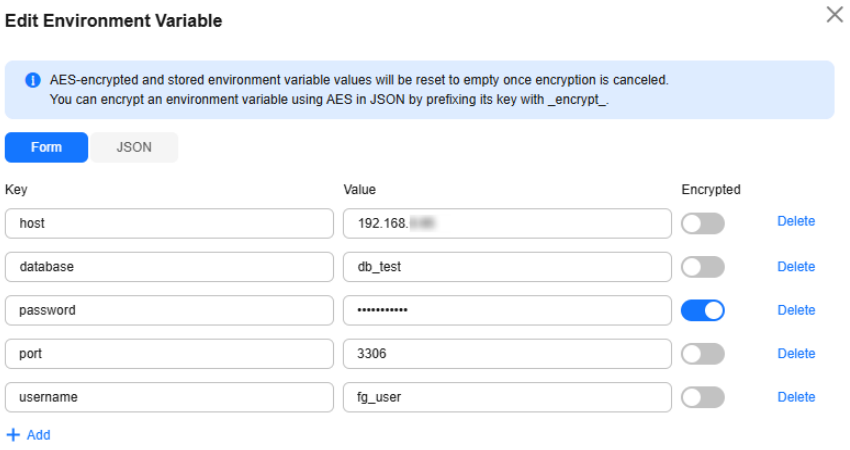

6. Choose **Configuration** > **Environment Variables**, and click **Edit Environment Variable**. In the dialog box that is displayed, click **Add**. Add environment variables by referring to **Table 5-7**. After the environment variables are added, the page shown in **Figure 5-17** is displayed. Click **OK**.

**Table 5-7** Environment variables

| Key | Example Value | Description |
| --- | --- | --- |
| host | 192.168.x.x | Access address of the database instance.<br>• For RDS databases in the same VPC as the function, set this environment variable to the private IP address of the database.<br>• For RDS databases in different VPCs or regions, set this environment variable to the public IP address of the database.<br>On the RDS console, go to the details page of the target RDS instance, choose **Connectivity & Security** in the navigation pane, and obtain the private or public IP address of the database. |
| database | db_test | Name of the database created in the RDS instance. |
| password | ****** | Database password set in the RDS instance.<br>The password is sensitive information. You are advised to enable encryption. |
| port | 3306 | Database port set in the RDS instance. |
| username | fg_user | Name of the account created in the RDS instance. |

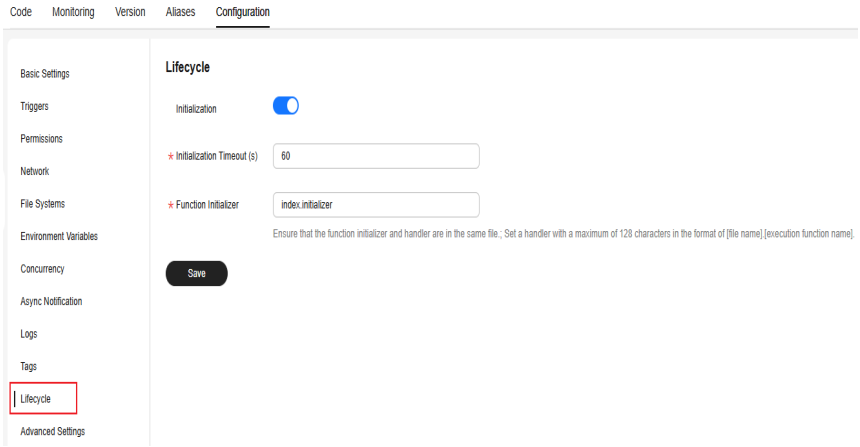
**Figure 5-17** Editing environment variables




7. Choose **Configuration** > **Lifecycle**, enable **Initialization**, set **Initialization Timeout (s)** to **60** and **Function Initializer** to **index.initializer**, as shown in **Figure 5-18**, and click **Save**.

