# FunctionGraph

# User Guide

**Issue** 01

**Date** 2025-12-29

**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



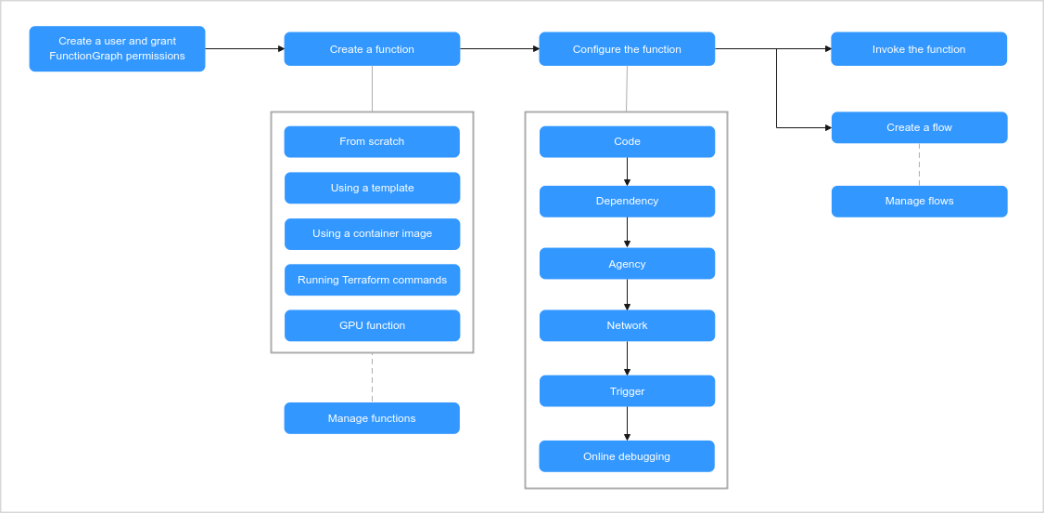
  

# 1 Process of Using FunctionGraph

FunctionGraph allows you to run your code without provisioning or managing servers, while ensuring high availability and scalability. All you need to do is upload your code and set execution conditions, and FunctionGraph will take care of the rest. You pay only for what you use and you are not charged when your code is not running.

To quickly create a function using FunctionGraph, do as follows:

**Figure 1-1** FunctionGraph process overview



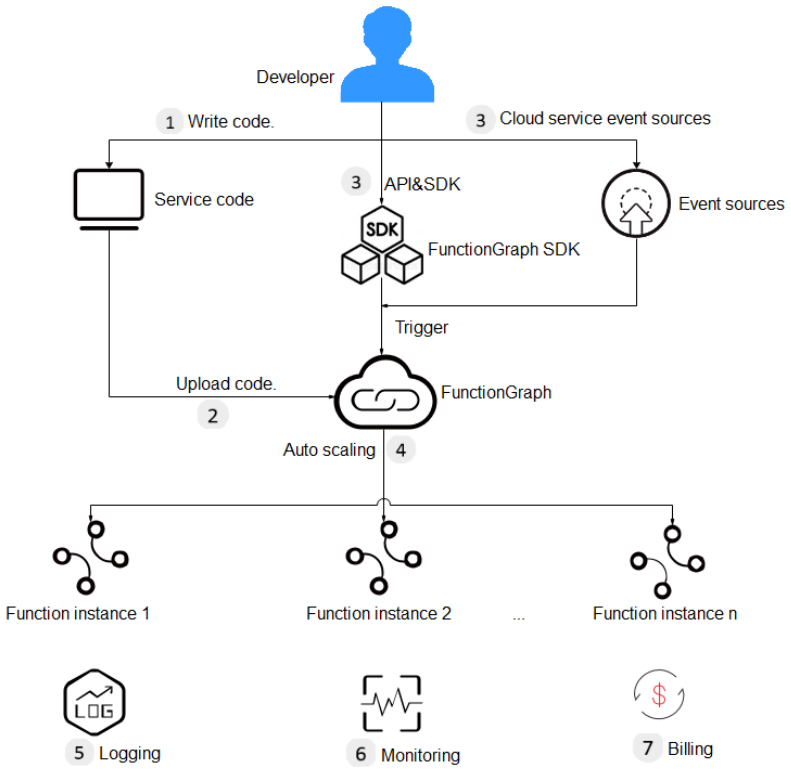

1. **Create a user and grant FunctionGraph permissions**: Ensure that the login user has the FunctionGraph Administrator permission.
2. **Create a function**: Create a function from scratch, using a template, by running Terraform commands, or with an image.
3. **Configure the function**: Configure function code, dependencies, public network or VPC access, agency permissions, triggers, and online debugging.
4. **Invoke the function**: Functions can be triggered through RESTful APIs or cloud service event sources.
5. **Flow**: Create and manage function flows.

## Process

Figure 1-2 describes how to use FunctionGraph to implement service requirements.

**Figure 1-2** Use of FunctionGraph




The following shows the details:

1. Write code.

   Write code in Node.js, Python, Java, C#, PHP, Go, Cangjie, or custom runtimes. For details, see **FunctionGraph Developer Guide**.

2. Upload code.

   Upload the service code. You can edit code online or upload code files. For details, see **Configuring Function Code** and **Configuring Dependencies**.

3. Trigger functions by API calls or cloud service events.

   Functions are triggered by API calls or cloud service events. For details, see **Configuring Triggers**.

4. Implement auto scaling.

   FunctionGraph scales automatically during function execution based on the number of requests without the need for configurations. For details about concurrency restrictions, see **Notes and Constraints**.

5. View logs.

   View run logs of function as FunctionGraph is interconnected with Log Tank Service (LTS). For details, see **Configuring and Viewing Function Invocation Logs**.

6. View monitoring information.

   View graphical monitoring information. FunctionGraph is interconnected with Cloud Eye. For details, see **FunctionGraph Metrics**.

7. Check your bills.

   After function execution is complete, you will be billed based on the number of function execution requests and execution duration. For details, see .**Bills**.

## Introduction to Dashboard

Log in to the FunctionGraph console and choose **Dashboard** in the navigation pane on the left.

- View your created functions/function quota, used storage/storage quota, and monthly invocations and resource usage.

  **Figure 1-3** Monthly statistics

- View tenant-level metrics, including invocations, top 10 functions by invocation, errors, top 10 functions by error, duration, and throttles.

  **Table 1-1** describes the function metrics.

  **Table 1-1** Function metrics

| Metric | Unit | Description |
| --- | --- | --- |
| Invocations | Count | Total number of invocation requests, including invocation errors and throttled invocations. In case of asynchronous invocation, the count starts only when a function is executed in response to a request. |
| The 10 Functions with the Most Invocations | - | Top 10 functions by invocation in the last day, last 3 days, or a custom period. |
| Duration | ms | **Maximum duration**: the maximum duration all functions are executed at a time within a period.<br>**Minimum duration**: the minimum duration all functions are executed at a time within a period.<br>**Average duration**: the average duration all functions are executed at a time within a period. |
| Errors | Count | Number of times that your functions failed with error code **200** being returned. Errors caused by function syntax or execution are also included. |

| Metric | Unit | Description |
| --- | --- | --- |
| The 10 Functions with the Most Errors | - | Top 10 functions by error in the last day, last 3 days, or a custom period. |
| Throttles | Count | Number of times that FunctionGraph throttles your functions due to the resource limit. |

- View flow metrics, including the number of invocations, duration, number of errors, and number of running workflows.

| Metric | Unit | Description |
| --- | --- | --- |
| Invocations | Count | Total number of invocation requests, including successful, failed, and ongoing requests. In case of asynchronous invocation, the count starts only when a flow executes in response to a request. |
| Duration | ms | Average time taken to execute a flow in a specified period. |
| Errors | Count | Number of times that flows failed to be executed. |
| Running Workflows | Count | Number of ongoing flow executions. |

# 2 Creating a User and Granting Permissions

This section describes how to use **Identity and Access Management (IAM)** to implement fine-grained permissions control for your FunctionGraph resources. With IAM, you can:

- Create IAM users for employees based on the organizational structure of your enterprise. Each IAM user has their own security credentials for accessing FunctionGraph resources.
- Grant only the permissions required for users to perform a task.
- Entrust an account or a cloud service to perform efficient O&M on your FunctionGraph resources.

If your account does not need individual IAM users, then you may skip over this chapter.

This section describes the procedure for granting permissions. For details, see **Figure 2-1**.

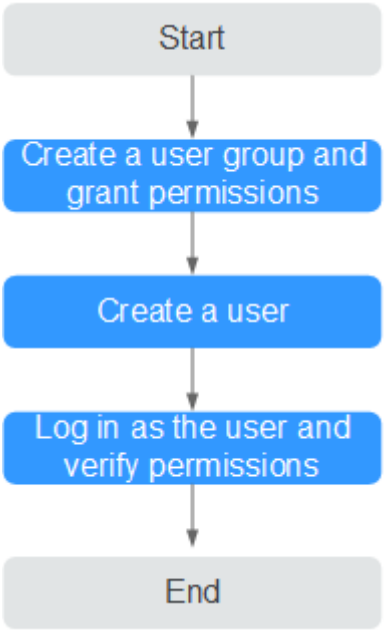
## Prerequisites

Before assigning permissions to user groups, you should learn about the system permissions listed in **Permissions Management**. For the system policies of other services, see **System Permissions**.

## Process

**Figure 2-1** Process for granting FunctionGraph permissions




1. **Create a user group and assign permissions**.

   Create a user group on the IAM console using an IAM account, and assign the **FunctionGraph Invoker** role to the group.

2. **Create an IAM user and add it to the user group**.

   Create a user on the IAM console using an IAM account and add the user to the group created in **1**.

3. **Logging In as an IAM User** and Verifying Permissions

   Log in to the FunctionGraph console as the created user, and check whether it only has read permissions for FunctionGraph.

   - Choose **Service List** > **FunctionGraph** to access the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**. Then click **Create Function**. If a message appears indicating insufficient permissions to perform the operation, the **FunctionGraph Invoker** role has already taken effect.
   - Choose any other service in the **Service List**. If a message appears indicating insufficient permissions to access the service, the **FunctionGraph Invoker** role has already taken effect.

## Creating a Custom Policy

Custom policies can be created as a supplement to the system policies of FunctionGraph. For details, see **Creating a Custom Policy**.

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.

- JSON: Create a JSON policy or edit an existing one.

The following contains examples of common FunctionGraph custom policies.

- Example 1: Authorizing a user to query function code and configuration

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "functiongraph:function:list",
            "functiongraph:function:getConfig",
            "functiongraph:function:getCode"
         ]
      }
   ]
}
```

- Example 2: Denying function deletion

  A policy with only "Deny" permissions must be used together with other policies. If both "Allow" and "Deny" permissions are assigned to a user, the "Deny" permissions take precedence over the "Allow" permissions.

  If you need to assign permissions of the **FunctionGraph FullAccess** policy to a user but prevent the user from deleting functions, create a custom policy for denying function deletion, and attach both policies to the group to which the user belongs. In this way, the user can perform all operations on FunctionGraph except deleting functions. The following is an example of a deny policy:

```
{
   "Version": "1.1",
   "Statement": [
      "Effect": "Deny",
      "Action": [
         "functiongraph:function:delete"
      ]
   ]
}
```

- Example 3: Configuring permissions for specific resources

  You can grant an IAM user permissions for specific resources. For example, to grant a user permissions for the **functionname** function in the **Default** application, set **functionname** to a specified resource path, that is, **FUNCTIONGRAPH:*:*:function:Default/functionname**.

  **Format**: **FUNCTIONGRAPH:*:*:function:***application or function name*

  For function resources, IAM automatically generates the resource path prefix **FUNCTIONGRAPH:*:*:function:**. The path is specified by the application and function name. The wildcard (*) is supported. For example, **FUNCTIONGRAPH:*:*:function:Default/*** indicates any function in the **Default** application.

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "functiongraph:function:list"
         ]
      },
      {
         "Effect": "Allow",
```

```
        "Action": [
            "functiongraph:function:listAlias",
            "functiongraph:function:listVersion",
            "functiongraph:function:getConfig",
            "functiongraph:function:getCode",
            "functiongraph:function:updateCode",
            "functiongraph:function:invoke",
            "functiongraph:function:updateConfig",
            "functiongraph:function:createVersion",
            "functiongraph:function:updateAlias",
            "functiongraph:function:createAlias"
        ],
        "Resource": [
            "FUNCTIONGRAPH:*:*:function:Default/*"
        ]
    }
  ]
}
```

## FunctionGraph Resources

A resource is an object that exists within a service. FunctionGraph resources include functions and triggers. To select these resources when creating a policy, specify their paths.

**Table 2-1** FunctionGraph resources and their paths

| Resource Type | Resource Name | Path |
| --- | --- | --- |
| function | Functions | Format:<br>FunctionGraph::::function:*group/function name*<br>Description:<br>IAM automatically generates the prefix **FunctionGraph:*:*:function:** for bucket resource paths.<br>By adding *Function name* to the end of the generated prefix, you can define a specific path. An asterisk ***** is allowed to indicate any function. For example, **FunctionGraph:*:*:function:default/*** indicates any function in the default group. |
| trigger | Triggers | Format:<br>FunctionGraph::::trigger:*trigger ID*<br>Description:<br>IAM automatically generates the prefix **FunctionGraph:*:*:trigger:** for object resource paths.<br>By adding *trigger ID* to the end of the generated prefix, you can define a specific path. An asterisk ***** is allowed to indicate any trigger. For example, **FunctionGraph:*:*:trigger:*** indicates any trigger. |

# 3 Supported Runtimes

To create a function in FunctionGraph, you need to specify a runtime that passes events, context, and responses. Choose from a built-in runtime or customize your own.

For details about the runtimes and versions supported by FunctionGraph, see **Table 3-1**.

**Table 3-1** Supported runtimes

| Runtime | Supported Version | Development Instructions |
| --- | --- | --- |
| Node.js | 6.10, 8.10, 10.16, 12.13, 14.18, 16.17, 18.15, 20.15 | For details about the function syntax, SDK APIs, and function development, see **Developing Functions in Node.js**. |
| Python | 2.7, 3.6, 3.9, 3.10, 3.12 | For details about the function syntax, SDK APIs, and function development, see **Developing Functions in Python**. |
| Java | 8, 11, 17, 21 (only available in **ME-Riyadh** and **TR-Istanbul**) | For details about the function syntax, SDK APIs, and function development, see **Developing Functions in Java**. |

| Runtime | Supported Version | Development Instructions |
| --- | --- | --- |
| Go | 1.x | For details about the function syntax, SDK APIs, and function development, see **Developing Functions in Go**. |
| C# | .NET Core 2.1, .NET Core 3.1, .NET Core 6.0, .NET Core 8.0 (only in **ME-Riyadh** and **TR-Istanbul**) | For details about the function syntax, SDK APIs, and function development, see **Developing Functions in C#**. |
| PHP | 7.3 and 8.3 | For details about the function syntax, SDK APIs, and function development, see **Developing Functions in PHP**. |
| Custom | - | - |
| Cangjie | 1.0 | - |

# 4 Creating a Function

## 4.1 Creating a Function from Scratch

### 4.1.1 Creating an Event Function

Creating a function is the first step in using FunctionGraph. To trigger tasks via cloud events, create an event function and configure code, network, and triggers as needed. This section shows how to create one via the console.

For details about the comparison between event functions and HTTP functions, see **Table 4-1**.

**Table 4-1** Comparison between event functions and HTTP functions

| Item | Event Function | HTTP Function |
| --- | --- | --- |
| Usage | It is used to process files and data flows, which can be **triggered by events of various cloud products**. It is also used to process asynchronous requests and track and save the status of each asynchronous invocation. | It supports popular web application frameworks and AI projects. You can access the functions using a browser or by calling a URL. |

| Item | Event Function | HTTP Function |
| --- | --- | --- |
| Scenario | • Cloud product integration: OBS real-time file processing and LTS log processing.<br>• ETL data processing: database data cleaning and message queue processing.<br>• Regular tasks: scheduled, periodic, and script tasks.<br>• Multimedia processing: audio/video transcoding, live recording, and image processing. | • Application building based on popular web frameworks such as Express and Flask.<br>• AI model inference service building based on Stable Diffusion, ComfyUI, and DeepSeek.<br>• Migration of existing applications, such as HTML5 websites, REST APIs, mobile apps, applets, and game settlement. |

## Notes and Constraints

By default, an account can create a maximum of 400 functions. To increase the quota, **submit a service ticket**.

## Prerequisites

- To perform the operations described in this section, ensure that you have the **FunctionGraph Administrator** permissions, that is, the full permissions for FunctionGraph. For more information, see **Permissions Management**.
- To access other cloud services such as Log Tank Service (LTS) and Virtual Private Cloud (VPC), create an agency by referring to **Creating a Function Agency**. If FunctionGraph does not need to access other cloud services, you do not need to create or select an agency.
- To enable FunctionGraph to access resources in a VPC, create a VPC and subnet by referring to **Creating a VPC and a Subnet**.

## Creating an Event Function

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

**Step 2** On the **Function List** page, click **Create Function** in the upper right corner.

**Step 3** Select **Create from scratch** and configure basic information by referring to **Table 4-2** and advanced settings by referring to **Table 4-3** as shown in **Figure 4-1**.

**Figure 4-1** Basic function information

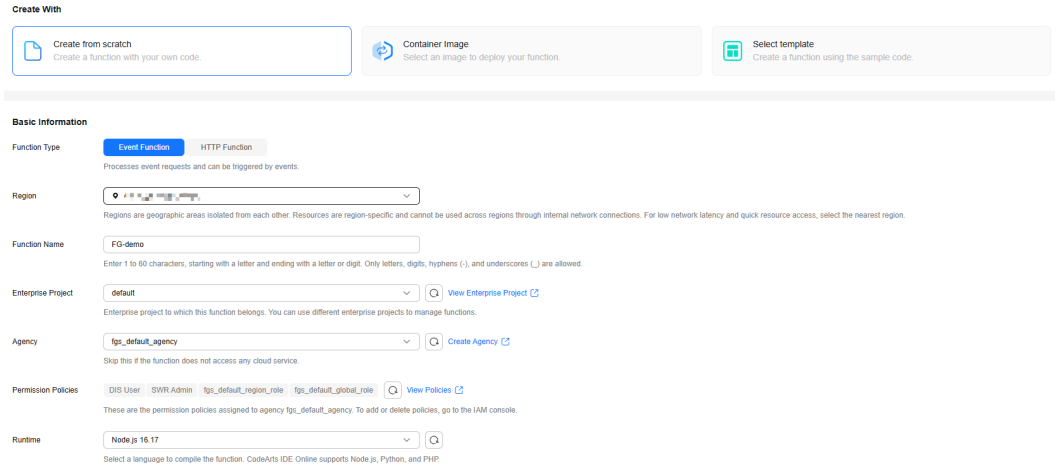



**Table 4-2** Basic information parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Function Type | Select **Event Function**.<br>An event function is triggered by a specific event, which is usually a request event in JSON format. | Event Function |
| Region | Select the region where the function is located.<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. | CN East-Shanghai1 |
| Function Name | 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 ends with a letter or digit. | FG-demo |
| Enterprise Project | 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.** | default |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Agency | Select an agency for the function. An agency is used to authorize FunctionGraph to access other cloud services. If FunctionGraph does not access any cloud service, you do not need to select an agency.<br>By default, **Use no agency** is used. You can select an existing agency.<br>If no default agency is available, FunctionGraph allows you to quickly create a default agency named **fgs_default_agency**. For details, see **Default Agency**. | fgs_default_agency |
| Permission Policies | **This parameter is displayed only when an agency is selected.**<br>After an agency is selected, the permission policies associated with the agency are displayed. For details about how to adjust them, see **Modifying a Function Agency**. | DIS User; SWR Admin; fgs_default_region_role; fgs_default_global_role |
| Runtime | Select a runtime to compile the function.<br>For details about the runtimes supported by FunctionGraph, see **Supported Runtimes**.<br>• CloudIDE supports only online editing of Node.js, Python, PHP, and custom runtimes.<br>• After a function is created, the runtime language cannot be changed. | Node.js 16.17 |

**Figure 4-2** Advanced setting parameters

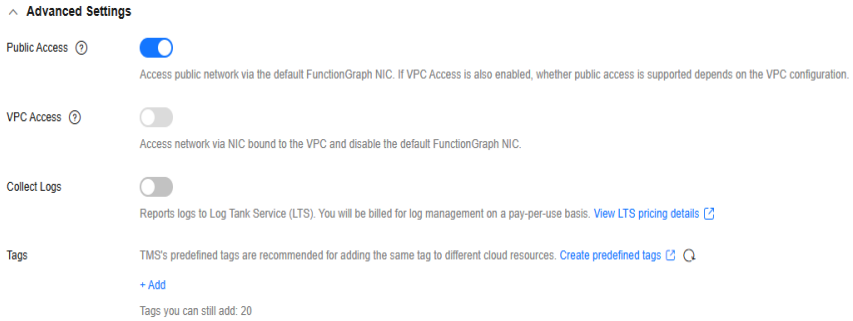


**Table 4-3** Advanced setting parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Public Access | 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. | Enabled |
| VPC Access | **To enable this feature, you need to configure an agency with VPC management permissions. If you select Use no agency in the Basic Information area, this feature cannot be enabled.**<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.<br>After this feature is enabled, you can select the VPC and subnet that the function needs to access. | Disabled |
| Collect Logs | After it is enabled, logs generated during function execution will be reported to LTS.<br>**CAUTION**<br>LTS will be billed on a pay-per-use basis. For details, see **LTS Pricing Details**.<br>Configure the following parameters:<br>• Log Configuration<br>– **Auto**: Use the default log group and log stream.<br>– **Custom**: Select the **log group** and **log stream** you created.<br>• Log Tag<br>You can filter function logs on the LTS console by tag. For details, see **Log Management**.<br>A tag key or value can contain a maximum of 64 characters, including only digits, letters, underscores (_), and hyphens (-). A maximum of 10 tags can be added. | Disabled |
| Tags | Add tags to identify and categorize functions.<br>Tags enable quick search and management in the function list.<br>Each function supports up to 20 tags with a many-to-many relationship. | - |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Static Encryption with KMS | This parameter can be configured only in the **LA-Sao Paulo1** region.<br>Determine whether to use KMS-based static encryption for function code.<br>**CAUTION**<br>DEW charges you on a pay-per-use basis. For details, see **DEW Billing**.<br>You can select the following encryption types:<br>• **functiongraph/default (default)**: The function automatically creates a default key in DEW under your account.<br>If you use the default key for encryption and decryption for the first time, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:create, kms:cmk:get, and kms:cmk:list.<br>• **Customer master key (CMK)**: You can select a created key to encrypt the function code. For details about how to create a customer master key, see **Creating a Custom Key**. To use customer master key, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:get and kms:cmk:list.<br>**CAUTION**<br>If you select **Customer master key (CMK)**, do not delete the CMK used for function encryption in DEW. Otherwise, the function execution will fail because encrypted data cannot be decrypted.<br>Configure the agency permission policy on the IAM console by referring to **Creating a Custom Policy in JSON View**. | functiongraph/default (default) |

**Step 4** After the configuration is complete, click **Create Function**. The function details page is displayed, and a message is displayed in the upper part of the page, indicating that the function is created.

**Step 5** After the function is created, configure it based on service requirements by referring to **Configuring Functions**.

**----End**

## Helpful Links

- For different scenarios, you can create event and HTTP functions, compile functions using built-in runtimes, custom runtimes, or custom images, and allocate GPU compute resources to functions. For details about how to select a function type, see **Function Selection**.

- You can create functions using APIs. For details, see **Creating a Function**.
- For FAQs about function creation, see **Function Creation FAQs**.
- For details about how to create a simple event function, see **Creating a Function from Scratch and Executing the Function** and **Creating an Event Function Using a Container Image and Executing the Function**.

## 4.1.2 Creating an HTTP Function

To use a popular web application framework to compile function code and build an AI project, you can create an HTTP function and configure the function code, network, and triggers based on service requirements. This section describes how to create an HTTP function and provides an example of how to use it.

For details about the comparison between event functions and HTTP functions, see **Table 4-4**.

**Table 4-4** Comparison between event functions and HTTP functions

| Item | Event Function | HTTP Function |
| --- | --- | --- |
| Usage | It is used to process files and data flows, which can be **triggered by events of various cloud products**. It is also used to process asynchronous requests and track and save the status of each asynchronous invocation. | It supports popular web application frameworks and AI projects. You can access the functions using a browser or by calling a URL. |
| Scenario | • Cloud product integration: OBS real-time file processing and LTS log processing.<br>• ETL data processing: database data cleaning and message queue processing.<br>• Regular tasks: scheduled, periodic, and script tasks.<br>• Multimedia processing: audio/video transcoding, live recording, and image processing. | • Application building based on popular web frameworks such as Express and Flask.<br>• AI model inference service building based on Stable Diffusion, ComfyUI, and DeepSeek.<br>• Migration of existing applications, such as HTML5 websites, REST APIs, mobile apps, applets, and game settlement. |

### HTTP Function Overview

HTTP functions focus on optimizing web service scenarios. You can directly send HTTP requests to URLs to trigger function execution and use your own web services. HTTP/1.1 is supported.

HTTP functions have the following advantages:

- Support for multiple frameworks

You can use common web frameworks, such as Node.js Express and Koa, to write HTTP functions, and migrate your local web framework services to the cloud with least modifications.

- Fewer request processing steps

  Functions can directly receive and process HTTP requests, eliminating the need for API Gateway to convert the JSON format. This accelerates request processing and improves web service performance.

- Premium writing experience

  Writing HTTP functions is similar to writing native web services. You can also use native Node.js APIs to enjoy local development-like experience.

## Common Request Headers of HTTP Functions

HTTP request headers are an important part of the HTTP protocol for passing metadata. When a function is invoked, specific metadata or configuration information can be passed. **Table 4-5** describes the common request headers carried by functions by default.

The key information of HTTP functions can be transferred only through request headers. For details about how to obtain the AK, SK, and token of HTTP functions, see **Transferring Secret Keys Through the Request Header**.

**Table 4-5** Default request header fields

| Field | Description |
| --- | --- |
| X-CFF-Request-Id | ID of the current request. |
| X-CFF-Memory | Memory allocated to the function. |
| X-CFF-Timeout | Function timeout. |
| X-CFF-Func-Version | Function version. |
| X-CFF-Func-Name | Function name. |
| X-CFF-Project-Id | Project ID of the function. |
| X-CFF-Package | App to which the function belongs. |
| X-CFF-Region | Region where the function is located. |

## Notes and Constraints

**Table 4-6** Constraints on creating HTTP functions

| Item | Description |
| --- | --- |
| Function quantity | By default, an account can create a maximum of 400 functions. To increase the quota, **submit a service ticket**. |

| Item | Description |
| --- | --- |
| HTTP function | • HTTP functions do not distinguish between programming languages. The handler must be set in the **bootstrap** file. You can directly write the startup command, and allow access over port **8000**. The bound IP address is **127.0.0.1**.<br>• The HTTP response body cannot exceed 6 MB.<br>**A valid HTTP function response must contain body(String), statusCode(int), headers(Map), and isBase64Encoded(boolean). By default, the response is encoded using Base64. The default value of isBase64Encoded is true. The same applies to other frameworks.**<br>• Only APIG and APIC triggers can be created for HTTP functions. For details, see **Using an APIG Trigger**.<br>• HTTP functions cannot be executed for a long time, invoked asynchronously, or retried. |
| Other | • The **bootstrap** file is the startup file of the HTTP function. The HTTP function can only read **bootstrap** as the startup file name. If the file name is not **bootstrap**, the service cannot be started. For more information, see the **bootstrap file example**.<br>If you run the JAR package, you are advised to add the JVM parameter **-Dfile.encoding=utf-8** to **bootstrap**. Otherwise, garbled Chinese characters may be displayed.<br>• When a function initiates an HTTP request, the request IP address is dynamic for private network access and fixed for public network access. For details about the request IP address, **submit a service ticket** to contact technical support. |

## Prerequisites

- To perform the operations described in this section, ensure that you have the **FunctionGraph Administrator** permissions, that is, the full permissions for FunctionGraph. For more information, see **Permissions Management**.
- To access other cloud services such as Log Tank Service (LTS) and Virtual Private Cloud (VPC), create an agency by referring to **Creating a Function Agency**. If FunctionGraph does not need to access other cloud services, you do not need to create or select an agency.
- To enable FunctionGraph to access resources in a VPC, create a VPC and subnet by referring to **Creating a VPC and a Subnet**.

## Creating an HTTP Function

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

**Step 2** On the **Function List** page, click **Create Function** in the upper right corner.

**Step 3** Select **Create from scratch** and configure basic information by referring to **Table 4-7** and advanced settings by referring to **Table 4-3** as shown in **Figure 4-3**.

**Figure 4-3** Basic information for creating an HTTP function

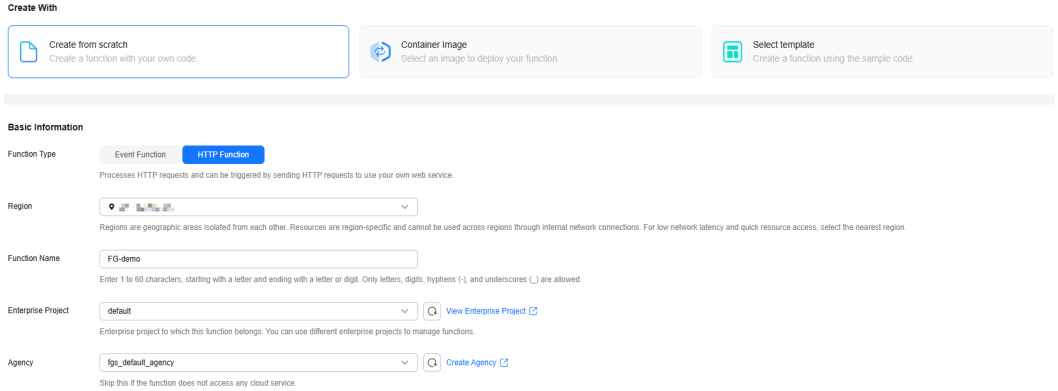



**Table 4-7** Basic function information

| Parameter | Description | Example Value |
| --- | --- | --- |
| Function Type | Select **HTTP Function**.<br>The HTTP functions process HTTP requests. You can send an HTTP request to a URL to trigger the function and use your web services. | HTTP Function |
| Region | Select the region where the function is located.<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. | CN East-Shanghai1 |
| Function Name | 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 ends with a letter or digit. | FG-demo |
| Enterprise Project | 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.** | default |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Agency | Select an agency for the function. An agency is used to authorize FunctionGraph to access other cloud services. If FunctionGraph does not access any cloud service, you do not need to select an agency.<br>By default, **Use no agency** is used. You can select an existing agency.<br>If no default agency is available, FunctionGraph allows you to quickly create a default agency named **fgs_default_agency**. For details, see **Default Agency**. | fgs_default_agency |
| Permission Policies | **This parameter is displayed only when an agency is selected.**<br>After an agency is selected, the permission policies associated with the agency are displayed. For details about how to adjust them, see **Modifying a Function Agency**. | DIS User; SWR Admin; fgs_default_region_role; fgs_default_global_role |

**Figure 4-4** Advanced setting parameters

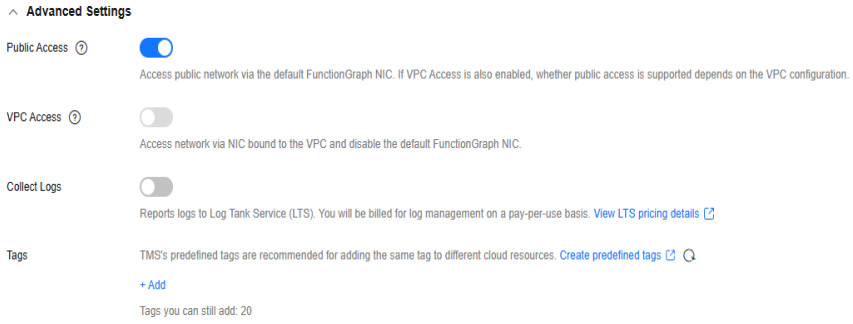



**Table 4-8** Advanced setting parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Public Access | 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. | Enabled |

| Parameter | Description | Example Value |
| --- | --- | --- |
| VPC Access | **To enable this feature, you need to configure an agency with VPC management permissions. If you select Use no agency in the Basic Information area, this feature cannot be enabled.**<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.<br>After this feature is enabled, you can select the VPC and subnet that the function needs to access. | Disabled |
| Collect Logs | After it is enabled, logs generated during function execution will be reported to LTS.<br>**CAUTION**<br>LTS will be billed on a pay-per-use basis. For details, see **LTS Pricing Details**.<br>Configure the following parameters:<br>• Log Configuration<br>– **Auto**: Use the default log group and log stream.<br>– **Custom**: Select the **log group** and **log stream** you created.<br>• Log Tag<br>You can filter function logs on the LTS console by tag. For details, see **Log Management**.<br>A tag key or value can contain a maximum of 64 characters, including only digits, letters, underscores (_), and hyphens (-). A maximum of 10 tags can be added. | Disabled |
| Tags | Add tags to identify and categorize functions.<br>Tags enable quick search and management in the function list.<br>Each function supports up to 20 tags with a many-to-many relationship. | - |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Static Encryption with KMS | This parameter can be configured only in the **LA-Sao Paulo1** region.<br>Determine whether to use KMS-based static encryption for function code.<br>**CAUTION**<br>DEW charges you on a pay-per-use basis. For details, see **DEW Billing**.<br>You can select the following encryption types:<br>• **functiongraph/default (default)**: The function automatically creates a default key in DEW under your account.<br>If you use the default key for encryption and decryption for the first time, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:create, kms:cmk:get, and kms:cmk:list.<br>• **Customer master key (CMK)**: You can select a created key to encrypt the function code. For details about how to create a customer master key, see **Creating a Custom Key**. To use customer master key, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:get and kms:cmk:list.<br>**CAUTION**<br>If you select **Customer master key (CMK)**, do not delete the CMK used for function encryption in DEW. Otherwise, the function execution will fail because encrypted data cannot be decrypted.<br>Configure the agency permission policy on the IAM console by referring to **Creating a Custom Policy in JSON View**. | functiongraph/default (default) |

**Step 4** After the configuration is complete, click **Create Function**. The function details page is displayed, and a message is displayed in the upper part of the page, indicating that the function is created.

**Step 5** After the function is created, configure it based on service requirements by referring to **Configuring Functions**.

**----End**

## HTTP Function Usage Example

The following uses an example to describe how to create and configure an HTTP function.

**Before calling an API, ensure that the network of your service system can communicate with the API access domain name or address.**

- If the service system and the HTTP functions are in the same VPC, the API can be directly accessed.
- If the service system and the HTTP functions are in different VPCs of a region, connect them using a peering connection. For details, see .
- If the service system and the HTTP functions are in different VPCs of different regions, create a cloud connection and load the two VPCs to connect them. For details, see .
- If the service system and the HTTP functions are connected over the public network, ensure that the HTTP function has been bound with an EIP.

## Step 1: Creating a Code Package

1. Prepare a Node.js script file named **index.js**. The following is a code example:

```
const http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) {   //create web server
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><h2>This is http function.</h2></body></html>');
    res.end();
});

server.listen(8000, '127.0.0.1'); //6 - listen for any incoming requests

console.log('Node.js web server at port 8000 is running..')
```

   To parse the body of an event, refer to the following code example:

```
const http = require('http');
const server = http.createServer((req, res) => {
    //Process only POST requests (PUT, PATCH, DELETE, and other methods with request bodies can also be processed).
    if (req.url.startsWith("/apig-event-template")) {
        let body = '';
        // 1. Listen to the 'data' event to collect data blocks.
        req.on('data', (chunk) => {
            body += chunk; // Convert the binary data block to a string.
            // (Optional) Set the request body size limit (to prevent memory overflow).
            if (body.length > 1e6) { // 1 MB
                req.destroy(); // Terminate the connection.
            }
        });
        // 2. Listen to the 'end' event to process the complete request body.
        req.on('end', () => {
            try {
                // Process different formats based on contentType.
                const contentType = req.headers['content-type'] || '';
                // Process the JSON format.
                if (contentType.includes('application/json')) {
                    const data = JSON.parse(body);
                    console.log('Received JSON:', data);
                    //Process the form format (URL encoding).
                } else if (contentType.includes('application/x-www-form-urlencoded')) {
                    const params = new URLSearchParams(body);
                    const data = Object.fromEntries(params);
                    console.log('Received form data:', data);
                    //Process common text.
                } else {
                    console.log('Received raw text:', body);
                }
                //Respond to the client.
                res.writeHead(200, { 'Content-Type': 'text/plain' });
                res.end('Body received');
            } catch (error) {
                //Process errors (for example, JSON parsing failure).
                console.error('Error processing body:', error);
                res.writeHead(400, { 'Content-Type': 'text/plain' });
```

```
            res.end('Invalid body format');
        }
    });
    //3. Process request errors.
    req.on('error', (error) => {
        console.error('Request error:', error);
        res.writeHead(500, { 'Content-Type': 'text/plain' });
        res.end('Internal server error');
    });
  } else {
    // Process the method without request body.
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('No body needed');
  }
});
// Start the server.
const PORT = 8000;
server.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

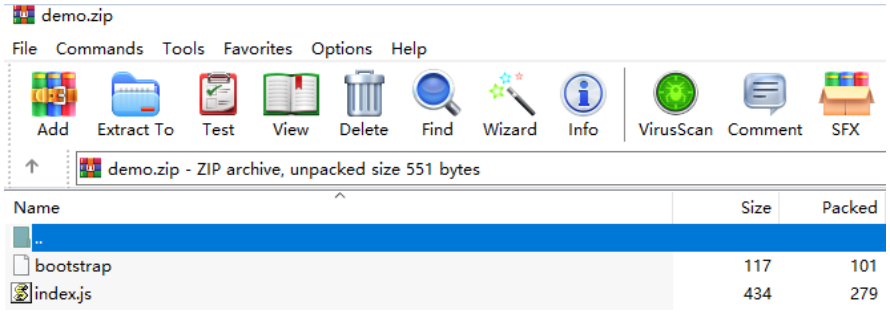
2. You have prepared a **bootstrap** file as the startup file of the HTTP function.

   The content of the **bootstrap** file is as follows:

   ```
   /opt/function/runtime/nodejs12.13/rtsp/nodejs/bin/node $RUNTIME_CODE_ROOT/index.js
   ```

3. Compress the preceding two files into a ZIP package.

**Figure 4-5** Compressing files into a ZIP package




For HTTP functions in Python, add the **-u** parameter in the **bootstrap** file to ensure that logs can be flushed to the disk. For example:

```
/opt/function/runtime/python3.6/rtsp/python/bin/python3 -u $RUNTIME_CODE_ROOT/index.py
```

To use another runtime, change the runtime path by referring to **Table 4-9**. The code package path does not need to be changed.

**Table 4-9** Paths for different runtimes

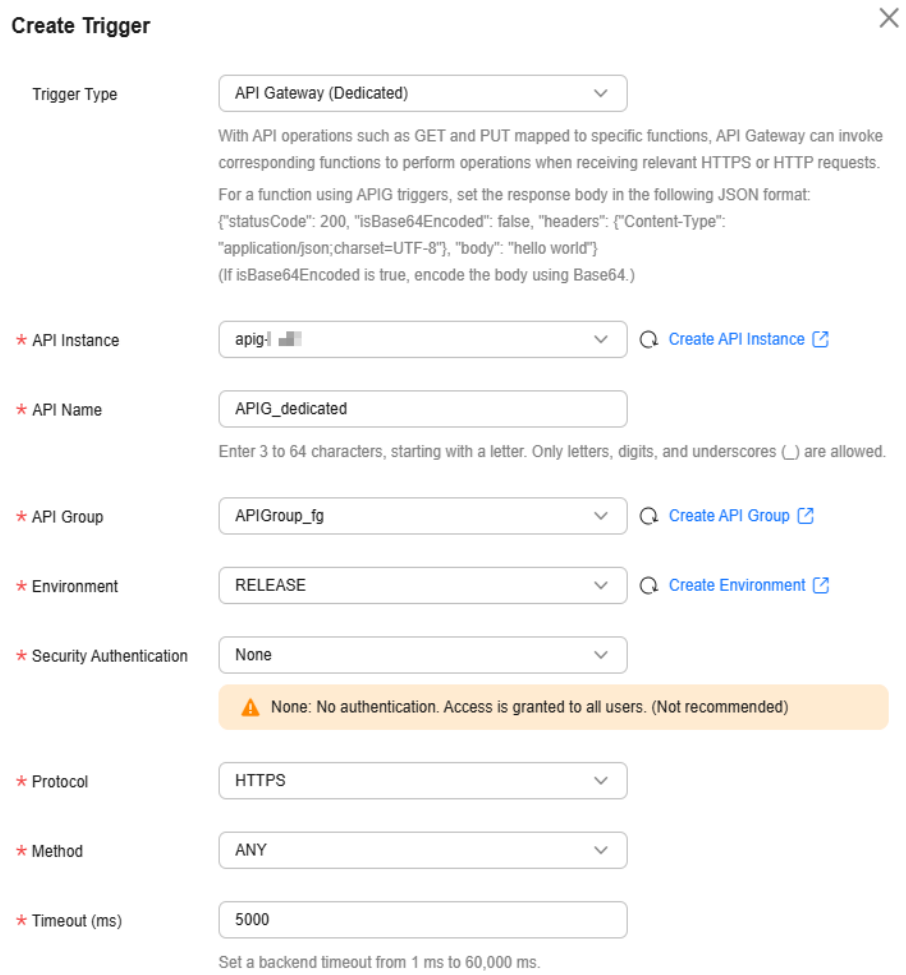
| Runtime | Path |
| --- | --- |
| Java 21 | /opt/function/runtime/java21/rtsp/jre/bin/java |
| Python 3.12 | /opt/function/runtime/python3.12/rtsp/python/bin/python3 |
| PHP 8.3 | /opt/function/runtime/php8.3/rtsp/php/bin/php |
| C# (.NET Core 8.0) | /opt/function/runtime/dotnet8.0/rtsp/dotnet/dotnet |

## Step 2: Deploying the Code Package

1. Create an HTTP function by referring to **Creating an HTTP Function**.
2. On the **Code** page of the HTTP function, click **Upload** > **Local ZIP**, and upload the ZIP file created in **Step 1: Creating a Code Package**.
3. Create a trigger.
   a. Choose **Configuration** > **Triggers** and click **Create Trigger**.
   b. Configure the trigger information, as shown in **Figure 4-6**. The following uses API Gateway (Dedicated) as an example. For details about the parameters, see **API Gateway (Dedicated) Trigger**.

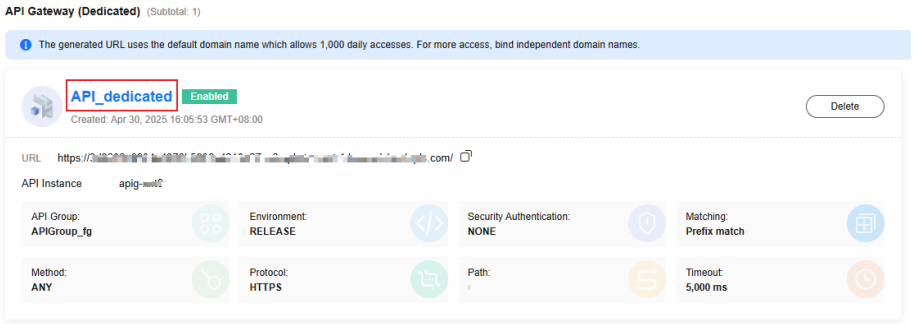
   **Figure 4-6** Create a trigger.

   In this example, **Security Authentication** is set to **None**. You are advised to enable App or IAM authentication in the production environment.
   c. Click **OK**.
4. Publish the API.
   a. On the **Triggers** tab page, click an API name to go to the API overview page.

**Figure 4-7** API trigger

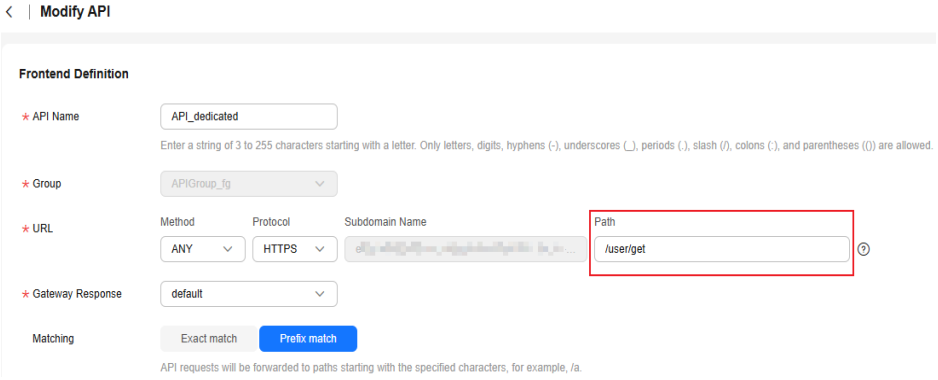



b. Click **Edit** in the upper right corner. The **Basic Information** page is displayed.

**Figure 4-8** Editing an API


c. Change the value of **Path** to **/user/get** and click **Finish**.

**Figure 4-9** Defining an API request



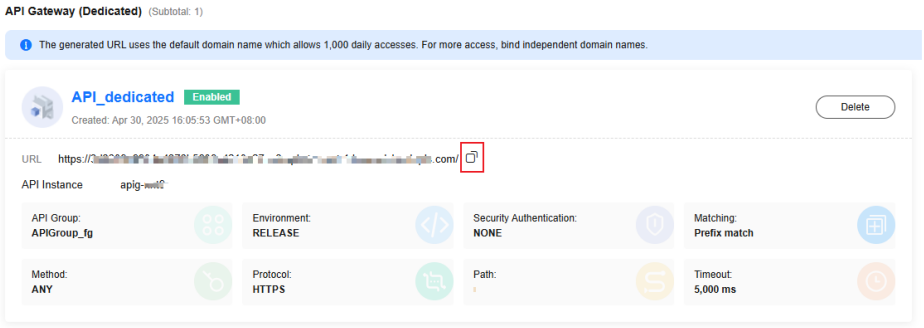

d. On the API details page, click **Publish Latest Version** in the upper right corner. On the displayed page, click **OK**.

## Step 3: Triggering Function Execution

1. Go to the FunctionGraph console, choose **Functions** > **Function List** in the navigation pane, and click the created HTTP function to go to its details page.
2. Choose **Configuration** > **Triggers**, copy the URL as shown in **Figure 4-10**, and access it using a browser.

**Figure 4-10** Copying the URL




3. View the request result.

**Figure 4-11** Viewing the request result