**Figure 5-18** Configuring initialization

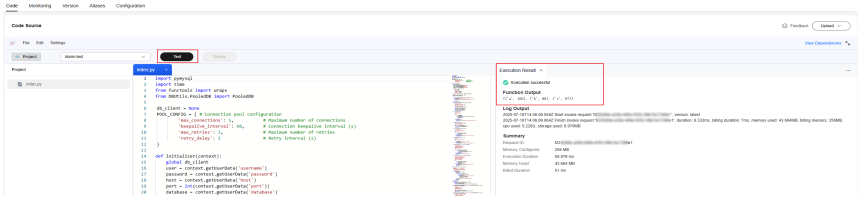



## Step 4: Testing the Function

1. Return to the **Code** tab page and click **Test**. In the displayed dialog box, select **Blank Template**, and click **Create**.
2. Click **Test** to view the function execution result, as shown in **Figure 5-19**.

   According to the sample code, the function returns the first 10 records in the user table of the RDS for MySQL DB instance.

**Figure 5-19** Testing the function



# 5.3.3 Sample Code for Accessing RDS for MySQL

## Sample Code

This sample code queries the first 10 records from the **user** table in the RDS for MySQL instance database. The code can efficiently and reliably perform database operations by using the database connection pool and retry mechanism.

The following shows the complete sample code. For details about the code of the connection pool and retry mechanism, see **Sample Code Interpretation**.

```
import pymysql
import time
from functools import wraps
from DBUtils.PooledDB import PooledDB

db_client = None
POOL_CONFIG = { # Connection pool configuration
        'max_connections': 5,           # Maximum number of connections
        'keepalive_interval': 60,       # Connection keepalive interval (s)
```

```
    'max_retries': 3,              # Maximum number of retries
    'retry_delay': 1               # Retry interval (s)
}

def initializer(context):
    global db_client
    user = context.getUserData('username')
    password = context.getUserData('password')
    host = context.getUserData('host')
    port = int(context.getUserData('port'))
    database = context.getUserData('database')

    dbConfig = { # MySQL database configuration
        'host': host,
        'port': port,
        'user': user,
        'password': password,
        'database': database,
        'charset': 'utf8',
    }
    db_client = Database(context, POOL_CONFIG, dbConfig)

def handler(event, context): # Handler
    logger = context.getLogger()

    try:
        result= db_client.query("SELECT * FROM user LIMIT 10")
    except Exception as e:
        logger.info("query database error:%s" % e)
        return {"code": 400, "errorMsg": "internal error %s" % e}

    return result

class MySQLConnectionPool:
    def __init__(self, context, pool_config, db_config):
        """
        Initialize the database connection pool
        :param db_config: database configuration
        :param pool_config: connection pool configuration
        """
        self.context = context
        self.logger = context.getLogger();

        self.db_config = db_config
        self.pool_config = pool_config
        self.pool = self._create_pool()
        self.last_keepalive_time = 0

    def _create_pool(self):
        """
        Create a connection pool
        :return: connection pool object
        """
        try:
            pool = PooledDB(
                creator=pymysql,
                maxconnections=self.pool_config['max_connections'],
                mincached=1,
                **self.db_config
            )
            return pool
        except Exception as e:
            self.logger.error(f"Failed to create connection pool: {e}")
            raise

    def _get_connection(self):
        """
        Obtain a connection from the connection pool and ensure that the connection is valid
        :return: database connection object
```

```
        """
        conn = self.pool.connection()
        if not self._is_connection_alive(conn):
            conn = self.pool.connection()
        return conn

    def _is_connection_alive(self, conn):
        """
        Check whether the connection is alive
        :param conn: database connection object
        :return: bool
        """
        try:
            with conn.cursor() as cursor:
                cursor.execute("SELECT 1")
                return True
        except Exception as e:
            self.logger.warning(f"Connection is not alive: {e}")
            return False

    def _close_connection(self, conn):
        """
        Closes the connection
        :param conn: database connection object
        """
        try:
            conn.close()
            self.logger.info("Connection closed")
        except Exception as e:
            self.logger.error(f"Failed to close connection: {e}")

    def _execute_query(self, conn, sql, params=None):
        """
        Query the database
        :param conn: database connection object
        :param sql: SQL statement
        :param params: SQL parameter
        :return: query result
        """
        try:
            with conn.cursor() as cursor:
                cursor.execute(sql, params)
                if sql.strip().lower().startswith('select'):
                    return cursor.fetchall()
                return None
        except Exception as e:
            self.logger.error(f"Query failed: {e}")
            raise

    def _execute_write(self, conn, sql, params=None):
        """
        Perform write operations (insert, update, and delete)
        :param conn: database connection object
        :param sql: SQL statement
        :param params: SQL parameter
        :return: number of affected rows
        """
        try:
            with conn.cursor() as cursor:
                cursor.execute(sql, params)
                conn.commit()
                return cursor.rowcount
        except Exception as e:
            self.logger.error(f"Write operation failed: {e}")
            conn.rollback()
            raise

def retry(max_retries=3, retry_delay=1):
    """
```

```
    Retry decorator
    :param max_retries: Maximum retries
    :param retry_delay: Retry interval (s)
    """
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            retries = 0
            while retries < max_retries:
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    print(f"Attempt {retries + 1} failed: {e}")
                    if retries < max_retries - 1:
                        time.sleep(retry_delay)
                    retries += 1
            print(f"Failed after {max_retries} attempts")
            raise
        return wrapper
    return decorator

class Database:
    def __init__(self, context, pool_config, db_config):
        self.pool_config = pool_config
        self.db_config = db_config
        self.pool = MySQLConnectionPool(context, pool_config, db_config)

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def query(self, sql, params=None):
        """
        Perform the query operation
        :param sql: SQL statement
        :param params: SQL parameter
        :return: query result
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_query(conn, sql, params)
        return result

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def execute(self, sql, params=None):
        """
        Perform write operations (insert, update, and delete)
        :param sql: SQL statement
        :param params: SQL parameter
        :return: number of affected rows
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_write(conn, sql, params)
        return result
```