```
<html><body><h2>This is http function.</h2></body></html>
```

## Helpful Links

- For details about how to build a FunctionGraph HTTP function using Go, see **Building a FunctionGraph HTTP Function Using Go**.
- For details about how to build an HTTP function using an existing Spring Boot project, see .
- FunctionGraph allows you to create event and HTTP functions by compiling code online, uploading code files, or using container images. And it supports GPU computing resources. For details about how to select a function type, see **Function Type Selection**.
- You can create functions using APIs. For details, see **Creating a Function**.
- For FAQs about function creation, see **Function Creation FAQs**.

# 4.2 Creating a Function Using a Template

## Overview

FunctionGraph provides function templates for different scenarios. When you create a function using a template, the template automatically configures the function code, environment variables, and triggers.

You can filter function templates by function type and scenario. Click **More** to view the description, input and output parameters, and precautions of a function template.

## Prerequisites

To perform the operations described in this section, ensure that you have the **FunctionGraph Administrator** permissions, that is, the full permissions for FunctionGraph. For more information, see **Permissions Management**.

## Creating a Function Using a Template

The **context-class-introduction** template for Python 3.6 is used as an example. You can also select other templates.

1. Log in to the **FunctionGraph console**, and create a function using either of the following methods:
   - In the navigation pane, choose **Functions** > **Function List**. Then click **Create Function** in the upper right corner. Select **Select template** for the creation mode.
   - In the navigation pane, choose **Templates**. Hover the cursor over the target template. Click **Configure**. The **Create Function** page is displayed.
2. After you select a function template, the code and configuration information built in the template are loaded, as shown in **Figure 4-12**. For details, see **Table 4-10**.

**Figure 4-12** Creating a function using a template


**Table 4-10** context template parameter table

| Parameter | Description | Example Value |
| --- | --- | --- |
| Templates | The selected function template is displayed by default. To change it, click **Reselect**. | context-class-introduction |
| Region | Select the region where the function is located.<br>**Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.** | CN East-Shanghai1 |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Function Name | 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 ends with a letter or digit. | context-class-introduction-python-36 |
| Enterprise Project | 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.** | default |
| Agency | Select an agency for the function. An agency is used to authorize FunctionGraph to access other cloud services. For example, if FunctionGraph needs to access LTS or VPC, you must select an agency with required service permissions. If FunctionGraph does not access any cloud services, you do not need to select an agency.<br>By default, **Use no agency** is used. You can select an existing agency.<br>If no default agency is available, FunctionGraph allows you to quickly create a default agency named **fgs_default_agency**. For details, see **Default Agency**. | Use no agency |
| Runtime | Select a runtime to compile the function. The runtime language is automatically selected based on the function template and cannot be changed. | Python 3.6 |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Advanced Settings | • **Public Access**: 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.<br>• **VPC Access**: 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.<br>**To enable this feature, you need to configure an agency with VPC management permissions. If you select Use no agency in the Basic Information area, this feature cannot be enabled.**<br>• **Collect Logs**: 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.**<br>• **Tags**: Add tags to functions. Tags are used to identify and classify function resources.<br>• **Static Encryption with KMS** (supported only in the **LA-Sao Paulo1** region): Select whether to use KMS static code encryption.<br>**DEW charges you on a pay-per-use basis. For details, see DEW Billing.** | • **Public Access**: Enabled.<br>• **VPC Access**: Disabled.<br>• **Collect Logs**: Disabled.<br>• Tags: -<br>• **Static Encryption with KMS**: Disabled. |

3. Click **Create Function**. For details about other configurations, see **Configuring Functions**.

## Helpful Links

- For FAQs about function creation, see **Function Creation FAQs**.
- Manage function templates through APIs. For details, see **Function Template APIs**.

# 4.3 Creating a Function with an Image

FunctionGraph supports loading and running functions in container images. Compared with directly uploading code, you can use custom code packages, which is flexible and reduces migration costs.

Public and private images are supported. For details, see **Setting Image Attributes**.

## Notes and Constraints

**Table 4-11** Notes and Constraints

| Item | Description |
| --- | --- |
| Custom container image port | • The custom container image must contain an HTTP server with listening port 8000.<br>• The port of a custom container image must be 8000. |
| Container image-based function | • When creating an event function, create an HTTP server to implement a handler (method: **POST**, path: **/invoke**) and an initializer (method: **POST**, path: **/init**).<br>• The function must return a valid HTTP response that complies with the following structure.<br>`{`<br>`"isBase64Encoded": true\|false,`<br>`"statusCode": httpStatusCode,`<br>`"headers": {"headerName":"headerValue",...},`<br>`"body": "..."`<br>`}`<br>• **Command**, **Args**, and **Working dir** can contain up to 5120 characters.<br>• When a function is executed at the first time, the image is pulled from SWR, and the container is started during cold start of the function, which takes a certain period of time. If there is no image on a node during subsequent cold starts, an image will be pulled from SWR.<br>• The image package cannot exceed 10 GB. For a larger package, reduce the capacity. For example, mount the data of a question library to a container where the data was previously loaded through an external file system.<br>• When an out of memory (OOM) error occurs, view the memory usage in the function execution result. |

## Prerequisites

- To perform the operations described in this section, ensure that you have the **FunctionGraph Administrator** permissions, that is, the full permissions for FunctionGraph. For more information, see **Permissions Management**.
- You have created an agency with the SWR Admin permission as instructed in **Configuring Agency Permissions**. Images are stored in SWR, and only users with this permission can invoke and pull images.

## Creating a Function

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

3. Select **Container Image** for creation mode. For details about the basic information, see **Table 4-12**. For details about the advanced settings, see **Table 4-13**.

**Table 4-12** Container image configuration

| Parameter | Description | Example Value |
| --- | --- | --- |
| Function Type | Two types are available:<br>• **Event Function**: Requests are usually in JSON format. All trigger types supported by FunctionGraph can be used to trigger event functions.<br>• **HTTP Function**: triggered once HTTP requests are sent to specific URLs. | Event Function |
| Region | Select a region where you will deploy your code.<br>**Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.** | CN East-Shanghai1 |
| Function Name | Name of the function, which must meet the following requirements:<br>• Consists of 1 to 60 characters, and can contain letters, digits, hyphens (-), and underscores (_).<br>• Starts with a letter and ends with a letter or digit. | SWR-demo |
| Enterprise Project | 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.** | default |
| Agency | Select an agency for the function. To access other cloud services through an agency, select an agency with the SWR Admin permission. For details about how to create an agency, see **Creating a Function Agency**.<br>If no default agency is available, FunctionGraph allows you to quickly create a default agency named **fgs_default_agency**. For details, see **Default Agency**. | fgs_default_agency |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Container Image | Enter an image URL, that is, the location of the container image. You can click **View Image** to view private and shared images. You can also click **Select SWR Image** and select a public or private image from the image list. After the image is selected, the image URL is automatically filled in. For details about how to create an image, see **Creating an Image**.<br>Image in SWR, for example, **swr.region_id.myhuaweicloud.com/my_group/my_image:latest**. | swr.region_id.myhuaweicloud.com/my_group/my_image:latest |
| Listening Port | **Currently, this parameter is available only in the ME-Riyadh region.**<br>Port listened by the HTTP server in the code, which must be the same as the **EXPOSE** port configured in the image. The port number ranges from 1025 to 65535. Default: **8000**. | 8000 |
| Container Image Override | Optional.<br>If this parameter is set, the image configuration in the **dockerfile** file will be overwritten. If this parameter is not set, the default image configuration is used.<br>• **CMD**: container startup command. Example: **/bin/sh**. If no command is specified, the entrypoint or CMD in the image configuration will be used. Enter one or more commands separated with commas (,).<br>• **Args**: container startup parameter. Example: **-args,value1**. If no argument is specified, CMD in the image configuration will be used. Enter one or more arguments separated with commas (,).<br>• **Working Dir**: working directory where a container runs. If no directory is specified, the directory in the image configuration will be used. The directory must start with a slash (/).<br>• **User ID**: user ID for running the image. If no user ID is specified, the default value **1003** will be used.<br>• **Group ID**: user group ID. If no user group ID is specified, the default value **1003** will be used. | - |

**Figure 4-13** Advanced setting parameters

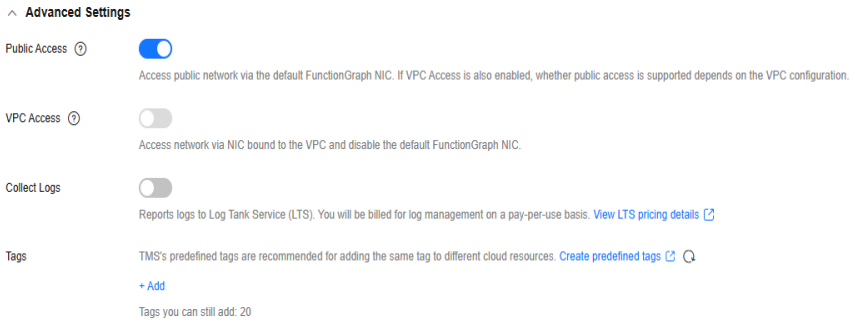



**Table 4-13** Advanced setting parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Public Access | 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. | Enabled |
| VPC Access | **To enable this feature, you need to configure an agency with VPC management permissions. If you select Use no agency in the Basic Information area, this feature cannot be enabled.**<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.<br>After this feature is enabled, you can select the VPC and subnet that the function needs to access. | Disabled |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Collect Logs | After it is enabled, logs generated during function execution will be reported to LTS.<br>**CAUTION**<br>LTS will be billed on a pay-per-use basis. For details, see **LTS Pricing Details**.<br>Configure the following parameters:<br>• Log Configuration<br>– **Auto**: Use the default log group and log stream.<br>– **Custom**: Select the log group and log stream you created.<br>• Log Tag<br>You can filter function logs on the LTS console by tag. For details, see **Log Management**.<br>A tag key or value can contain a maximum of 64 characters, including only digits, letters, underscores (_), and hyphens (-). A maximum of 10 tags can be added. | Disabled |
| Tags | Add tags to identify and categorize functions.<br>Tags enable quick search and management in the function list.<br>Each function supports up to 20 tags with a many-to-many relationship. | - |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Static Encryption with KMS | This parameter can be configured only in the **LA-Sao Paulo1** region.<br>Determine whether to use KMS-based static encryption for function code.<br>**CAUTION**<br>DEW charges you on a pay-per-use basis. For details, see **DEW Billing**.<br>You can select the following encryption types:<br>• **functiongraph/default (default)**: The function automatically creates a default key in DEW under your account.<br>If you use the default key for encryption and decryption for the first time, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:create, kms:cmk:get, and kms:cmk:list.<br>• **Customer master key (CMK)**: You can select a created key to encrypt the function code. For details about how to create a customer master key, see **Creating a Custom Key**. To use customer master key, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:get and kms:cmk:list.<br>**CAUTION**<br>If you select **Customer master key (CMK)**, do not delete the CMK used for function encryption in DEW. Otherwise, the function execution will fail because encrypted data cannot be decrypted.<br>Configure the agency permission policy on the IAM console by referring to **Creating a Custom Policy in JSON View**. | functiongraph/default (default) |

4. Click **Create Function**. The function details page is displayed. For details about other configurations, see **Configuring Functions**.

## Updating the Function Image

On the **Code** tab page of the image-based function, click **Deploy Image**. In the displayed dialog box, enter the new image URL and click **OK**. To obtain the URL, perform the following operations:

1. Log in to the SWR console. In the navigation pane, choose **My Images**.
2. Click the **Private Images** or **Images From Others** tab. In the image list, click the image name to go to the details page.

3. Click the **Tags** tab, copy the download command in the image tag list, and delete **docker pull** from the command to obtain the image URL.

### Helpful Links

- FunctionGraph allows you to create event and HTTP functions by compiling code online, uploading code files, or using container images. And it supports GPU computing resources. For details about how to select a function type, see **Function Selection**.
- You can create functions using APIs. For details, see **Creating a Function**.
- FAQ for image-based functions: **After I Updated an Image with the Same Name, Reserved Instances Still Use the Old Image. What Can I Do?**

## 4.4 Creating a Function by Running Terraform Commands

Terraform is an open-source tool that lets you safely build, change, and version infrastructure. Just declare the final state of the infrastructure you want in the configuration files, without specifying how to implement the state.

Terraform advantages:

- More consistent architecture: Manual configuration errors and drift are reduced.
- Lower O&M costs: VMs are managed programmatically, reducing the need for manual hardware configurations and updates.
- Improved operational efficiency: System configuration, maintenance, and management are simplified, and software development and deployment are accelerated.
- Faster deployment: Complex configuration tasks are simplified using scripts, which speeds up application release.
- Fewer operation risks: Version control is supported, reducing configuration errors.

This section describes how to create a function using Terraform.

### Prerequisites

**You have obtained an access key.**

For details about how to obtain an access key, see **Access Keys**. **You are advised to use temporary access keys (Access Key ID, Secret Access Key, and Token) to enhance security. For details about the temporary access keys, see Temporary Access Key (for Federated Users). For details about how to obtain them, see Obtaining Temporary Access Keys and Security Tokens of an IAM User.**

### Preparing a Terraform Environment

- Install Terraform.

  Terraform provides installation packages for different environments. For details, see the **Terraform official website**.

The following uses Linux CentOS (public access required) as an example to describe how to install Terraform.

Log in to the system as the **root** user, create the **/home/Terraform** directory, run the **cd** command to go to this directory, and then run the following commands:

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
sudo yum -y install terraform
```

- View basic Terraform commands.

  After Terraform commands are executed, command details are displayed. For details, see **Terraform CLI**.

- View basic Terraform syntax.

  The Terraform configuration language is based on the HCL syntax. It is easy to configure and readable, and is compatible with the JSON syntax. For details, see **Terraform official website**.

## Writing a Function Resource Script

Huawei Cloud has registered with Terraform as a provider. You can mount your functions to the provider as resources. For details, see **huaweicloud_fgs_function**.

The following is an example.

Create a **main.tf** file on the server, copy the following script to the file, and save it.

```
terraform {
  required_providers {
    huaweicloud = {
      source  = "huaweicloud/huaweicloud"
      version = ">= 1.40.0"
    }
  }
}

provider "huaweicloud" {
  region     = "cn-east-3" #Actual region
  access_key = "********"  #Use the obtained temporary AK.
  secret_key = "********" #Use the obtained temporary SK.
  security_token = "********" #Use the obtained temporary token.
}

resource "huaweicloud_fgs_function" "fgs_function" {
  name        = "test_func_rf"
  app         = "default"
  agency      = "function-admin"
  description = "function test"
  handler     = "index.handler"
  memory_size = 128
  timeout     = 3
  runtime     = "Python3.6"
  code_type   = "inline"
  func_code   =
"aW1wb3J0IGpzb24KZGVmIGhhbmRsZXIgKGV2ZW50LCBjb250ZXh0KToKICAgIG91dHB1dCA9ICdIZWxsbyBtZXNzYWdlOiAnICsganNvbi5kdW1wcyhldmVudCkKICAgIHJldHVybiBvdXRwdXQ="
}
```

## Creating a Function by Running Terraform Commands

1. Go to the file path and run the following command to initialize a working directory that contains Terraform code.

   ```
   terraform init
   ```

```
[root@function-deploy [redacted]# terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of local-registry/huaweicloud/huaweicloud from the dependency lock file
- Using previously-installed local-registry/huaweicloud/huaweicloud v1.45.1

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

2. Run the following command to apply the infrastructure configuration to the function and enter **yes** after **Enter a value:**.

```
terraform apply
```

```
[root@function-deploy [redacted]# terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # huaweicloud_fgs_function.fgs_function will be created
  + resource "huaweicloud_fgs_function" "fgs_function" {
      + agency                = "function-admin"
      + app                   = "default"
      + app_agency            = (known after apply)
      + code_filename         = (known after apply)
      + code_type             = "inline"
      + depend_list           = (known after apply)
      + description           = "fuction test"
      + enterprise_project_id = (known after apply)
      + func_code             = "d7a44bed8dcea78bdb5b8486f5e8ac18d59e8389"
      + functiongraph_version = (known after apply)
      + handler               = "index.handler"
      + id                    = (known after apply)
      + initializer_handler   = (known after apply)
      + initializer_timeout   = (known after apply)
      + memory_size           = 128
      + mount_user_group_id   = (known after apply)
      + mount_user_id         = (known after apply)
      + name                  = "zyl_test_func_rf"
      + region                = (known after apply)
      + runtime               = "Python3.6"
      + timeout               = 3
      + urn                   = (known after apply)
      + version               = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

huaweicloud_fgs_function.fgs_function: Creating...
huaweicloud_fgs_function.fgs_function: Creation complete after 1s [id=urn:fss:cn-east-3:09df01fba000f5e12ffac00733b3532d:function:default:zyl_test_func_rf:latest]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

3. If the execution is successful, the function has been created.

# 5 Configuring Functions

## 5.1 Overview

After a function is created, you need to configure the resources and environment required for running the function to ensure that the function meets service requirements.

### Function Configuration Process

Figure 5-1 describes the overall process of configuring a function.

**Figure 5-1** Function configuration process


For details, see Table 5-1.

**Table 5-1** Function configuration procedure

| No. | Procedure | Description |
| --- | --- | --- |
| 1 | Configuring Function Code | FunctionGraph supports the following function code deployment modes:<br>• **Edit code inline**: Edit function code inline using the console code editor.<br>• **Upload function code**: Upload a local ZIP or JAR file. This mode applies to scenarios where the code file size does not exceed 40 MB.<br>• **Upload code from OBS**: Upload a ZIP or JAR file in an OBS bucket by specifying the OBS bucket address. This mode applies to scenarios where the code file size does not exceed 300 MB. |

| No. | Procedure | Description |
| --- | --- | --- |
| 2 | **Configuring Dependencies** | Dependencies provide dependent libraries, runtime environments, and extended execution capabilities for functions to ensure that function code meets service requirements. |
| 3 | (Optional) **Configuring Agency Permissions** | If FunctionGraph needs to access other cloud services, you must create an agency to authorize FunctionGraph to access these cloud services. Otherwise, retain the default settings. |
| 4 | (Optional) **Configuring Network** | After a function is created, **Public Access** is enabled by default. You can manually modify the network configuration.<br>Functions can access the following two types of networks:<br>• **Public Access**: Functions can access public network resources. The scenarios include calling external APIs, accessing other cloud storage services, and integrating third-party services.<br>• **VPC Access**: Functions can access resources in VPCs. For services that require high bandwidth, high performance, and high reliability, you are advised to enable this feature. |
| 5 | (Optional) **Configuring Triggers** | A trigger triggers the execution of a function. You can create a trigger and define the trigger condition. When the condition is met, the event source automatically invokes the function associated with the trigger.<br>If FunctionGraph functions need to automatically respond to specific events, configure triggers. For details, see **Supported Trigger Events**. |
| 6 | **Debugging a Function Online** | After configuring the function, set test events to verify and debug the function execution. |

# 5.2 Configuring Function Code

## 5.2.1 Editing Function Code Inline

This section describes how to edit and deploy function code on the FunctionGraph console, and how to view and modify the function handler.

### Scenarios

FunctionGraph provides an SDK for editing code in Node.js, Python, PHP, and custom runtime. If your function code depends only on the SDK library, you can edit code using the inline editor on the **Code** tab page of the FunctionGraph

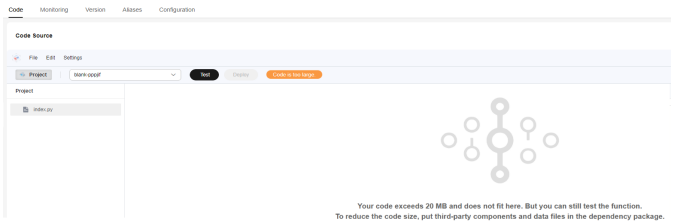
console. If you have developed function code locally, upload the function code by referring to **Uploading Function Code**.

In the inline code editing area, you can manage files and folders in the same way as managing projects. For details, see **Editing Function Code Inline**.

## Notes and Constraints

- Only Node.js, Python, PHP, and custom runtime functions support inline code editing.
- If the code deployed in the inline code editor is greater than 20 MB, the code is not displayed, as shown in **Figure 5-2**. However, the function code can still be tested.

**Figure 5-2** Code not displayed in the editor




## Editing Function Code Inline

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

**Step 2** Click a function name to go to the function details page.

**Step 3** On the **Code** tab page, edit the function code inline.

**Step 4** After editing the code, click **Deploy**. The console compresses the code and related configurations into a code file that can run on FunctionGraph. No other operations are required.

If the code is modified, click **Deploy** again.

**----End**

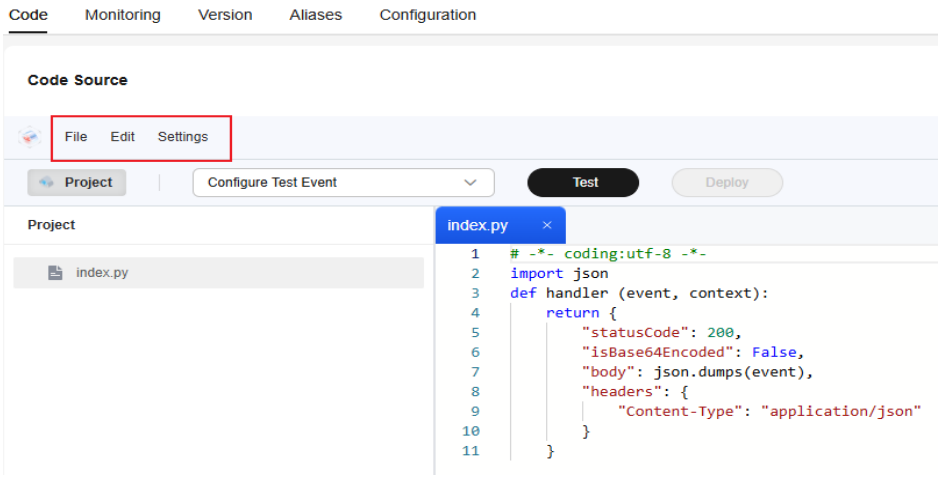
**Inline code editing area management:**

The inline code editing area provides project-based code file management capabilities as shown in **Figure 5-3**. You can create files and folders, and edit and set code in the code box.

- **File**: You can create files and folders, and save and close all files.
- **Edit**: Undo/redo typing; cut, copy, and paste code; find and replace content.
- **Settings**: You can set the font size of the code, the theme color of the code box, and whether to automatically format the code.

**Figure 5-3** Code area management




## Modifying the Function Handler

Function handler is the entry point specified in the function code for executing logic. Generally, it is a specific function or method. When a function is triggered, the code is executed from the handler.

For details about the naming rules of function handlers in different runtimes, see **Table 5-2**.

**Table 5-2** Rules for configuring the function handler

| Runtime | Handler Rule | Example |
| --- | --- | --- |
| Node.js | [File name].[Handler name] | myfunction.handler |
| Python | | |
| PHP | | |
| Java | [Package name].[Class name].[Execution function name] | com.xxxxx.exp.Myfunction.myHandler |
| Go | The handler name must be the same as that of the executable file in the code file. | handler |
| C# | [Assembly name]::[Namespace].[Class name]::[Execution function name] | CsharpDemo::CsharpDemo.Program::MyFunc |

The following uses a Python function as an example to describe how to view and modify the function handler on the function details page.

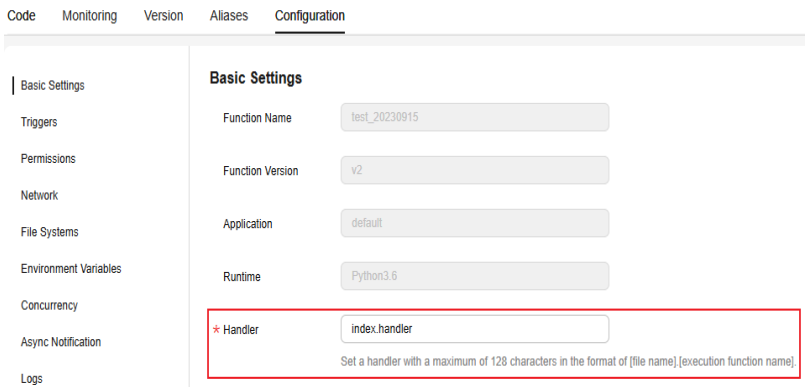
  

**Step 1** Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click a function name to go to the function details page.

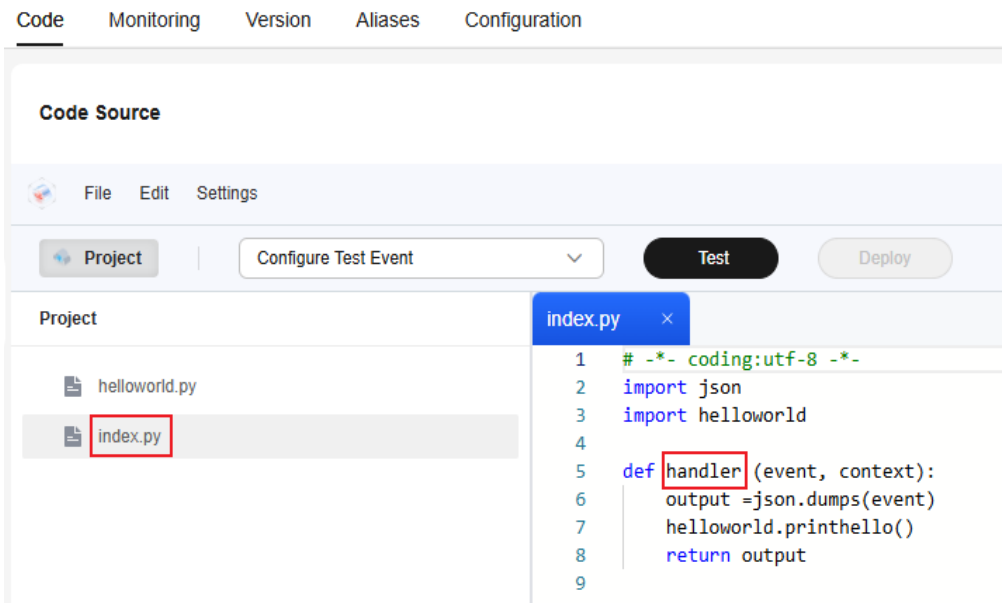
**Step 3** Choose **Configuration** > **Basic Settings**. View and modify the handler information as shown in **Figure 5-4**. After the modification, click **Save**.

**Figure 5-4** Handler




**Step 4** On the **Code** tab page, modify the file name and function name based on the handler, as shown in **Figure 5-5**.

**Figure 5-5** File name and execution function name




**----End**

## Helpful Links

- Manage function code using APIs. For details, see **Function Lifecycle Management APIs**.

- For details about how to view and configure the basic information of a function, see **Basic Settings**.

## 5.2.2 Uploading Function Code

This section describes how to upload a function code file on the FunctionGraph console for deployment.

### Scenarios

If the function code file is no more than 40 MB, you can upload a local ZIP or JAR file on the FunctionGraph console. For details about the types of code files that can be directly uploaded, see **Table 5-3**. For details about how to package code files, see **Function Project Packaging Rules**.

**Table 5-3** File types that can be directly uploaded

| Runtime | Uploading a ZIP File | Uploading a JAR File | Description |
| --- | --- | --- | --- |
| Node.js | Supported | Not supported | - |
| Python | Supported | Not supported | When you write code in Python, do not name your dependency package with the same suffix as a standard Python library, such as **json**, **lib**, and **os**. Otherwise, an error indicating a module loading failure will be reported. |
| Java | Supported | Supported | You can upload a ZIP or JAR file of Java code.<br>**Uploading a JAR file**<br>• If no dependency is introduced, you can directly upload the JAR file of the function code.<br>• If dependencies are introduced, you can configure them after creating a function and then upload the JAR file of the function code.<br>**Uploading a ZIP file**<br>If dependencies are introduced, you can create a ZIP file that contains all dependencies and the JAR file of the function code and upload the ZIP file. For details, see **Developing Functions in Java (Using an IDEA Java Project)** and **Developing Functions in Java (Using an IDEA Maven Project)**. |

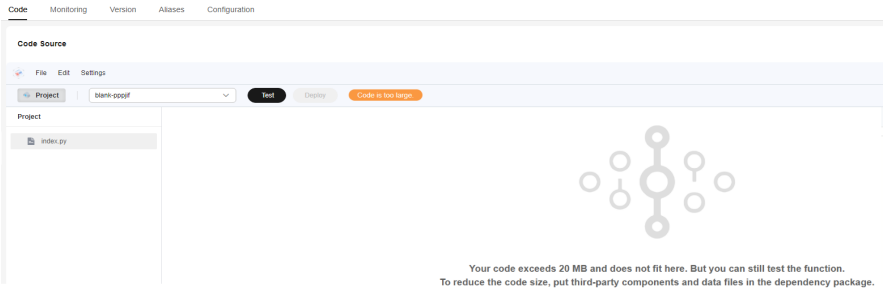
  

| Runtime | Uploading a ZIP File | Uploading a JAR File | Description |
| --- | --- | --- | --- |
| Go | Supported | Not supported | Ensure that the name of the dynamic library file is consistent with the plug-in name of the handler. For example, if the name of the dynamic library file is **testplugin.so**, set the handler name to **testplugin.Handler**. |
| C# | Supported | Not supported | - |
| PHP | Supported | Not supported | - |
| Custom | Supported | Not supported | - |
| Cangjie | Supported | Not supported | - |

## Notes and Constraints

- You are advised to set the encoding format of the code file to be uploaded to UTF-8.
- The code file to be uploaded cannot exceed 40 MB. If the file size exceeds 40 MB, upload it by following the instructions provided in **Uploading Function Code from OBS**.
- If the code to be uploaded contains sensitive information (such as account passwords), encrypt the sensitive information to prevent leakage.
- If the code deployed in the inline code editor is greater than 20 MB, the code is not displayed, as shown in **Figure 5-6**. However, the function code can still be tested.

**Figure 5-6** Code not displayed in the editor




## Uploading Function Code

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

**Step 2** Click a function name to go to the function details page.

**Step 3** Click the **Code** tab and choose **Upload** > **Local ZIP** or **Upload** > **Local JAR**.

**Step 4** In the dialog box that is displayed, click **Add** and select the function code file.

**Step 5** Use KMS static encryption to encrypt function code as required. (This feature is optional and available only in the **LA-Sao Paulo1** region.)

- **functiongraph/default (default)**: The function automatically creates a default key in DEW.

  If you use the default key for encryption and decryption for the first time, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:create, kms:cmk:get, and kms:cmk:list.

- **Customer master key (CMK)**: You can select a created key to encrypt the function code. For details about how to create a customer master key, see **Creating a Custom Key**.

  To use customer master key, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:get and kms:cmk:list.

**CAUTION**

If you select **Customer master key (CMK)**, do not delete the CMK used for function encryption in DEW. Otherwise, the function execution will fail because encrypted data cannot be decrypted.

**NOTE**

Go to the IAM console and add the **agency permission policy** (using the default key as an example) to the function agency by referring to **Configuring a Custom Policy in JSON View**. Then, configure the agency permission for the function by referring to **Configuring an Agency**.

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "kms:dek:decrypt",
                "kms:dek:create",
                "kms:cmk:create",
                "kms:cmk:get",
                "kms:cmk:list"
            ],
            "Effect": "Allow"
        }
    ]
}
```

**Step 6** After the file is uploaded, click **OK**. The code is automatically deployed to the code editing area.

If you need to modify the code in the code editing area, click **Deploy** each time after the modification is complete. For details about how to modify the function handler, see **Modifying the Function Handler**.

**----End**

# 5.2.3 Uploading Function Code from OBS

This section describes how to upload a function code file stored in OBS to the FunctionGraph console.

## Scenarios

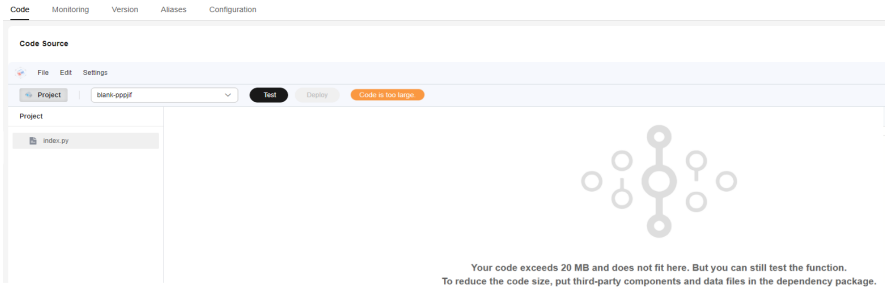
On the FunctionGraph console, you can upload a ZIP file (up to 300 MB) in an OBS bucket by specifying its OBS link URL to deploy your function code.

All runtimes support function code upload from OBS. For details about how to package a function project, see **Function Project Packaging Rules**.

## Notes and Constraints

- Ensure that the OBS bucket is in the same region as the function.
- You are advised to set the encoding format of the code file uploaded to the OBS bucket to UTF-8.
- If the size of the code file in the OBS bucket exceeds 300 MB or the size of the source code after decompression exceeds 1.5 GB, **submit a service ticket**.
- If the code deployed in the inline code editor is greater than 20 MB, the code is not displayed, as shown in **Figure 5-7**. However, the function code can still be tested.

**Figure 5-7** Code not displayed in the editor




## Prerequisites

You have uploaded the function code file to an OBS bucket and granted the read permission on the object to anonymous users. For details, see **Granting All Accounts the Read Permission for Certain Objects**.

## Uploading Function Code

**Step 1** Log in to the **OBS console**. Copy the URL of the code file object by referring to **Sharing Objects with Anonymous Users Using URLs**.

**Step 2** Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.

**Step 3** Click a function name to go to the function details page.

**Step 4** Click the **Code** tab and choose **Upload** > **OBS ZIP**.

**Step 5** In the displayed dialog box, enter the copied OBS bucket object URL.

**Step 6** Use KMS static encryption to encrypt function code as required. (This feature is optional and available only in the **LA-Sao Paulo1** region.)

- **functiongraph/default (default)**: The function automatically creates a default key in DEW.

  If you use the default key for encryption and decryption for the first time, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:create, kms:cmk:get, and kms:cmk:list.

- **Customer master key (CMK)**: You can select a created key to encrypt the function code. For details about how to create a customer master key, see **Creating a Custom Key**.

  To use customer master key, ensure that the function agency has the following permissions: kms:dek:decrypt, kms:dek:create, kms:cmk:get and kms:cmk:list.

**CAUTION**

If you select **Customer master key (CMK)**, do not delete the CMK used for function encryption in DEW. Otherwise, the function execution will fail because encrypted data cannot be decrypted.

**NOTE**

Go to the IAM console and add the **agency permission policy** (using the default key as an example) to the function agency by referring to **Configuring a Custom Policy in JSON View**. Then, configure the agency permission for the function by referring to **Configuring an Agency**.

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "kms:dek:decrypt",
                "kms:dek:create",
                "kms:cmk:create",
                "kms:cmk:get",
                "kms:cmk:list"
            ],
            "Effect": "Allow"
        }
    ]
}
```

**Step 7** After the OBS address is entered, click **OK**. The code is automatically deployed to the code editing area.

If you need to modify the code in the code editing area, click **Deploy** each time after the modification is complete. For details about how to modify the function handler, see **Modifying the Function Handler**.

**----End**

# 5.3 Configuring Dependencies

## 5.3.1 Overview

### Introduction

A dependency contains public libraries that support the running of function code. You can encapsulate the required public libraries into a dependency for easier management, sharing, and smaller deployment sizes.

You can keep multiple versions of the same dependency for systematic management.

Function dependencies are classified into public dependencies and private dependencies. For details, see **Configuring a Dependency for a Function**.

### Public Dependencies

Public dependencies are built in FunctionGraph and can be added to your function code to implement service logic.

Public dependencies have the following advantages over private dependencies:

- Out-of-the-box: You can directly add dependencies on the function details page and focus on code and service logic.
- Reduced latency: Public dependencies are cached on the platform, and they do not need to be obtained from storage services during cold starts.
- Simplified operations: The maximum size of a private dependency is 300 MB. For dependencies greater than 300 MB, you need to split and upload them for multiple times. Public dependencies are not subject to the 300 MB limit.

### Private Dependencies

A private dependency is created by encapsulating common libraries required by service code on your local PC. For details about how to create private dependencies of each runtime, see **Creating a Private Dependency for a Function**.

### Supported Dependency Libraries

FunctionGraph allows you to import standard libraries and some non-standard libraries to your code.

- Standard libraries

  When using standard libraries, you can import them to your inline code or package and upload them to FunctionGraph.

- Built-in non-standard libraries

  FunctionGraph has some non-standard libraries built in, as shown in **Table 5-4** and **Table 5-5**. The method of using the non-standard libraries is the same as that of using standard libraries.

**Table 5-4** Third-party components integrated with the Node.js runtime

| Name | Usage | Version |
| --- | --- | --- |
| q | Asynchronous method encapsulation | 1.5.1 |
| co | Asynchronous process control | 4.6.0 |
| lodash | Common tool and method library | 4.17.10 |
| esdk-obs-nodejs | OBS SDK | 2.1.5 |
| express | Simplified web-based application development framework | 4.16.4 |
| fgs-express | Provides a Node.js application framework for FunctionGraph and APIG to run serverless applications and REST APIs. This component provides an example of using the Express framework to build serverless web applications or services and RESTful APIs. | 1.0.1 |
| request | Simplifies HTTP invocation and supports HTTPS and redirection. | 2.88.0 |

The following is an example of introducing a dependency library in function code using the Node.js runtime:

```
const ObsClient = require('esdk-obs-nodejs');
```

**Table 5-5** Non-standard libraries supported by the Python runtime

| Library | Usage | Version |
| --- | --- | --- |
| dateutil | Advanced library for handling dates and times, supporting date parsing, formatting, and time zone operations. | 2.6.0 |
| requests | HTTP library, which is used to send HTTP requests and process responses. | 2.7.0 |

| Library | Usage | Version |
| --- | --- | --- |
| httplib2 | HTTP client library, which supports HTTP/HTTPS requests, authentication, and proxies. | 0.10.3 |
| numpy | Library for scientific computing, which supports efficient array operations and mathematical functions. | Python 2.7, numpy==1.16.6<br>Python 3.6, numpy==1.18.5<br>Python 3.9, numpy==1.18.5<br>Python 3.10, numpy==1.24.2 |
| redis | Library for connecting to and operating Redis databases, which supports multiple data structures. | 2.10.5 |
| ObsClient | Python client of OBS, which is used to manage object storage resources. | 3.0.3 |
| smnsdk | Python SDK of SMN, which is used to access the SMN service. | 1.0.1 |

- Other third-party dependency libraries

  Compress the dependent third-party libraries into a ZIP file, upload the ZIP file to an OBS bucket or upload it on the function console. On the **Code** tab page, add the dependencies so that they can be used in function code. For details, see **Configuring a Dependency for a Function**.

### Helpful Links

Manage function dependencies through APIs. For details, see **Dependency APIs**.

## 5.3.2 Creating a Private Dependency for a Function

This section describes how to create a private dependency of a function on your local PC.

### Notes and Constraints

- The maximum size of a dependency is 300 MB, and the maximum number of files in a dependency is 30,000.
- The name of a file in a dependency cannot end with a tilde (~), for example, **module~**.

- If the modules to be installed need dependencies such as **.dll**, **.so**, and **.a**, archive them to a **.zip** package.
- **You are advised to create function dependencies in Huawei Cloud EulerOS 2.0**. If other OSs are used, the dynamic link library may not be found due to the differences between underlying dependency libraries.

## Setting Up the EulerOS Environment

You are advised to create function dependencies in EulerOS. EulerOS is an enterprise-grade Linux OS based on open-source technology. It features high security, scalability, and performance, meeting customers' requirements for IT infrastructure and cloud computing services.

You can set up the **Huawei Cloud EulerOS** environment using the following methods:

- Buy a EulerOS ECS on Huawei Cloud by referring to **Purchasing and Logging In to a Linux ECS**. On the **Configure Basic Settings** page, select **Public Image**, and select **Huawei Cloud EulerOS** and an image version.
- Download the EulerOS image, and use virtualization software to set up the EulerOS VM on a local PC.

## Creating a Function Dependency

The following describes how to create dependencies for different runtime functions.

## Creating a Dependency for a Node.js Function

Before creating a dependency, ensure that Node.js matching the function runtime has been installed in the environment. The following uses Node.js 8.10 as an example to describe how to create a MySQL dependency.

Do not run the **CNPM** command to generate Node.js dependencies.

**Step 1** Run the following command to install the MySQL dependency.

```
npm install mysql --save
```

The **node_modules** folder is generated under the current directory.

**Step 2** Run the following command to package the **node_modules** folder into a ZIP package, that is, the dependency.

```
zip -rq mysql-node8.10.zip node_modules
```

**----End**

To package multiple dependencies, follow the steps below:

**Step 1** Create a **package.json** file with the following content:

```
{
   "name": "test",                 //Customized dependency name
   "version": "1.0.0",             //Dependency version
   "dependencies": {
      "redis": "~2.8.0",
      "mysql": "~2.17.1"
   }
}
```

**Step 2** Run the following command in the directory where the package.json file is stored:

```
npm install --save
```

**Step 3** Compress **node_modules** into a ZIP package. This generates a dependency that contains both MySQL and Redis.

```
zip -rq node8.10.zip node_modules
```

**----End**

## Creating a Dependency for a Python Function

Before creating a dependency, ensure that Python matching the function runtime has been installed in the environment.

The following uses Python 3.12 as an example to describe how to create a PyMySQL dependency.

**Step 1** Run the following command to install the PyMySQL dependency to the local **/tmp/pymysql** directory.

```
pip install PyMySQL --root /tmp/pymysql
```

**Step 2** Run the following command to access the specified directory.

```
cd /tmp/pymysql/
```

**Step 3** Go to the **site-packages** directory (generally **lib/python3.12/site-packages/**). If no dependency file exists in this directory, run the **find** command to find the file and go to its directory. Then run the following command to compress the dependency file.

The required dependency is generated.

```
zip -rq pymysql.zip *
```

**----End**

NOTE

To install the local wheel installation package, run the following commands:

```
pip install piexif-1.1.0b0-py2.py3-none-any.whl --root /tmp/piexif
//Replace piexif-1.1.0b0-py2.py3-none-any.whl with the actual installation package name.
```

## Creating a Dependency for a PHP Function

Before creating a dependency, ensure that PHP matching the function runtime has been installed in the environment. The following uses PHP 7.3 as an example to describe how to install the protobuf3.19 dependency using Composer. Composer has been installed in the default environment. The procedure is the same for other PHP versions.