# 5.3.4 Sample Code Interpretation

## Connection Pool

In the sample code, the MySQL connection pool (DBUtils.PooledDB) is created, the built-in connection keepalive mechanism is provided, and the maximum number of connections (max_connections) and connection keepalive interval (keepalive_interval) are configured. The code snippet is as follows:

```
POOL_CONFIG = { # Connection pool configuration
    'max_connections': 5,          # Maximum number of connections
    'keepalive_interval': 60,      # Connection keepalive interval (s)
    'max_retries': 3,              # Maximum number of retries
    'retry_delay': 1               # Retry interval (s)
}
```

```
class MySQLConnectionPool:
    def __init__(self, context, pool_config, db_config):
        """
        Initialize the database connection pool
        :param db_config: database configuration
        :param pool_config: connection pool configuration
        """
        self.context = context
        self.logger = context.getLogger();

        self.db_config = db_config
        self.pool_config = pool_config
        self.pool = self._create_pool()
        self.last_keepalive_time = 0

    def _create_pool(self):
        """
        Create a connection pool
        :return: connection pool object
        """
        try:
            pool = PooledDB(
                creator=pymysql,
                maxconnections=self.pool_config['max_connections'],
                mincached=1,
                **self.db_config
            )
            return pool
        except Exception as e:
            self.logger.error(f"Failed to create connection pool: {e}")
            raise

    def _get_connection(self):
        """
        Obtain a connection from the connection pool and ensure that the connection is valid
        :return: database connection object
        """
        conn = self.pool.connection()
        if not self._is_connection_alive(conn):
            conn = self.pool.connection()
        return conn

    def _is_connection_alive(self, conn):
        """
        Check whether the connection is alive
        :param conn: database connection object
        :return: bool
        """
        try:
            with conn.cursor() as cursor:
                cursor.execute("SELECT 1")
                return True
        except Exception as e:
            self.logger.warning(f"Connection is not alive: {e}")
            return False

    def _close_connection(self, conn):
        """
        Closes the connection
        :param conn: database connection object
        """
        try:
            conn.close()
            self.logger.info("Connection closed")
        except Exception as e:
            self.logger.error(f"Failed to close connection: {e}")

    def _execute_query(self, conn, sql, params=None):
        """
```

```
    Query the database
    :param conn: database connection object
    :param sql: SQL statement
    :param params: SQL parameter
    :return: query result
    """
    try:
        with conn.cursor() as cursor:
            cursor.execute(sql, params)
            if sql.strip().lower().startswith('select'):
                return cursor.fetchall()
            return None
    except Exception as e:
        self.logger.error(f"Query failed: {e}")
        raise

def _execute_write(self, conn, sql, params=None):
    """
    Perform write operations (insert, update, and delete)
    :param conn: database connection object
    :param sql: SQL statement
    :param params: SQL parameter
    :return: number of affected rows
    """
    try:
        with conn.cursor() as cursor:
            cursor.execute(sql, params)
            conn.commit()
            return cursor.rowcount
    except Exception as e:
        self.logger.error(f"Write operation failed: {e}")
        conn.rollback()
        raise
```