**Step 1** Create the **composer.json** file with the following content:

```
{
   "require": {
      "google/protobuf": "^3.19"
   }
}
```

**Step 2** Run the following command:

```
Composer install
```

The **vendor** folder is generated with the **autoload.php**, **composer**, and **google** subfolders in the current directory.

**Step 3** Run the following command to generate a ZIP package:

```
zip –rq vendor.zip vendor
```

**----End**

If multiple dependencies need to be packaged, specify them in the **composer.json** file, compress the **vendor** folder into a ZIP file and upload it.

NOTE

In PHP projects, third-party dependencies downloaded using Composer need to be loaded using **require "./vendor/autoload.php"**. By default, FunctionGraph stores the decompressed files in a directory at the same level as the project code directory.

### Creating a Dependency for a Java Function

When you compile a function using Java, dependencies need to be compiled locally. For details about how to create a dependency, see **Developing Functions in Java (Using an IDEA Java Project)**.

### Creating a Dependency for a Go Function

When you compile a function using Go, dependencies need to be compiled locally.

### Creating a Dependency for a C# Function

When you compile a function using C#, dependencies need to be compiled locally.

### Helpful Links

- After creating a dependency locally, create and configure the dependency on the FunctionGraph console by referring to **Configuring a Dependency for a Function**.
- Manage function dependencies through APIs. For details, see **Dependency APIs**.

## 5.3.3 Configuring a Dependency for a Function

This section describes how to create, configure, and delete a dependency on the FunctionGraph console.

### Scenarios

Dependencies provide libraries, runtime environments, and extension capabilities for functions. For details about dependencies, see **Overview**.

Private dependencies must be created on the FunctionGraph console before being used. Public dependencies can be used directly. To use standard libraries and some built-in non-standard libraries in your function code, refer to **Supported Dependency Libraries**.

## Notes and Constraints

- You can add a maximum of 20 dependencies for a function.
- Dependencies in use cannot be deleted.
- Do not use the same directory or file name for dependencies and code files. For example, if **depends.zip** contains **index.py** and you use the inline editor with the handler **index.handler**, FunctionGraph will create another **index.py** file. This may cause file overwriting or merging errors.

## Prerequisites

Before configuring a private dependency for a function, create one locally. For details, see **Creating a Private Dependency for a Function**.

## Creating a Dependency

**The following procedure applies only to private dependencies. For public dependencies, you can perform operations in Configuring Dependencies.**

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

**Step 2** Click **Create Dependency**.

**Step 3** Set the dependency parameters by referring to **Table 5-6**.

**Table 5-6** Dependency configuration parameters

| Parameter | Description |
| --- | --- |
| Name | 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 | Runtime language of the dependency.<br>You can select Cangjie, Node.js, Python, Java, PHP, Go, C#, or Custom. |
| Code Entry Mode | You can upload a ZIP file, write code online, 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**: Enter the OBS link of the ZIP file in the OBS bucket. Note that the OBS bucket must be in the same region as the function. 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**. |
| Description | Enter a description of the dependency. |

**Step 4** Click **Create**. The dependency details page is displayed. By default, a new dependency is version **1**.

**----End**

**Creating a version for a dependency**

On the dependency details page, view all versions and version information of the current dependency, as shown in **Figure 5-8**. Each dependency can have multiple versions.

- To create a dependency version, click **Create Version** in the upper right corner of the page. The new version number is the current version number plus 1. After the creation is successful, the latest dependency version is displayed on the page by default.
- Click a version number in the **Version** column to view the version information.
- Click **Download** or **Delete** in the row where the version number is located to download or delete the version.

**Figure 5-8** Dependency version management

## Configuring Dependencies

If the private dependency is large, you are advised to choose **Configuration** > **Basic Settings** on the function details page and extend the execution timeout to prevent function execution failures caused by timeout.

**Procedure**

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

**Step 2** Click the name of the desired function.

**Step 3** On the displayed function details page, click the **Code** tab, click **Add** in the **Dependencies** area.

**Step 4** Select the dependency by referring to **Table 5-7** and click **OK**.

**Table 5-7** Dependency configuration

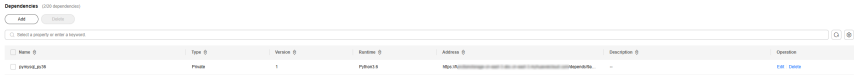
| Parameter | Description |
| --- | --- |
| Runtime | Runtime of this function. It cannot be changed. |
| Type | Add a **Public** or **Private** dependency.<br>• **Public**: built-in dependency provided by FunctionGraph.<br>• **Private**: dependency uploaded in **Creating a Dependency**. |
| Name | Select a dependency. |

| Parameter | Description |
| --- | --- |
| Version | Select a version to be added. |

**Step 5** Figure 5-9 shows the added dependency.

**Figure 5-9** Adding a dependency



**----End**

## Deleting a Dependency

To delete a dependency, just delete all of its versions.

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

**Step 2** Click the dependency name. The dependency details page is displayed.

**Step 3** As shown in Figure 5-10, click **Delete** in the row where the dependency version number is located under **Versions** to delete the version.

**Figure 5-10** Deleting a dependency version

**----End**

## Helpful Links

Manage function dependencies through APIs. For details, see **Dependency APIs**.

# 5.4 Configuring Agency Permissions

This section describes how to create, configure, and modify agency permissions for functions.

## Scenarios

Create an agency to authorize FunctionGraph to access other cloud services, and configure agency permissions for functions.

For details about how to configure agency permissions for common function scenarios, see **Common Permissions**.

## Default Agency

If you do not have any agency in your Huawei Cloud account, FunctionGraph provides you with a default one.

The default agency contains some of the cloud resource permissions required by FunctionGraph, as shown in **Table 5-8**. For details about the **fine-grained minimum permissions** of related services, see **Table 5-11**.

**Table 5-8** Default agency permissions

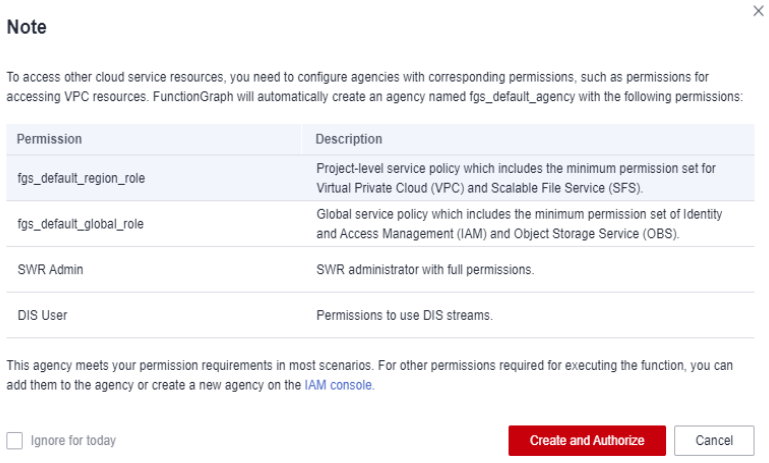
| Permission | Description |
| --- | --- |
| fgs_default_region_role | Project-level service policy which includes the minimum permission set for Virtual Private Cloud (VPC) and Scalable File Service (SFS). |
| fgs_default_global_role | Global service policy which includes the minimum permission set of Identity and Access Management (IAM) and Object Storage Service (OBS). |
| SWR Admin | SoftWare Repository for Container (SWR) administrator with full permissions. |
| DIS User | Permissions to use Data Ingestion Service (DIS) streams. |

**The following are the three ways to create a default agency:**

Method 1: Pop-up box

- When you log in to the FunctionGraph console for the first time, no function and default agency is available. On the **Dashboard** page, a dialog box is displayed, asking you whether to create a default agency. Click **Create and Authorize**. The default agency named **fgs_default_agency** is automatically created.

**Figure 5-11** Pop-up box

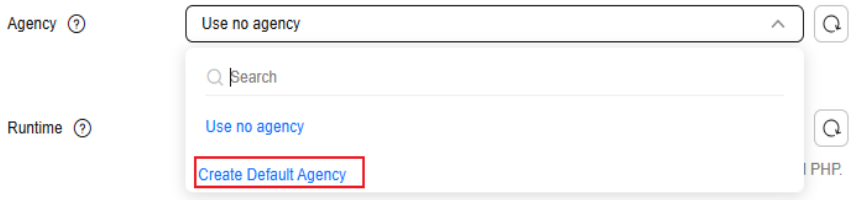



Method 2: Creating a default agency when creating a function

- When creating a function, select **Create Default Agency** from the **Agency** drop-down list as shown in **Figure 5-12**. Then the **Authorization** dialog box is displayed. Click **OK** to create a default agency named **fgs_default_agency**.

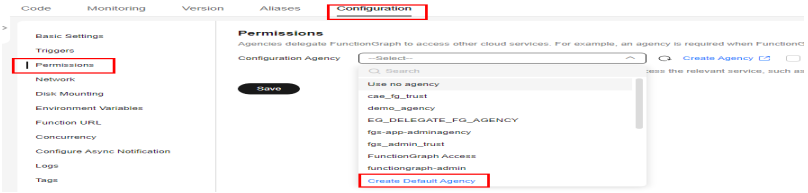
**Figure 5-12** Creating a default agency when creating a function




Method 3: Creating a default agency in Permissions

- Choose **Configuration** > **Permissions** on the function details page, and select **Create Default Agency** as shown in **Figure 5-13**. In the displayed dialog box, click **OK** to create a default agency named **fgs_default_agency**.

**Figure 5-13** Creating a default agency in Permissions




**NOTE**

After a default agency is created, you will not be prompted to do so again.

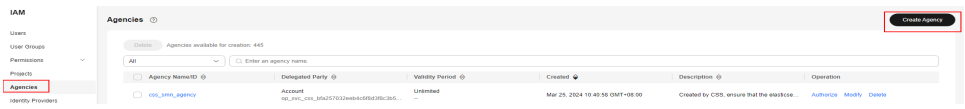
## Creating a Function Agency

To configure a function to access resources in a VPC, you need to create an agency for FunctionGraph and grant the agency the permission to access VPC.

**Step 1** Log in to the **IAM console**.

**Step 2** In the navigation pane, choose **Agencies** and click **Create Agency** in the upper right corner.

**Figure 5-14** Creating an agency



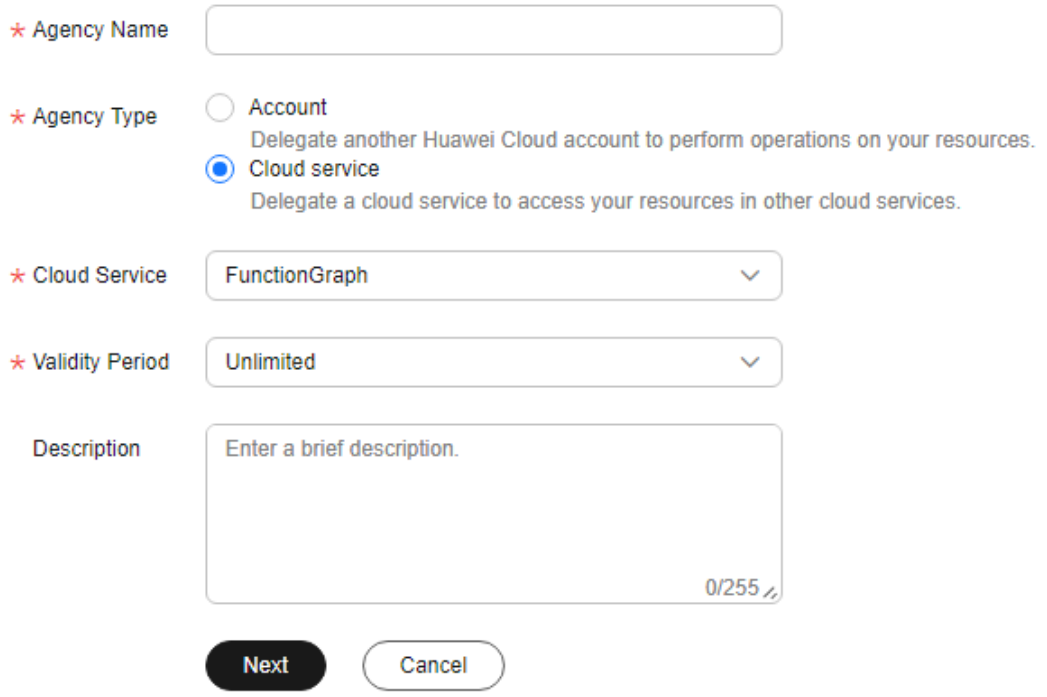

**Step 3** Configure agency parameters. After the parameters are configured, as shown in **Figure 5-15**, click **OK**. The system displays a message indicating that the creation is successful. Click **Authorize**.

- **Agency Name**: Enter a name, for example, **serverless-trust**.
- **Agency Type**: Select **Cloud service**.

- **Cloud Service**: Select **FunctionGraph**.
- **Validity Period**: Select **Unlimited**. You can set this parameter based on service requirements.
- **Description** (Optional): Enter a description.

**Figure 5-15** Setting basic information




**Step 4** On the **Select Policy/Role** page, search for the required permissions in the search box on the right and select them. The following uses **VPC Administrator** as an example. **Figure 5-16** shows the details. Click **Next**.

**Figure 5-16** Selecting policies

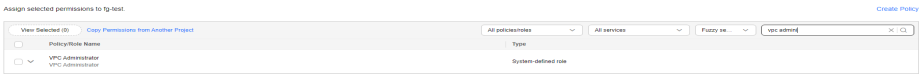


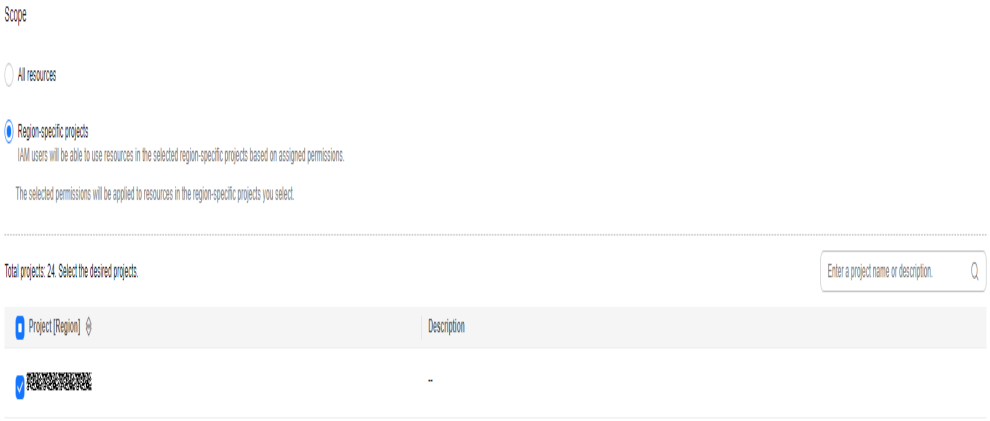
**Table 5-9** Example of agency permissions

| Policy Name | Scenario |
| --- | --- |
| VPC Administrator | VPC administrator |

**Step 5** On the **Select Scope** page, select **Region-specific projects** and the required region as shown in **Figure 5-17**. Click **OK**. The authorization success page is displayed.

**Figure 5-17** Selecting the required permissions




**----End**

NOTE

If the default policies do not meet your requirements, you can create custom policies in the visual editor or JSON view, and attach custom policies to user groups for refined access control. For details, see **Creating Custom Policies**.

## Configuring an Agency

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

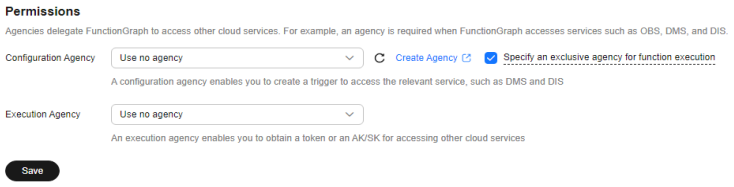
**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Permissions** and configure an agency by referring to **Table 5-10**. Click **Save**.

**Table 5-10** Agency configuration parameters

| Parameter | Description |
| --- | --- |
| Agency | Select the created agency from the drop-down list. If no agency is available, click **Create Agency** on the right to go to the IAM page. For details, see **Creating a Function Agency**. |

| Parameter | Description |
| --- | --- |
| Execution Agency | This parameter is displayed when you select **Specify an exclusive agency for function execution** as shown in **Figure 5-18**. After the configuration, you can obtain the token or SecurityAccessKey, SecuritySecretKey, and SecurityToken with the permissions through the **context** parameter in the function handler method to access other cloud services. For details about the code example, see **6.4-Application Example of Environment Variables**.<br>The execution agency can be configured independently to ensure clear management of agency permissions.<br>**Figure 5-18** Setting agencies<br>  |

**----End**

## Modifying a Function Agency

To modify the permissions, validity period, and description of an agency, you need to modify the corresponding FunctionGraph agency on the **IAM console** as shown in **Figure 5-19**.

**Figure 5-19** Modifying a function agency

After an agency is modified, it takes about 10 minutes for the modification (for example, **context.getSecurityToken()**) to take effect. The agency information obtained using the **context** method is valid for 24 hours. Refresh it before it expires.

## Common Permissions

To use the scenarios specified by **Table 5-11**, that is, to work with other services, create and configure an agency. For details, see **Creating a Function Agency** and **Configuring an Agency**.

When creating an agency, you need to adjust the granted permission type based on the actual service requirements. **In the production environment, you are advised to adjust the permission type to the fine-grained minimum permission** to ensure that the service running requirements are met and the potential risks of excessive permissions are effectively reduced.

**Table 5-11** Common permissions

| Scenario | Policy Name | Description | Fine-Grained Minimum Permission |
| --- | --- | --- | --- |
| Using a custom image | SWR Admin | SoftWare Repository for Container (SWR) administrator with full permissions.<br>For how to create a custom image, see **Creating a Function with an Image**. | Unavailable |
| Mounting an SFS Turbo file system | SFS Turbo ReadOnlyAccess | Read-only permissions for SFS Turbo.<br>For details about how to mount an SFS Turbo file system, see **Mounting an SFS Turbo file system**. | • sfsturbo:shares:getShare (Query details about a file system)<br>• sfsturbo:shares:showFsDir (Check whether a directory exists) |
| Mounting an ECS shared directory | ECS ReadOnlyAccess | Read-only permissions for ECS.<br>For details about how to mount an ECS shared directory, see **Mounting an ECS Shared Directory**. | ecs:cloudServers:get (Query details about an ECS) |
| Configuring a reserved instance policy | AOM ReadOnlyAccess | Read-only permissions for AOM. | • aom:metric:get (Query a metric)<br>• aom:metric:list (Query metric list) |
| | FunctionGraph ReadOnlyAccess | This policy grants read-only permissions for FunctionGraph. | functiongraph:function:getConfig (Query function configurations.) |
| Using a DIS trigger | DIS Administrator | Administrator who has all permissions for the DIS service.<br>For details about how to create a DIS trigger, see **Data Ingestion Service (DIS) Trigger**. | Unavailable |
| Using DMS triggers | DMS ReadOnlyAccess | Read-only permissions for DMS | dms:instance:get (Query instance details) |

| Scenario | Policy Name | Description | Fine-Grained Minimum Permission |
| --- | --- | --- | --- |
| Configuring cross-domain VPC access | VPC Administrator | VPC administrator has all permissions on all resources in the VPC.<br>For details about how to configure cross-domain VPC access, see **Configuring VPC Access**. | • vpc:ports:get (Query a port)<br>• vpc:ports:create (Create a port)<br>• vpc:vpcs:get (Query a VPC)<br>• vpc:subnets:get (Query a subnet)<br>• vpc:vips:delete (Unbind a virtual IP address from a VM)<br>• vpc:securityGroups:get (Query security groups or details about a security group) |
| DNS Resolution | DNS ReadOnlyAccess | Read-only permissions for DNS. Users granted these permissions can only view DNS resources.<br>For details about how to call the DNS API to resolve private domain names, see **How Does FunctionGraph Resolve a Private DNS Domain Name?** | • dns:recordset:get (Query a record set)<br>• dns:zone:get (Query a tenant zone)<br>• dns:recordset:list (Query record set list)<br>• dns:zone:list (Query the zone list) |

| Scenario | Policy Name | Description | Fine-Grained Minimum Permission |
| --- | --- | --- | --- |
| Configuring asynchronous notification | If the target service is OBS:<br>OBS Administrator | OBS administrator has all permissions for OBS.<br>For details about how to configure asynchronous notification, see **Asynchronous Notification Policy**. | • obs:bucket:HeadBucket (Obtain bucket metadata)<br>• obs:bucket:CreateBucket (Create a bucket)<br>• obs:object:PutObject (Upload objects using PUT method, upload objects using POST method, copy objects, append an object, initialize a multipart task, upload parts, and merge parts) |
| | If the target service is SMN:<br>SMN Administrator | SMN administrator has all permissions for SMN.<br>For details about how to configure asynchronous notification, see **Asynchronous Notification Policy**. | • smn:topic:publish (Publish a message)<br>• smn:topic:list (Query the topic list) |
| | If the target service is DIS:<br>DIS Administrator | Administrator who has all permissions for the DIS service.<br>For details about how to configure asynchronous notification, see **Asynchronous Notification Policy**. | Unavailable |

| Scenario | Policy Name | Description | Fine-Grained Minimum Permission |
| --- | --- | --- | --- |
| Using an OBS bucket | OBS Administrator | OBS administrator has all permissions for OBS. | • obs:bucket:GetBucketLocation (Query a bucket location)<br>• obs:bucket:ListAllMyBuckets (Query buckets)<br>• obs:bucket:GetBucketNotification (Obtain the event notification configuration of a bucket)<br>• obs:bucket:PutBucketNotification (Configure event notifications for a bucket) |

# 5.5 Configuring Networks

This section describes how to configure functions to access the public network or resources in a VPC.

## Scenarios

Table 5-12 describes the function network capabilities. You can configure them based on service requirements.

**Table 5-12** Network configuration description

| Parameter | Description |
| --- | --- |
| Public Access | You can configure functions to access the public network in either of the following ways:<br>• **Using the Default NIC**: After a function is created, you can use the default NIC to access the public network without manual configuration.<br>• **Using a Fixed VPC Public IP Address**: Enable **VPC Access**, configure a public NAT gateway in the VPC, and bind an EIP with exclusive bandwidth to it. |

| Parameter | Description |
| --- | --- |
| VPC Access | You can configure functions to access a VPC in either of the following ways:<br>• Accessing VPC Resources: Functions can access resources in the selected VPC. By default, public access is disabled. To access the public network, enable public access by referring to **Configuring a Fixed VPC Public IP Address**.<br>• **Invocation Only by Specific VPC**: This option allows the function to be invoked only from the specified VPC instead of the public network |

## Notes and Constraints

- You can bind a maximum of four subnets to all functions of an account.
- If VPC access is enabled, the default NIC is disabled and the NIC bound to the VPC will be used instead. Whether public access is supported depends on the VPC.

## Configuring Public Access

You can configure public access in either of the following ways:

## Using the Default NIC

After a function is created, **Public Access** is enabled by default, allowing the function to access the public network using the default NIC.

The public access bandwidth is shared among users, which is applicable to test scenarios. In the production environment, you are advised to configure a fixed VPC public IP address. For details, see **Configuring a Fixed VPC Public IP Address**.

## Configuring a Fixed VPC Public IP Address

To enable the function to access the public network in the production environment or set fixed public IP address (for whitelist verification), you can configure a public NAT gateway in the VPC and bind an EIP to the gateway. In this case, whether public access is supported depends on the VPC.

**Prerequisites**

1. You have created a VPC and a subnet according to **Creating a VPC**.
2. You have obtained an EIP according to **Assigning an EIP**.
3. You have configured the VPC access for the function. For details , see **Configuring VPC Access**.

**Procedure for creating a public NAT gateway:**

**Step 1** In the left navigation pane of the management console, choose **Network** > **NAT Gateway** to go to the NAT Gateway console. Then click **Buy Public NAT Gateway**.

**Step 2** On the displayed page, enter the required information by referring to **Buying a Public NAT Gateway**, and submit the configuration.

**Step 3** Click the public NAT gateway name. On the details page that is displayed, click **Add SNAT Rule**, set the rule, and click **OK**.

**----End**

## Configuring VPC Access

FunctionGraph allows functions to access resources in created VPCs or shared VPC subnets. For details about the shared VPC, see **Shared VPC**.

On the FunctionGraph console, you can configure VPC access as follows:

- Accessing VPC resources: Functions can access resources in the selected VPC. By default, public access is disabled. To access the public network, enable this feature and then configure public network access by referring to **Configuring a Fixed VPC Public IP Address**.
- **Invocation Only by Specific VPC**: This option allows the function to be invoked only from the specified VPC instead of the public network. It is suitable for scenarios where you need to strictly control the invocation source.

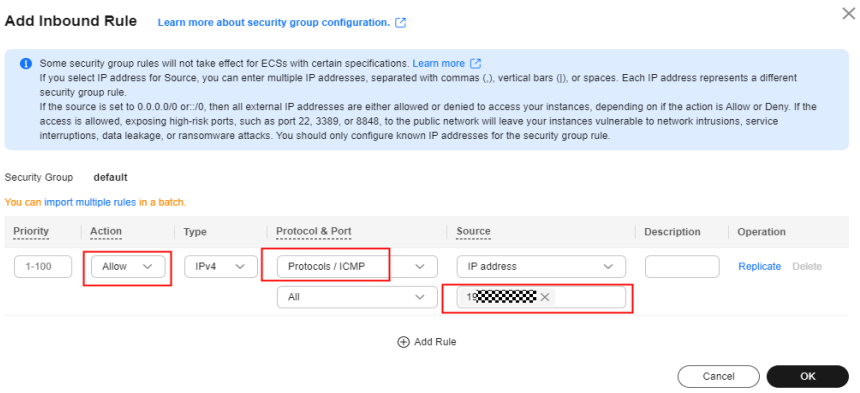
### Prerequisites

You have configured the inbound and outbound rules in the **default** security group as follows. For details, see **Adding a Security Group Rule**.

- Inbound rule: Set **Action** to **Allow**, **Protocol & Port** to **ICMP**, and the minimum range for **Source** to the VPC CIDR block selected for the function.

  For example, if the VPC CIDR block of the function is **192.168.*x.x*/24**, add an inbound rule with **Allow** for **Action**, **ICMP** for **Protocol & Port**, and **192.168.*x.x*/24** for **Source** as shown in **Figure 5-20**.

  **Figure 5-20** Inbound rule




- Outbound rule: Set **Action** to **Allow**.

### Configuring agency permissions

To enable VPC access, configure an agency with VPC permissions. For details, see **Configuring Agency Permissions**. The following agency permissions are involved:

- VPC Administrator: To use VPC, configure the **VPC Administrator** permission or grant the minimum permissions for VPC access by referring to **Table 5-13**.

**Table 5-13** Least permissions required

| Permission | Action |
| --- | --- |
| Deleting a port | vpc:ports:delete |
| Querying a port | vpc:ports:get |
| Creating a port | vpc:ports:create |
| Querying a VPC | vpc:vpcs:get |
| Querying a subnet | vpc:subnets:get |

- DNS ReadOnlyAccess: If a private domain name is configured in a VPC, assign the **DNS ReadOnlyAccess** permission to the function to resolve it.

**Enabling VPC access**

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

**Step 2** Click the name of the function to be configured. The function details page is displayed.

**Step 3** Choose **Configuration** > **Network**, enable **VPC Access** and set the parameters according to **Table 5-14**, as shown in **Figure 5-21**.

**The function uses the NIC bound to the configured VPC for network access and disables the default FunctionGraph NIC.**

**Figure 5-21** Configuring VPC access

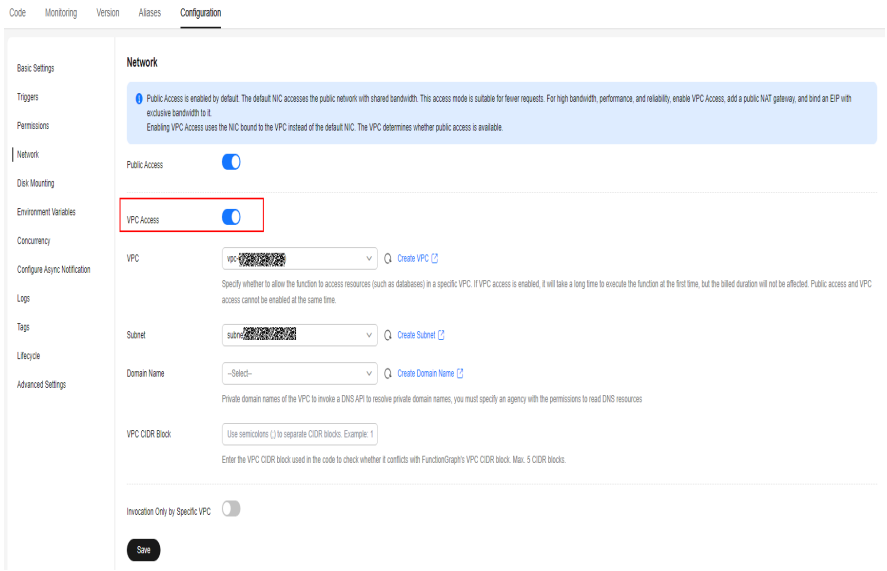


**Table 5-14** Network configuration parameters

| Parameter | Description |
| --- | --- |
| VPC | Mandatory.<br>Select the VPC to be accessed. For details on how to create a VPC and a subnet, see **Creating a VPC**. |
| Subnet | Mandatory.<br>Select a subnet of the VPC. |
| Domain Name | Optional. **This parameter is not supported in the AF-Johannesburg region**.<br>You can configure one or more private domain names of the VPC so that the function can use them to access resources in this VPC.<br>The following operations are related to domain name configuration. Use them as required.<br>• For details about how to create a private domain name, see **Creating a Private Zone**.<br>• For details about how to configure a function to implement domain name resolution, see . Functions can resolve only domain names of the A record set type. For details about how to add a record set, see **Record Set Types and Configuration Rules**.<br>• For details about how to access Redis in a VPC, see . |
| VPC CIDR block | Optional.<br>You can enter the VPC CIDR block used in the code to check whether it conflicts with FunctionGraph's VPC CIDR block. |
| IPv6 | Optional.<br>When you create the VPC, ensure that IPv6 is enabled for the default subnet. IPv6 will be automatically enabled here. For details, see **Creating a VPC with a Subnet**.<br>After the IPv6 function is enabled, the system automatically assigns an IPv6 CIDR block to the created subnet. Currently, the IPv6 CIDR block cannot be customized. **Once enabled, this function cannot be disabled.** For more information, see **IPv6 Network**. |

**----End**

### Shared VPC

Shared VPC is a mechanism based on Resource Access Manager (RAM). The owner of a VPC can share subnets in the VPC with other accounts to implement cross-tenant network resource sharing. For details, see **Shared VPC** in the *Virtual Private Cloud User Guide*.

You can configure a shared subnet in a function to access its resources. First you need to ensure that the subnet owner has configured subnet sharing for the function. For details, see **Sharing a Subnet with Other Accounts**. Then configure the shared subnet by referring to **Configuring VPC Access**. If the subnet owner cancels the sharing, the subnet cannot be accessed in the function.

# 5.6 Configuring Triggers

## 5.6.1 Overview

### Introduction

Triggers are key mechanisms in FunctionGraph for initiating function execution. When other cloud services detect specific events as event sources, triggers automatically invoke the associated function based on predefined rules, enabling real-time event processing. For supported trigger events, see **Supported Trigger Events**.

When events reach the function, they contain only event data relevant to that trigger. This structured JSON data is converted by FunctionGraph into objects usable by the function. A single function can be configured with one or multiple triggers according to service needs. Each trigger can independently invoke the function.

### Function Trigger Invocation Mode

Triggers can be invoked synchronously or asynchronously. For details about function invocation, see **Invoking the Function**.

- Synchronous invocation: After a client invokes a function, FunctionGraph executes the function immediately and returns a response and execution result to the client.
- Asynchronous invocation: After a client invokes a function, FunctionGraph queues the requests and directly returns responses to the client without waiting for the function execution result. FunctionGraph processes the queued requests one by one when the server is idle.

**Table 5-15** Function trigger invocation

| Trigger | Invocation Mode |
| --- | --- |
| API Gateway (dedicated) | Synchronous. You can change it to asynchronous. For details, see **Configuring Asynchronous Invocation**. |

| Trigger | Invocation Mode |
| --- | --- |
| API Connect (APIC) | Synchronous. You can change it to asynchronous. For details, see **Configuring Asynchronous Invocation**. |
| Timer | Asynchronous (default and cannot be changed) |
| GeminiDB DynamoDB | Synchronous (default and cannot be changed) |
| Cloud Trace Service (CTS) | Asynchronous (default and cannot be changed) |
| Document Database Service (DDS) | Asynchronous (default and cannot be changed) |
| Data Ingestion Service (DIS) | Asynchronous (default and cannot be changed) |
| DMS (for Kafka) | Asynchronous (default and cannot be changed) |
| Kafka (Open-Source) | Asynchronous (default and cannot be changed) |
| DMS (for RabbitMQ) | Asynchronous (default and cannot be changed) |
| GeminiDB Mongo | Asynchronous (default and cannot be changed) |
| Log Tank Service (LTS) | Asynchronous (default and cannot be changed) |
| Simple Message Notification (SMN) | Asynchronous (default and cannot be changed) |
| EventGrid (EG) | Asynchronous (default and cannot be changed) |

## Supported Trigger Events

**Table 5-16** lists the supported cloud services and trigger events for FunctionGraph. After an event source trigger is configured, the corresponding function is automatically invoked when an event is detected.

**Table 5-16** Cloud service events supported by FunctionGraph

| Cloud Service/ Feature | Trigger Event |
| --- | --- |
| Timer | You can schedule a timer (**timer example event**) to invoke function code based on a fixed rate of minutes, hours, or days or a cron expression.<br>For details, see **Using a Timer Trigger**. |

| Cloud Service/ Feature | Trigger Event |
| --- | --- |
| APIG | FunctionGraph functions are invoked over HTTP or HTTPS by defining REST APIs and endpoints on APIG. You can map each API operation (such as, GET and PUT) to a specific function. APIG invokes the relevant function when an HTTPS request (**APIG example event**) is sent to the API endpoint.<br>For details, see **Using an APIG (Dedicated) Trigger**. |
| APIC | With API operations such as GET and PUT mapped to specific functions, APIC can invoke corresponding functions to perform operations when receiving relevant HTTPS or HTTP requests.<br>For details, see **Using an APIC Trigger**. |
| DIS | DIS can ingest large amounts of data in real time. You can create a function to automatically poll a DIS stream and process all new data records, such as website click streams, financial transactions, social media streams, IT logs, and data tracking events (**DIS example event**). FunctionGraph periodically polls the stream for new data records.<br>For details, see **Using a DIS Trigger**. |
| DMS for Kafka | When a message is created in a Kafka topic, FunctionGraph consumes the message and triggers the function to perform other operations (**Kafka example event)**).<br>For details about how to use Kafka triggers, see:<br>• **Using a Kafka Trigger**<br>• **Using an Open-Source Kafka Trigger** |
| DMS for RabbitMQ | When a DMS for RabbitMQ trigger is used, FunctionGraph periodically polls for new messages in a specific topic bound to the exchange of a RabbitMQ instance and passes the messages as input parameters to invoke functions (**RabbitMQ example event**).<br>For details, see **Using a DMS (for RabbitMQ) Trigger**. |
| GeminiDB Mongo | If you create a GeminiDB Mongo trigger for a function, any updates (**GeminiDB Mongo example event**) to the specified database table will trigger the function.<br>For details about how to use GeminiDB Mongo triggers, see **Using a GeminiDB Mongo Trigger**. |

| Cloud Service/ Feature | Trigger Event |
| --- | --- |
| IoTDA | FunctionGraph can use IoTDA trigger to track device properties, message reporting, and status changes as well as analyze, sort out, and measure data flows (**IoTDA example event**).<br>For details, see **Using an IoTDA Trigger**. |
| SMN | Simple Message Notification (SMN) sends messages to email addresses, mobile phones, or HTTP/HTTPS URLs. If you create a function with an SMN trigger, messages published to a specified topic will be passed as a parameter (**SMN example event**) to invoke the function. Then, the function processes the event, for example, publishing messages to other SMN topics or sending them to other cloud services.<br>For details, see **Using an SMN Trigger**. |
| OBS | Create a function for processing OBS events, such as object creation or deletion (**OBS example events**). When an image is uploaded to a specified bucket, OBS invokes the function to read the image and create a thumbnail.<br>For details, see:<br>• **Using an OBS Trigger**. |
| EG | EventGrid (EG) receives messages from event sources and passes the message payload as a parameter (**EG example event**) to invoke a function. Then, the function processes the events, for example, sending messages to other cloud services.<br>EventGrid supports the following event sources:<br>• **Creating an EG Trigger (OBS Application Service)**<br>• **Using an EG Trigger (RocketMQ custom event source)**<br>• **Using an EG Trigger (RabbitMQ custom event source)** |

## Trigger Event Examples

Timer

- TIMER example event. For details about the parameters, see **Table 5-17**.

```
{
    "version": "v2.0",
    "time": "2023-06-01T08:30:00+08:00",
    "trigger_type": "TIMER",
    "trigger_name": "Timer_001",
    "user_event": "User Event"
}
```

**Table 5-17** Parameters of a timer example event

| Parameter | Category | Description |
| --- | --- | --- |
| version | String | Event version |
| time | String | Event occurrence time, in ISO 8601 format |
| trigger_type | String | Trigger type |
| trigger_name | String | Trigger name |
| user_event | String | Additional information of the trigger |

API Gateway (Dedicated)

- API Gateway event example. For details about the parameters, see **Table 5-18**.

```
{
    "body": "",
    "requestContext": {
        "apiId": "bc1dcffd-aa35-474d-897c-d53425a4c08e",
        "requestId": "11cdcdcf33949dc6d722640a13091c77",
        "stage": "RELEASE"
    },
    "queryStringParameters": {
        "responseType": "html"
    },
    "httpMethod": "GET",
    "pathParameters": {},
    "headers": {
        "accept-language": "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2",
        "accept-encoding": "gzip, deflate, br",
        "x-forwarded-port": "443",
        "x-forwarded-for": "103.218.216.98",
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "upgrade-insecure-requests": "1",
        "host": "50eedf92-c9ad-4ac0-827e-d7c11415d4f1.apigw.region.cloud.com",
        "x-forwarded-proto": "https",
        "pragma": "no-cache",
        "cache-control": "no-cache",
        "x-real-ip": "103.218.216.98",
        "user-agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0"
    },
    "path": "/apig-event-template",
    "isBase64Encoded": true
}
```

**Table 5-18** Parameters of an APIG example event

| Parameter | Category | Description |
| --- | --- | --- |
| body | String | Actual request in string format |

| Parameter | Category | Description |
| --- | --- | --- |
| requestContext | Map | Request information, including the API gateway configuration, request ID, authentication information, and source |
| httpMethod | String | HTTP method |
| queryStringParameters | Map | Query strings configured in APIG and their actual values |
| pathParameters | Map | Path parameters configured in APIG and their actual values |
| headers | Map | Complete headers |
| path | String | Complete path |
| isBase64Encoded | Boolean | The default value is **true**. |

**Constraints:**

- When calling a function using APIG, **isBase64Encoded** is valued **true** by default, indicating that the request body transferred to FunctionGraph is encoded using Base64 and must be decoded for processing.
- The function must return characters strings by using the following structure.

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": {"headerName":"headerValue",...},
    "body": "..."
}
```

Cloud Trace Service (CTS)

DDS

- DDS example event. For details about the parameters, see **Table 5-19**.

```
{
   "records": [
      {
         "event_source": "dds",
         "event_name": "insert",
         "region": "region",
         "event_version": "1.0",
         "dds": {
            "size_bytes": "100",
            "token": "{\"_data\":
\"825D8C2F4D0000001529295A100474039A3412A64BA89041DC952357FB4446645F696400645D8C2F
8E5BECCB6CF5370D6A0004\"}",
            "full_document": "{\"_id\": {\"$oid\": \"5d8c2f8e5beccb6cf5370d6a\"},\"name\": \"dds
\",\"age\": {\"$numberDouble\": \"52.0\"}}",
            "ns": "{\"db\": \"functiongraph\",\"coll\": \"person\"}"
         },
         "event_source_id": "e6065860-f7b8-4cca-80bd-24ef2a3bb748"
```

```
    }
  ]
}
```

**Table 5-19** Parameters of a DDS example event

| Parameter | Category | Description |
| --- | --- | --- |
| region | String | Region where the DDS instance is located |
| event_version | String | Event version |
| event_source | String | Event source |
| event_name | String | Event name |
| size_bytes | Int | Message bytes |
| token | String | Base64-encoded data |
| full_document | String | Complete file information |
| ns | String | Column name |
| event_source_id | String | Event source ID |

Data Ingestion Service (DIS)

- DIS example event. For details about the parameters, see **Table 5-20**.

```
{
   "ShardID": "shardId-0000000000",
   "Message": {
      "next_partition_cursor":
"eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzLXN3dGVzdCIsInBhcnRpdGlvbi1pZCI6InNoYXJkSWQtMDAwMDAwMDAwMCIsImN1cnNvci10eXBlIjoiVFJJTV9IT1JJWk9OIiwic3RhcnRpbmctc2VxdWVuY2UtbnVtYmVyIjoiNCJ9LCJnZW5lcmF0ZVRpbWVzdGFtcCI6MTUwOTYwNjM5MjE5MX0",
      "records": [
         {
            "partition_key": "shardId_0000000000",
            "data": "d2VsY29tZQ==",
            "sequence_number": "0"
         },
         {
            "partition_key": "shardId_0000000000",
            "data": "dXNpbmc=",
            "sequence_number": "1"
         },
         {
            "partition_key": "shardId_0000000000",
            "data": "RnVuY3Rpb25TdGFnZQ==",
            "sequence_number": "2"
         },
         {
            "partition_key": "shardId_0000000000",
            "data": "c2VydmljZQ==",
            "sequence_number": "3"
         }
      ],
      "millis_behind_latest": ""
   },
   "Tag": "latest",
```

```
    "StreamName": "dis-swtest"
}
```

**Table 5-20** Parameters of a DIS example event

| Parameter | Category | Description |
| --- | --- | --- |
| ShardID | String | Partition ID |
| next_partition_cursor | String | Next partition cursor |
| Records | Map | Data records stored in a DIS stream |
| partition_key | String | Partition key |
| data | String | Data blocks, which are added by the data producer to the stream |
| sequence_number | Int | Record ID, which is automatically allocated by DIS |
| Tag | String | Stream tag |
| StreamName | String | Stream name |

DMS (for Kafka)/Kafka (Open-Source)

- Kafka example event. For details about the parameters, see **Table 5-21**.

```
{
    "event_version": "v1.0",
    "event_time": 1576737962,
    "trigger_type": "KAFKA",
    "region": "region",
    "instance_id": "81335d56-b9fe-4679-ba95-7030949cc76b",
    "records": [
        {
            "messages": [
                "kafka message1",
                "kafka message2",
                "kafka message3",
                "kafka message4",
                "kafka message5"
            ],
            "topic_id": "topic-test"
        }
    ]
}
```

**Table 5-21** Parameters of a Kafka example event

| Parameter | Category | Description |
| --- | --- | --- |
| event_version | String | Event version |
| event_time | String | Time when an event occurs |

| Parameter | Category | Description |
| --- | --- | --- |
| trigger_type | String | Event type |
| region | String | Region where a Kafka instance resides |
| instance_id | String | Kafka instance ID |
| messages | String | Message content |
| topic_id | String | Message ID |

DMS (for RabbitMQ)

- DMS for RabbitMQ example event. For details about the parameters, see **Table 5-22**.

```
{
    "event_version": "v1.0",
    "event_time": 1576737962,
    "trigger_type": "RABBITMQ",
    "region": "region",
    "records": [
        {
            "messages": [
                "rabbitmq message1",
                "rabbitmq message2",
                "rabbitmq message3",
                "rabbitmq message4",
                "rabbitmq message5"
            ],
            "instance_id": "81335d56-b9fe-4679-ba95-7030949cc76b",
            "exchange": "exchange-test"
        }
    ]
}
```

**Table 5-22** DMS for RabbitMQ parameters

| Parameter | Category | Description |
| --- | --- | --- |
| event_version | String | Event version |
| region | String | Region where a RabbitMQ instance resides |
| instance_id | String | RabbitMQ instance ID |

GeminiDB Mongo

- GeminiDB Mongo example event. For details about the parameters, see **Table 5-23**.