Reusing created connections with the MySQL connection pool can improve program performance. The maximum number of connections ensures that connection resources are used within a controllable range and thread security is ensured.

**Related concepts:**

**Table 5-8** Concepts related to connections

| Concept | Description |
| --- | --- |
| Range of the maximum number of connections | The recommended maximum MySQL connections fall within:<br>• **Lower limit of the maximum number of connections** = (**Concurrency of a single function instance**) x (Concurrent MySQL accesses per function execution)<br>• **Upper limit of the maximum number of connections** = (Maximum number of MySQL instance connections)/ (Maximum number of function instances)<br>For a function with a maximum of 5 concurrent requests per instance, 2 concurrent MySQL accesses per function execution, a maximum of 400 instances by default, and a maximum of 30,000 MySQL connections, the calculation is as follows:<br>• Lower limit of the maximum number of connections = 5 x 2 = 10<br>• Upper limit of the maximum number of connections = 30000/400 = 75<br>Based on the preceding result, you are advised to set the maximum number of connections to 50. |
| Connection Keepalive Interval | Do not set the connection keepalive interval longer than the **function execution timeout** to avoid disconnection issues. |

## Retry

Build an automatic retry mechanism using decorators to ensure that MySQL operations are retried for a specified number of times after a failure. This significantly reduces the impact of temporary faults. For example, if the service is temporarily unavailable or the invoking times out due to instantaneous network jitter or disk jitter, the mechanism can improve the MySQL operation success rate.

The code snippet is as follows:

```
POOL_CONFIG = { # Connection pool configuration
        'max_connections': 5,           # Maximum number of connections
        'keepalive_interval': 60,       # Connection keepalive interval (s)
        'max_retries': 3,               # Maximum number of retries
        'retry_delay': 1                # Retry interval (s)
}

class Database:
    def __init__(self, context, pool_config, db_config):
        self.pool_config = pool_config
        self.db_config = db_config
        self.pool = MySQLConnectionPool(context, pool_config, db_config)

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def query(self, sql, params=None):
        """
        Perform the query operation
        :param sql: SQL statement
        :param params: SQL parameter
        :return: query result
```

```
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_query(conn, sql, params)
        return result

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def execute(self, sql, params=None):
        """
        Perform write operations (insert, update, and delete)
        :param sql: SQL statement
        :param params: SQL parameter
        :return: number of affected rows
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_write(conn, sql, params)
        return result

def retry(max_retries=3, retry_delay=1):
    """
    Retry decorator
    :param max_retries: Maximum retries
    :param retry_delay: Retry interval (s)
    """
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            retries = 0
            while retries < max_retries:
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    print(f"Attempt {retries + 1} failed: {e}")
                    if retries < max_retries - 1:
                        time.sleep(retry_delay)
                    retries += 1
            print(f"Failed after {max_retries} attempts")
            raise
        return wrapper
    return decorator
```