```
{
    "records": [
        {
            "event_name": "\"insert\"",
            "event_version": "1.0",
            "event_source": "gauss_mongo",
            "region": "cn-north-xx",
```

```
          "gauss_mongo": {
              "full_document": "{\"_id\": {\"$oid\":\"5f61de944778db5fcded3f87\"},\"zhangsan\":
\"zhangsan\"}",
              "ns": "{\"db\": \"zhangsan\",\"coll\": \"zhangsan\"}",
              "size_bytes": "100",
              "token": "{\"_data\":
\"825F61DE940000000129295A1004A2D9AE61206C43A5AF47CAF7C5C00C5946645F696400645F61DE
944778DB5FCDED3F870004\"}"
          },
          "event_source_id": "51153d19-2b7d-402c-9a79-757163258a36"
       }
   ],
   "vernier": "{\"_data\":
\"825F61DE940000000129295A1004A2D9AE61206C43A5AF47CAF7C5C00C5946645F696400645F61DE
944778DB5FCDED3F870004\"}"
}
```


**Table 5-23** Parameters of a GeminiDB Mongo example event

| Parameter | Category | Description |
| --- | --- | --- |
| region | String | Region where a GeminiDB instance resides |
| event_source | String | Event source |
| event_version | String | Event version |
| full_document | String | Complete file information |
| size_bytes | Int | Message bytes |
| token | String | Base64-encoded data |
| event_source_id | String | Event source ID |
| vernier | String | Cursor |

GeminiDB DynamoDB

- IoTDA example event. For details about the parameters, see **Table 5-24**.


```
{
   "resource" : "device",
   "event" : "create",
   "event_time" : "20240919T011335Z",
   "event_time_ms" : "2024-09-19T01:13:35.854Z",
   "request_id" : "75127474-1a26-4578-8847-3128d6101954",
   "notify_data" : {
      "body" : {
         "app_id" : "3d40caf3ddfc4e83815b54b50f13aad7",
         "app_name" : "DefaultApp_6439vdv2",
         "device_id" : "66eb7a0ffa8d9c36870c6892_ttytytytytytyt",
         "node_id" : "ttytytytytytyt",
         "gateway_id" : "66eb7a0ffa8d9c36870c6892_ttytytytytytyt",
         "node_type" : "GATEWAY",
         "auth_info" : {
            "auth_type" : "SECRET",
            "secure_access" : false,
            "timeout" : 0
         },
         "product_id" : "66eb7a0ffa8d9c36870c6892",
         "product_name" : "test",
```

```
        "status" : "INACTIVE",
        "create_time" : "20240919T011335Z"
      }
    }
}
```

**Table 5-24** Parameters of an IoTDA example event

| Parameter | Category | Description |
| --- | --- | --- |
| resource | string | Data source, which includes device, device property, device message, device message status, device status, batch task, product, and device asynchronous command status and run log. |
| event | string | Triggering event. |
| event_time | string | Event triggering time in the character string format. |
| event_time_ms | string | Event triggering time in datetime format. |
| request_id | string | Request ID. |
| notify_data | Object. For details, see **Table 5-25**. | Message to push. |

**Table 5-25** NotifyData

| Parameter | Category | Description |
| --- | --- | --- |
| body | Object. For details, see **Table 5-26**. | Message content. |

**Table 5-26** NotifyDataBody

| Parameter | Category | Description |
| --- | --- | --- |
| app_id | string | Resource space ID. |
| app_name | string | Resource space name. |

| Parameter | Category | Description |
| --- | --- | --- |
| device_id | string | Device ID, used to uniquely identify a device. The value of this parameter is specified during device registration or allocated by the platform. If the value is allocated by the platform, the value is in the format of *[product_id]_[node_id]*.<br>Maximum length: 256 characters |
| node_id | string | Device identifier. This parameter is set to the IMEI, MAC address, or serial number.<br>Maximum length: 64 characters |
| gateway_id | string | Gateway ID, which is the device ID of the parent device. The gateway ID is the same as the device ID if the device is a directly connected device. If the device is an indirectly connected device, the gateway ID is the device ID of the directly connected device with which it associates. |
| node_type | string | Device node type. |
| product_id | string | Unique ID of the product associated with the device. |
| product_name | string | Name of the product associated with the device. |
| status | string | Device status.<br>• **ONLINE**: The device is online.<br>• **OFFLINE**: The device is offline.<br>• **ABNORMAL**: The device is abnormal.<br>• **INACTIVE**: The device is not activated.<br>• **FREEZED**: The device is frozen. |
| create_time | string | Time when the device was registered on the platform. The value is in the format of yyyyMMdd'T'HHmmss'Z', for example, **20151212T121212Z**. |
| auth_info | Object. For details, see **Table 5-27**. | Access authentication information about the device. |

**Table 5-27** AuthInfo

| Parameter | Category | Description |
| --- | --- | --- |
| auth_type | string | Authentication mode. Secret authentication (**SECRET**) and certificate authentication (**CERTIFICATES**) are supported. If secret authentication is used, fill in **secret**. If certificate authentication is used, fill in **fingerprint**. If **auth_type** is not set, secret authentication is used by default. |
| secure_access | Boolean | Whether the device is connected to the platform using a secure protocol. The default value is **true**.<br>• **true**: The device is connected to the platform using a secure protocol.<br>• **false**: The device is connected to the platform using an insecure protocol. |
| timeout | Integer | Validity period of the device verification code, in seconds. The default value is **0**. If the device has not been connected to the platform within the validity period, the platform deletes the registration information of the device. If this parameter is set to **0**, the verification code is always valid. The recommended value is **0**. Note: This parameter is returned only when **timeout** is modified in the device registration or modification API.<br>Minimum value: **0**<br>Maximum value: **2147483647**<br>Default value: **0** |

For details about device messages, see the IoTDA official website. For example, **device addition notification**.

Log Tank Service (LTS)

Simple Message Notification (SMN)

- SMN example event. For details about the parameters, see **Table 5-28**.

```
{
    "record": [
        {
            "event_version": "1.0",
            "smn": {
                "topic_urn": "topicUrn",
                "timestamp": "2018-01-09T07:11:40Z",
                "message_attributes": null,
                "message": "this is smn message content",
                "type": "notification",
                "message_id": "a51671f77d4a479cacb09e2cd591a983",
                "subject": "this is smn message subject"
```

```
        },
        "event_subscription_urn": "functionUrn",
        "event_source": "smn"
    }
  ],
  "functionname": "test",
  "requestId": "7c307f6a-cf68-4e65-8be0-4c77405a1b2c",
  "timestamp": "Tue Jan 09 2018 15:11:40 GMT+0800 (CST)"
}
```

**Table 5-28** Parameters of an SMN example event

| Parameter | Category | Description |
| --- | --- | --- |
| event_version | String | Event version |
| topic_urn | String | Unique ID of an SMN event, which is generated by the SMN service |
| type | String | Event type |
| requestId | String | Request ID, which is generated by FunctionGraph.<br>The ID of each request is unique. |
| message_id | String | Message ID, which is generated by SMN.<br>The ID of each message is unique. |
| message | String | Message |
| event_source | String | Event source |
| event_subscription_urn | String | Function URN. The value is unique and can be obtained from the function details page. |
| timestamp | String | Time when an event occurs |

Object Storage Service (OBS)

- OBS example event. For details about the parameters, see **Table 5-29**.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "2018-01-09T07:50:50.028Z",
      "requestParameters": {
        "sourceIPAddress": "103.218.216.125"
      },
      "s3": {
        "configurationId": "UK1DGFPYUKUZFHNQ00000160CC0B471D101ED30CE24DF4DB",
        "object": {
          "eTag": "9d377b10ce778c4938b3c7e2c63a229a",
          "sequencer": "00000000160D9E681484D6B4C0000000",
          "key": "job.png",
          "size": 777835
        },
        "bucket": {
          "arn": "arn:aws:s3:::syj-input2",
          "name": "functionstorage-template",
```

```
                "ownerIdentity": {
                    "PrincipalId": "0ed1b73473f24134a478962e631651eb"
                }
            }
        },
        "Region": "{region}",
        "eventName": "ObjectCreated:Post",
        "userIdentity": {
            "principalId": "9bf43789b1ff4b679040f35cc4f0dc05"
        }
    }
  ]
}
```

**Table 5-29** Parameters of an OBS example event

| Parameter | Category | Description |
| --- | --- | --- |
| eventVersion | String | Event version |
| eventTime | String | Time when an event occurs.<br>The ISO-8601 time format is used. |
| sourceIPAddress | String | Source IP address |
| s3 | Map | OBS event content |
| object | Map | object parameter description |
| bucket | Map | bucket parameter description |
| arn | String | Bucket ID |
| ownerIdentity | Map | ID of the user who creates the bucket |
| Region | String | Region where the bucket is located |
| eventName | String | Event name |
| userIdentity | Map | ID of the Huawei Cloud account that initiates the request |

EventGrid (EG)

- EG example event. For details about the parameters, see **Table 5-30**.

  **Custom RocketMQ event source**

```
{
  "datacontenttype": "application/json",
  "data": {
      "context": "yyyyy"
  },
  "subject": "ROCKETMQ:region:domainId/projectId:ROCKETMQ:eventSourceName",
  "specversion": "1.0",
```

```
    "id": "016d5bd3-6231-4e9e-86ef-e451a070d598",
    "source": "eventSourceName",
    "time": "2023-04-07T11:51:10Z",
    "type": "ROCKETMQ:CloudTrace:RocketmqCall"
}
```

### Custom RabbitMQ event source

```
{
"datacontenttype": "application/json",
    "data": {
        "context": "yyyyy"
    },
    "subject": "RABBITMQ:region:domainId/projectId:RABBITMQ:eventSourceName",
    "specversion": "1.0",
    "id": "016d5bd3-6231-4e9e-86ef-e451a070d598",
    "source": "eventSourceName",
    "time": "2023-04-07T11:51:10Z",
    "type": "RABBITMQ:CloudTrace:RabbitmqCall"
}
```

### OBS application service event source

```
{
    "channel_id":"b65779ed-d9d0-4a6c-b312-c767226964cf",
    "description":"",
    "name":"subscription-xeak",
    "sources":[
        {
            "id":null,
            "name":"HC.OBS.DWR",
            "detail":{
                "bucket":"eventbucket",
                "objectKeyEncode":true
            },
            "filter":{
                "source":[
                    {
                        "op":"StringIn",
                        "values":[
                            "HC.OBS.DWR"
                        ]
                    }
                ],
                "type":[
                    {
                        "op":"StringIn",
                        "values":[
                            "OBS:DWR:ObjectCreated:PUT",
                            "OBS:DWR:ObjectCreated:POST"
                        ]
                    }
                ],
                "subject":{
                    "and":[
                        {
                            "op":"StringStartsWith",
                            "values":[
                                "/ddd"
                            ]
                        }
                    ]
                },
                "data":{
                    "obs":{
                        "bucket":{
                            "name":[
                                {
                                    "op":"StringIn",
                                    "values":[
                                        "output-your"
                                    ]
```

```
                        }
                    ]
                }
            }
        }
    },
    "provider_type":"OFFICIAL"
  }
 ],
 "targets":[
  {
    "id":null,
    "name":"HC.FunctionGraph",
    "detail":{
      "urn":"urn:fss:cn-north-7:c53626012ba84727b938ca8bf03108ef:function:A-nodejs-
lqz:pylog:latest",
      "agency_name":"EG_AGENCY"
    },
    "dead_letter_queue":null,
    "provider_type":"OFFICIAL",
    "transform":{
      "type":"ORIGINAL",
      "value":""
    }
  }
 ]
}
```


**Table 5-30** Parameters of an EG example event

| Parameter | Category | Description |
| --- | --- | --- |
| datacontenttype | String | Data type |
| data | Map | Data |
| subject | String | Target value |
| specversion | String | Version |
| id | String | Unique key |
| source | String | Event source name |
| time | String | Subscription time |
| type | String | Subscription type |

## 5.6.2 Timer Trigger

This section describes how to create a timer trigger on the FunctionGraph console to trigger a function at a specified frequency.

Common application scenarios:

- Back up important data. For example, back up data every seven days.
- Monitors the server status and resource usage. For example, functions are executed and server usage reports are sent at 10:00 a.m. every day.

## Event Description

A timer trigger invokes a function based on a fixed frequency (minutes, hours, or days) or a specified **Cron expression**. It is suitable for scenarios where tasks need to be executed periodically.

You can use timer triggers for shared functions in the **LA-Sao Paulo1** region.

Example events of a timer trigger

- A timer trigger invokes a function based on the following event format. For details about the parameters, see **Table 5-31**.

```
{
    "version": "v2.0",
    "time": "2023-06-01T08:30:00+08:00",
    "trigger_type": "TIMER",
    "trigger_name": "Timer_001",
    "user_event": "User Event"
}
```

**Table 5-31** Parameters of a timer example event

| Parameter | Category | Description |
| --- | --- | --- |
| version | String | Event function version |
| time | String | Time when an event occurs |
| trigger_type | String | Trigger type. The trigger type is Timer. |
| trigger_name | String | Trigger name |
| user_event | String | Additional information of the trigger |

## Notes and Constraints

HTTP functions do not support timer triggers.

## Prerequisites

You have created a function. For details, see **Creating a Function from Scratch**

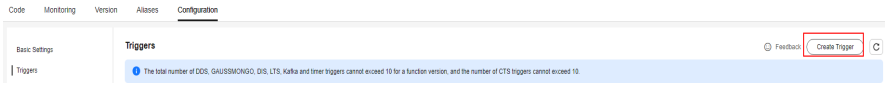
## Creating a Timer Trigger

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of the function to be configured. The function details page is displayed.

   You can configure a timer trigger for a shared function on the **Shared** page in the **LA-Sao Paulo1** region. For details about shared functions, see **Sharing Functions Based on RAM**.
3. Choose **Configuration** > **Triggers** and click **Create Trigger**, as shown in **Figure 5-22**.

**Figure 5-22** Creating a trigger




4. Configure the following parameters:

**Table 5-32** Timer trigger parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Select **Timer**. | Timer |
| Timer Name | Customize a name. The value can contain letters, digits, underscores (_), and hyphens (-). It must start with a letter and cannot exceed 64 characters. | Timer-fg |
| Rule | Trigger rule. You can select **Fixed rate** or **Cron expression**.<br>• **Fixed rate**: The function is triggered at a fixed rate of minutes, hours, or days. You can set a fixed rate from 1 to 60 minutes, 1 to 24 hours, or 1 to 30 days.<br>• **Cron expression**: The function is triggered based on a complex rule. For example, you can set a function to be executed at 08:30:00 from Monday to Friday. For more information, see **Cron Expression Rules**. | Fixed rate<br>1 minute |
| Enable Trigger | Determine whether to enable the timer trigger. If this parameter is disabled, the function will not be triggered at the specified frequency. | Enabled |
| Additional Information | Optional.<br>The additional information you configure will be put into the **user_event** field of the timer event source. For details, see **Supported Event Sources**. | - |

5. Click **OK**.

## Viewing the Execution Result

After a timer trigger is created, the function will be executed periodically according to the configured trigger rule. You can view function run logs on the function details page.

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** On the **Functions** page, click the name of the function to be configured.

**Step 3** Choose **Monitoring** > **Logs** to query **function running logs**.

**----End**

## Cron Expression Rules

You can configure a cron expression in the following two formats for a function timer trigger:

- **@every format**

  @every *N*unit. *N* is a positive integer. **unit** can be ns, µs, ms, s, m, or h. An @every expression means to invoke a function every *N* time units, as shown in **Table 5-33**.

  **Table 5-33** Example expressions

  | Expression | Description |
  | --- | --- |
  | @every 30m | Triggers a function every 30 minutes. |
  | @every 1h | Triggers a function every hour. |
  | @every 2h30m | Triggers a function every 2.5 hours. |

- **Standard format**

  The format is "*seconds minutes hours day-of-month month day-of-week*". *day-of-week* is optional. The fields must be separated from each other using a space. **Table 5-34** describes the fields in a standard cron expression. For details about the special characters, see **Table 5-37**.

**Table 5-34** Cron expression fields

| Field | Description | Value Range | Special Characters Allowed |
| --- | --- | --- | --- |
| CRON_TZ | Optional. If this parameter is not set, the region's time zone is used by default.<br>If your task will run in a specific time zone, use **CRON_TZ** to specify the time zone. For example, to trigger your function at 04:00 on the first day of each month (Beijing time), use **CRON_TZ=Asia/Shanghai 0 0 4 1 * ***.<br>The time zone expression varies depending on the region. Query the CRON_TZ expression of the required region. For details, see **Table 5-38**. | - | - |
| Seconds | Mandatory | 0-59 | , - * / |
| Minute | Mandatory | 0-59 | , - * / |
| Hours | Mandatory | 0-23 | , - * / |
| Day-of-month | Mandatory | 1-31 | , - * ? / |
| Month | Mandatory | 1–12 or Jan–Dec. The value is case-insensitive, as shown in **Table 5-35**. | , - * / |
| Day-of-week | Optional | 0–6 or Sun–Sat. The value is case-insensitive, as shown in **Table 5-36**. **0** means Sunday. | , - * ? / |

**Table 5-35** Value description of the month field

| Month | Digit | Abbreviation |
| --- | --- | --- |
| January | 1 | Jan |

| Month | Digit | Abbreviation |
| --- | --- | --- |
| February | 2 | Feb |
| March | 3 | Mar |
| April | 4 | Apr |
| May | 5 | May |
| June | 6 | Jun |
| July | 7 | Jul |
| August | 8 | Aug |
| September | 9 | Sep |
| October | 10 | Oct |
| November | 11 | Nov |
| December | 12 | Dec |

**Table 5-36** Value description of the day-of-week field

| Day of Week | Digit | Abbreviation |
| --- | --- | --- |
| Monday | 1 | Mon |
| Tuesday | 2 | Tue |
| Wednesday | 3 | Wed |
| Thursday | 4 | Thu |
| Friday | 5 | Fri |
| Saturday | 6 | Sat |
| Sunday | 0 | Sun |

Table 5-37 describes the special characters that can be used in a cron expression.

**Table 5-37** Special character description

| Special Character | Meaning | Description |
| --- | --- | --- |
| * | Used to specify all values within a field. | ***** in the minutes field means every minute. |

| Special Character | Meaning | Description |
| --- | --- | --- |
| , | Used to specify multiple values, which can be discontinuous. | For example, "Jan,Apr,Jul,Oct" or "1,4,7,10" in the month field and "Sat,Sun" or "6,0" in the day-of-week field. |
| - | Used to specify a range. | For example, "0-3" in the minutes field. |
| ? | Used to specify something in one of the two fields in which the character is allowed, but not the other. | You can specify something only in the day-of-month or day-of-week field. For example, if you want your function to be executed on a particular day (such as the 10th) of the month, but do not care what day of the week that is, then put "10" in the day-of-month field and "?" in the day-of-week field. |
| / | Used to specify increments. The character before the slash indicates when to start, and the one after the slash represents the increment. | For example, "1/3" in the minutes field means to trigger the function every 3 minutes starting from 00:01:00 of the hour. |

Table 5-38 describes several example cron expressions.

**Table 5-38** Example cron expressions

| Function Scheduling Example | Cron Expression (Beijing Time) |
| --- | --- |
| 12:00 every day | CRON_TZ=Asia/Shanghai 0 0 12 * * * |
| 12:30 every day | CRON_TZ=Asia/Shanghai 0 30 12 * * * |
| 26th, 29th, and 33rd minutes of each hour | CRON_TZ=Asia/Shanghai 0 26,29,33 * * * * |
| 12:30 from Monday to Friday | CRON_TZ=Asia/Shanghai 0 30 12 ? * MON-FRI |
| Every 5 minutes during 12:00 and 14:59 from Monday to Friday | CRON_TZ=Asia/Shanghai 0 0/5 12-14 ? * MON-FRI |
| 12:00 every day from January to April | CRON_TZ=Asia/Shanghai 0 0 12 ? JAN,FEB,MAR,APR * |

### Creating a Timer Trigger for a Shared Function

You can configure a timer trigger for a shared function in the **LA-Sao Paulo1** region. For details about the shared function, see **Sharing Functions Based on RAM**.

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. On the **Shared** tab, click the function name to go to the function details page.
3. The subsequent steps are the same as those for creating a common function. For details, see **3**.

### Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

## 5.6.3 API Gateway (Dedicated) Trigger

This section describes how to create an APIG trigger on the FunctionGraph console to invoke a function using an API.

### Event Description

Dedicated APIG triggers are used together with **APIG** to invoke functions over HTTPS or HTTP. You can map specific API operations (such as GET, POST, and PUT) to the corresponding functions using the custom REST APIs and endpoints of APIG. When you send an HTTPS request to an APIG endpoint, APIG automatically triggers the corresponding function.

Example event of a dedicated APIG trigger

- APIG triggers functions using the following event format. For details about the parameters, see **Table 5-39**.

```
{
    "body": "",
    "requestContext": {
        "apiId": "bc1dcffd-aa35-474d-897c-d53425a4c08e",
        "requestId": "11cdcdcf33949dc6d722640a13091c77",
        "stage": "RELEASE"
    },
    "queryStringParameters": {
        "responseType": "html"
    },
    "httpMethod": "GET",
    "pathParameters": {},
    "headers": {
        "accept-language": "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2",
        "accept-encoding": "gzip, deflate, br",
        "x-forwarded-port": "443",
        "x-forwarded-for": "103.218.216.98",
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "upgrade-insecure-requests": "1",
        "host": "50eedf92-c9ad-4ac0-827e-d7c11415d4f1.apigw.region.cloud.com",
        "x-forwarded-proto": "https",
        "pragma": "no-cache",
        "cache-control": "no-cache",
        "x-real-ip": "103.218.216.98",
        "user-agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0"
    },
    "path": "/apig-event-template",
```

```
    "isBase64Encoded": true
}
```

**Table 5-39** Parameters of an APIG example event

| Parameter | Category | Description |
| --- | --- | --- |
| body | String | Actual request in string format |
| requestContext | Map | Request information, including the API gateway configuration, request ID, authentication information, and source |
| httpMethod | String | HTTP method |
| queryStringParameters | Map | Query strings configured in APIG and their actual values |
| pathParameters | Map | Path parameters configured in APIG and their actual values |
| headers | Map | Complete headers |
| path | String | Complete path |
| isBase64Encoded | Boolean | The default value is true, indicating that the request body sent by APIG to FunctionGraph has been encoded using Base64 and needs to be decoded using Base64 before being processed. |

## Notes and Constraints

- The regions and runtimes supported for dedicated APIG triggers are subject to the console.
- Dedicated APIG triggers cannot be disabled and can only be deleted.
- The function response for APIG invocation is encapsulated and **must contain body(String), statusCode(int), headers(Map), and isBase64Encoded(boolean)**.

  ```
  {
      "isBase64Encoded": true|false,
      "statusCode": httpStatusCode,
      "headers": {"headerName":"headerValue",...},
      "body": "..."
  }
  ```

  **isBase64Encoded** is valued **true** by default, indicating that the request body transferred to FunctionGraph is encoded using Base64 and must be decoded for processing.

- The valid payload size of a request body is 4 MB when a dedicated APIG trigger is used.

## Prerequisites

- You have created a dedicated APIG gateway. For details, see **Creating a Gateway**.
  - To **invoke a function over the public network**, enable the public access and configure the required bandwidth when creating an APIG gateway.
  - To **configure a function to access resources in a VPC** in the production environment, select the same VPC when creating an APIG gateway.
- You have created an API group, for example, **APIGroup_test**. For details, see **Creating an API Group**.

## Creating an APIG (Dedicated) Trigger

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

**Step 2** On the **Function List** page, click **Create Function** in the upper right corner.

**Step 3** Select **Create from scratch**, configure the following function information, and retain the default values for other parameters.

- **Function Name**: Enter a function name, for example, **apig_demo**.
- **Enterprise Project**: Select **default**.
- **Agency**: Select **Use no agency**.
- **Runtime**: Select **Python 3.12**.

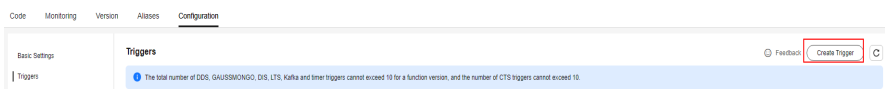
**Step 4** Click **Create Function**. The function creation is complete, and the function details page is displayed.

**Step 5** On the **Code** tab page, copy the following code to the code editing area 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
    }
```

**Step 6** Choose **Configuration** > **Triggers** and click **Create Trigger** as shown in **Figure 5-23**.

**Figure 5-23** Creating a trigger




**Step 7** Set trigger parameters by referring to **Table 5-40**.

**Table 5-40** Parameters for creating a dedicated APIG trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Select **API Gateway (Dedicated)**. | API Gateway (Dedicated) |
| API Instance | APIG gateway. If no gateway is available, click **Create API Instance**. | apig-fg |
| API Name | Name of a dedicated APIG trigger.<br>The name can contain 3 to 64 characters and must start with a letter. Only letters, digits, and underscores (_) are allowed.<br>For details about API configuration, see **Creating an API**. | API_apig |
| API Group | Select an API group. An API group is a collection of APIs. You can manage APIs by API group.<br>If no group is available, click **Create API Group**. | APIGroup_test |
| Environment | The environment where the API is published. An API can be called in different environments, such as production, test, and development environments only when the parameter is set to **RELEASE**.<br>If no environment is available, click **Create Environment**. | RELEASE |
| Security Authentication | Security authentication mode used by the API.<br>Options:<br>• **App**: AppKey and AppSecret authentication. This mode is of high security and is recommended. For details, see **App Authentication**.<br>• **IAM**: IAM authentication. This mode grants access permissions to IAM users only and is of medium security. For details, see **IAM Authentication**<br>• **None**: No authentication. This mode grants access permissions to all users.<br>In this example, **Security Authentication** is set to **None**. You are advised to enable App or IAM authentication in the production environment. | None |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Protocol | Request protocol of the API.<br>Supported request protocols:<br>• **HTTP**: Data is not encrypted during transmission.<br>• **HTTPS**: Data is encrypted during transmission. HTTPS is recommended for transmitting important or sensitive data.<br>• **HTTP&HTTPS**: Both HTTP and HTTPS protocols are supported. | HTTPS |
| Path | Enter the API request path.<br>It is a part of the API access address and is used to specify resource IDs or hierarchy.<br>**The path is case-sensitive.** It starts with a slash (/) and contains a maximum of 512 characters. Enclose parameters in braces. For example: **/a/{b}**. Or use a plus sign (+) to match parameters starting with specific characters. For example: **/a/{b+}**. | /ttest |
| Matching | Select a matching mode for the API.<br>Matching mode of the API. Options:<br>• **Exact match**: The request path must be the same as the defined path.<br>• **Prefix match**: The request path must start with the defined path. | Prefix match |
| Method | Request method of the API.<br>Options: GET, POST, DELETE, PUT, PATCH, HEAD, OPTIONS, and ANY.<br>**ANY** indicates that the API can be called using any request method. | ANY |
| Timeout (ms) | API backend timeout in milliseconds. Range: 1–60,000. | 5000 |

**Step 8** Click **OK**.

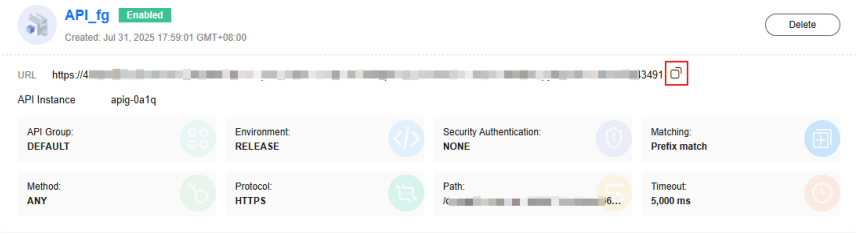
**----End**

## Invoking the Function

**Step 1** Copy the API URL of the APIG trigger, as shown in **Figure 5-24**, open a browser, and enter the URL in the address box.

**Figure 5-24** Copying the API URL




**Step 2** View the execution result, as shown in **Figure 5-25**.

**Figure 5-25** Returned result


**----End**

## Viewing the Execution Result

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click a function to go to its details page.

**Step 3** Choose **Monitoring** > **Logs** to query **function running logs**.

**----End**

# 5.6.4 API Connect (APIC) Trigger

This section describes how to create an APIC trigger on the FunctionGraph console to invoke a function using an API.

## Event Description

An APIC trigger is used together with **ROMA Connect**. With API operations such as GET and PUT mapped to specific functions, APIC can invoke corresponding functions to perform operations when receiving relevant HTTPS or HTTP requests.

## Notes and Constraints

The regions and runtimes supported for APIC triggers are subject to the console.

## Prerequisites

- You have created a function.
- You have created an APIC instance. For details, see **Creating an APIC Instance**.
- You have created an API group, for example, **APIConnect_test**. For details, see **Creating an API Group**.

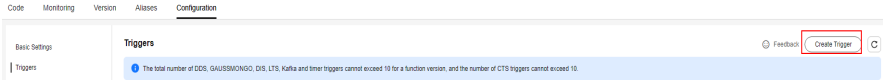
## Creating an APIC Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-26** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-41** APIC trigger parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **API Connect (APIC)**. | API Connect (APIC) |
| API Instance | Mandatory.<br>Select an APIC instance. If no instance is available, click **Create API Instance**. | APIC_fg |
| API Name | Mandatory.<br>APIG trigger name. The name can contain 3 to 64 characters and must start with a letter. Only letters, digits, and underscores (_) are allowed. | API_apic |
| API Group | Mandatory.<br>Select an API group. An API group is a collection of APIs. You can manage APIs by API group.<br>If no group is available, click **Create API Group**. | APIConnect_test |
| Environment | Mandatory.<br>The environment where the API is published. An API can be called in different environments, such as production, test, and development environments only when the parameter is set to **RELEASE**.<br>If no environment is available, click **Create Environment**. | RELEASE |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Security Authentication | Mandatory.<br>API authentication modes are as follows:<br>• **App**: AppKey and AppSecret authentication. This mode is of high security and is recommended. For details, see **App Authentication**.<br>• **IAM**: IAM authentication. This mode grants access permissions to IAM users only and is of medium security. For details, see **IAM Authentication**<br>• **None**: No authentication. This mode grants access permissions to all users. | None |
| Protocol | Mandatory.<br>There are two types of API request protocols:<br>• **HTTP**: Data is not encrypted during transmission.<br>• **HTTPS**: Data is encrypted during transmission. | HTTPS |
| Timeout (ms) | Mandatory.<br>API backend timeout in milliseconds. Range: 1–60,000. | 5000 |

**Step 5** Click **OK**.

NOTE

After the trigger is created, an API named **API_apic** is generated on the APIG console. You can click the API name in the trigger list to go to the APIG console.

**----End**

## Invoking the Function

**Step 1** Log in to ROMA Connect, find the selected instance (for example, **Ac6-instance-NoDelete**), and view the public IP address.

**Step 2** Enter the public IP address in the address box of the browser.

**Step 3** After the function is executed, a result is returned.

**----End**

## Viewing the Execution Result

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the desired function.

**Step 3** On the displayed function details page, click the **Logs** tab to query the function running logs. For details, see **Configuring and Viewing Function Invocation Logs**

**Step 4** Click **View Context** in the same row as a log to view log details.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.5 Cloud Trace Service (CTS) Trigger

This section describes how to create a CTS trigger for a function, and invoke the function in response to cloud resource operations recorded by CTS.

## Event Description

Cloud Trace Service (CTS) is a log audit service for cloud security. It allows you to collect, store, and query operation records of cloud resources for security analysis, compliance auditing, resource tracking, and fault locating.

With a CTS trigger, you can create a function to subscribe to event notifications based on the CTS service type and operation. Collected operation records will be passed as a parameter (**CTS example event**) to invoke the function. Then, the function analyzes and processes key information in the operation records, automatically recovers system or network modules, or reports alarms to service personnel by SMS or email.

You can configure CTS triggers for shared functions in the **LA-Sao Paulo1** region.

Example CTS trigger event

- CTS example events. For details about the parameters, see **Table 5-42**.

```
{
    "cts": {
        "time": 1529974447000,
        "user": {
            "name": "userName",
            "id": "5b726c4fbfd84821ba866bafaaf56aax",
            "domain": {
                "name": "domainName",
                "id": "b2b3853af40448fcb9e40dxj89505ba"
            }
        },
        "request": {},
        "response": {},
        "code": 204,
        "service_type": "FunctionGraph",
        "resource_type": "graph",
        "resource_name": "workflow-2be1",
        "resource_id": "urn:fgs:region:projectId:graph:workflow-2be1",
        "trace_name": "deleteGraph",
        "trace_type": "ConsoleAction",
        "record_time": 1529974447000,
        "trace_id": "69be64a7-0233-11e8-82e4-e5d37911193e",
        "trace_status": "normal"
    }
}
```

**Table 5-42** Parameters of a CTS example event

| Parameter | Category | Example Value | Description |
| --- | --- | --- | --- |
| time | Long | Reference example code | Time when a request is sent. The value is a 13-digit timestamp. |
| user | Map | Reference example code | Information about the user who initiates the request |
| request | Map | Reference example code | Event request content |
| response | Map | Reference example code | Event response content |
| code | Int | 204 | Response code, for example, **200** and **400** |
| service_type | String | FunctionGraph | Abbreviation of the sender, for example, **vpc** and **ecs** |
| resource_type | String | graph | Resource type of the sender, for example, **vm** and **vpn** |
| resource_name | String | workflow-2be1 | Resource name, for example, the name of an ECS |
| trace_name | String | deleteGraph | Event name (trace name), for example, **startServer** and **shutDown** |
| trace_type | String | ConsoleAction | Event type (trace type), for example, **ApiCall** |
| record_time | Long | Reference example code | Time when the CTS service receives the trace (13-digit timestamp) |

| Parameter | Category | Example Value | Description |
| --- | --- | --- | --- |
| trace_id | String | 69be64a7-0233-11e8-82e4-e5d37911193e | Event ID (trace ID) |
| trace_status | String | normal | Event status |

## Notes and Constraints

- The regions and runtimes supported for CTS triggers are subject to the console.

## Prerequisites

- You have **enabled CTS** and authorized related permissions.
- You have configured the CTS agency permission for the function. For details, see **Configuring Agency Permissions**.

## Creating a CTS Trigger

The following procedure describes how to configure a CTS trigger for a Python function.

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. On the **Function List** page, click **Create Function** in the upper right corner.
3. Set the following parameters:
   - **Function Name**: Enter a function name, for example, **HelloWorld**.
   - **Agency**: Select an agency that has the CTS permissions.
   - **Enterprise Project**: Select **default**.
   - **Runtime**: Select **Python 2.7**.
4. Click **Create Function**.
5. On the Code tab page, copy the following code to the code editing area. For details about parameters, see **Table 5-43**. Then, click **Deploy**.

```
# -*- coding:utf-8 -*-
'''
CTS trigger event:
{
 "cts":  {
      "time": "",
      "user": {
          "name": "userName",
          "id": "",
          "domain": {
              "name": "domainName",
              "id": ""
          }
      },
      "request": {},
      "response": {},
      "code": 204,
      "service_type": "FunctionGraph",
      "resource_type": "",
      "resource_name": "",
```

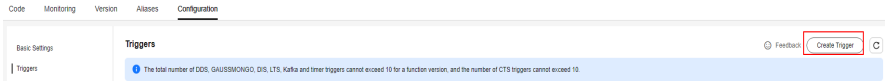
```
        "resource_id": {},
        "trace_name": "",
        "trace_type": "ConsoleAction",
        "record_time": "",
        "trace_id": "",
        "trace_status": "normal"
    }
}
'''
def handler (event, context):
    trace_name = event["cts"]["resource_name"]
    timeinfo = event["cts"]["time"]
    print(timeinfo+' '+trace_name)
```

**Table 5-43** Parameters

| Parameter | Description |
| --- | --- |
| time | The value is a 13-digit timestamp, for example, **1738805309469**. |
| record_time | The value is a 13-digit timestamp, for example, **1738805309469**. |

6. Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-27** Creating a trigger




7. Configure the following parameters.

**Table 5-44** Parameters for creating a CTS trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **Cloud Trace Service (CTS)**. | Select **Cloud Trace Service (CTS)**. |
| Event Notification Name | Mandatory.<br>Notification name of the CTS trigger. The value can contain a maximum of 64 characters, including letters, digits, and underscores (_). | CTS_fg |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Custom Operations | Mandatory.<br>A maximum of 10 services and 100 operations can be added. The parameters are as follows:<br>• **Service Type**: Select **FunctionGraph**.<br>If **Service Type** is set to a global cloud service, such as **OBS** or **IAM**, CTS triggers can be triggered only in **CN-Hong Kong**. For details about global cloud services, **submit a service ticket**.<br>• **Resource Type**: The resource type of the selected service, such as trigger, instance, and function.<br>• **Trace Name**:Operations that can be performed on the selected resource type, such as creating or deleting a trigger. | FunctionGraph; Functions; createFunction |

8. Click **OK**.

## Configuring a CTS Trigger for a Shared Function

You can configure a CTS trigger for a shared function in the **LA-Sao Paulo1** region. For details about the shared function, see **Sharing Functions Based on RAM**.

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. On the **Shared** tab, click the function name to go to the function details page.
3. The subsequent steps are the same as those for creating a common function. For details, see **6**.

## Configuring a CTS Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version, and click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 4** Set the parameters described in **Table 5-45** and click **Save**.

**Table 5-45** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **Cloud Trace Service (CTS)** and use the built-in CTS event template. |
| Event Name | Enter an event name, for example, **cts-test**. |
| Event data | The system automatically loads the event data in the CTS event template. You can modify the event data as required. |

**Step 5** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.6 Document Database Service (DDS) Trigger (Offline Soon)

This section describes how to create a DDS trigger on the FunctionGraph console. The trigger triggers a function each time a table in the DDS database is updated.

**The DDS trigger is about to be brought offline. You are advised not to create DDS triggers.**

## Notes and Constraints

- The DDS triggers are available only to existing users. You can check whether the trigger type is supported on the console.
- The valid payload size of a request body is 6 MB when a DDS trigger is used.

## Prerequisites

- **Function and configuration:**
  - You have created a function.
  - You have configured the DDS and VPC agency permissions for the function. For details, see **Configuring Agency Permissions**.
  - You have enabled VPC access for the function. For details, see **Configuring Networks**.
- **DDS:**
  - You have created a DDS DB instance.
  - You have created a DDS database.
  - You have configured the subnet permissions for the DDS security group. For details, see .

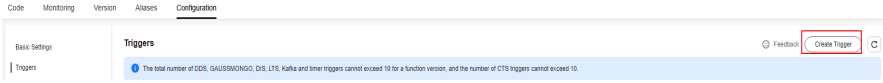
## Creating a DDS Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-28** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-46** Parameters for creating a DDS trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **Document Database Service (DDS)**. | Document Database Service (DDS) |
| DB Instance | Mandatory.<br>Select the created DDS database instance. If no instance is available, click **Create DB Instance**. | dds_fg |
| Password | Mandatory.<br>Password of the DDS DB instance administrator **rwuser**. The password must contain uppercase letters, lowercase letters, digits, and special characters. | DDStest@123 |
| Database | Mandatory.<br>Enter a database name. Note that **admin**, **local**, and **config** are reserved database names and cannot be used here. | DDS_test |
| Collection | Mandatory.<br>Enter a custom database collection name. For details about how to create a database collection, see **Creating a Collection**. | DDS_set |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Batch Size | Mandatory.<br>Maximum amount of data that can be processed by a function at a time. The value ranges from 1 to 10000. | 100 |

**Step 5** Click **OK**.

**----End**

## Configuring a DDS Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the function to be configured. The function details page is displayed.

**Step 3** Select a version and click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 4** Set the parameters described in **Table 5-47** and click **Save**.

**Table 5-47** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **Document Database Service (DDS)** to use the built-in DDS event template. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **dds-123test**. |
| Event data | The system automatically loads the built-in DDS event template, which is used in this example without modifications. |

**Step 5** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.7 Data Ingestion Service (DIS) Trigger

This section describes how to create a DIS trigger through the FunctionGraph console. You can configure DIS events using the built-in event template to trigger the function.

## Event Description

DIS builds data streams for custom applications capable of processing or analyzing streaming data to address the challenge of transmitting data from outside the cloud to inside the cloud. It can continuously capture, transmit, and store terabytes of data from multiple sources.

You can create a function to automatically poll a DIS stream and process all new data records, such as, website click streams, financial transactions, social media streams, IT logs, and location-tracking events.

Example event of a DIS trigger

- DIS triggers functions using the following event format. For details about the parameters, see **Table 5-48**.

```
{
   "ShardID": "shardId-0000000000",
   "Message": {
      "next_partition_cursor":
"eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzLXN3dGVzdCIsInBhcnRpdGlvbi1pZCI6InNoYXJkSWQtMDAwMDAwMDAwMDAwMCIsImN1cnNvci10eXBlIjoiVFJJTV9IT1JJWk9OIiwic3RhcnRpbmctc2VxdWVuY2UtbnVtYmVyIjoiNCJ9LCJnZW5lcmF0ZVRpbWVzdGFtcCI6MTUwOTYwNjM5MjE5MX0",
      "records": [
         {
            "partition_key": "shardId_0000000000",
            "data": "d2VsY29tZQ==",
            "sequence_number": "0"
         },
         {
            "partition_key": "shardId_0000000000",
            "data": "dXNpbmc=",
            "sequence_number": "1"
         },
         {
            "partition_key": "shardId_0000000000",
            "data": "RnVuY3Rpb25TdGFnZQ==",
            "sequence_number": "2"
         },
         {
            "partition_key": "shardId_0000000000",
            "data": "c2VydmljZQ==",
            "sequence_number": "3"
         }
      ],
      "millis_behind_latest": ""
   },
   "Tag": "latest",
   "StreamName": "dis-swtest"
}
```

**Table 5-48** Parameters of a DIS example event

| Parameter | Category | Description |
| --- | --- | --- |
| ShardID | String | Partition ID |

| Parameter | Category | Description |
| --- | --- | --- |
| next_partition_cursor | String | Next partition cursor |
| Records | Map | Data records stored in a DIS stream |
| partition_key | String | Partition key |
| data | String | Data blocks, which are added by the data producer to the stream |
| sequence_number | Int | Record ID, which is automatically allocated by DIS |
| Tag | String | Stream tag |
| StreamName | String | Stream name |

## Notes and Constraints

- The regions and runtimes supported for DIS triggers are subject to the console.
- The valid payload size of a request body is 6 MB when a DIS trigger is used.

## Prerequisites

- **Function and configuration:**
  - You have created a function. For details, see **Creating a Function from Scratch**
  - You have configured the DIS agency permission for the function. For details, see **Configuring Agency Permissions**.
- **DIS:**

  You have created a DIS stream, for example, **dis-function**. For details, see **Creating a DIS Stream**.

## Creating a DIS Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

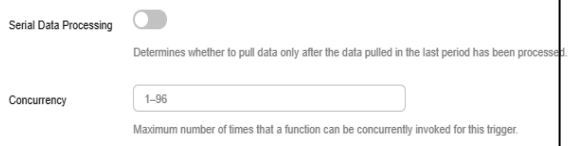
**Step 4** Set DIS trigger parameters by referring to **Table 5-49**.

**Table 5-49** DIS trigger parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Choose **Data Ingestion Service (DIS)**. | Data Ingestion Service (DIS) |
| Stream Name | Mandatory.<br>Select an existing DIS stream. Common streams and advanced streams are supported. If no DIS stream is available, click **Create Stream** to create one. For details, see **Creating a DIS Stream**. | dis-function |
| Starting Position | Mandatory.<br>Specify a position in the specified stream from which to start reading data. For details, see **Obtaining Data Cursors**.<br>Options:<br>• **TRIM_HORIZON**: Data is read from the earliest valid records that are stored in the partition.<br>For example, a tenant used a DIS stream to upload three pieces of data A1, A2, and A3. N days later, A1 has expired and A2 and A3 are still in the validity period. In this case, if the tenant uses TRIM_HORIZON to download the data, the system downloads data from A2.<br>• **LATEST**: Data is read just after the most recent record in the partition. This setting ensures that you always read the latest data. | latest |
| Max. Fetch Bytes | Mandatory.<br>Maximum volume of data that can be fetched in each request. Only the records smaller than this value will be fetched. The unit can be KB or MB.<br>• For a common stream, the total size of records in a single request cannot exceed 1 MB (excluding partitionKey data). The value ranges from 0 KB to 1024 KB (0 MB to 1 MB).<br>• For an advanced stream, the total size of records in a single request cannot exceed 5 MB (excluding partitionKey data). The value ranges from 0 KB to 5,120 KB (0 MB to 5 MB). | 1 MB |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Pull Period | Mandatory.<br>The period for pulling data from the stream. The unit can be millisecond or second. The value ranges from 1 ms to 60,000 ms (1s to 60s). | 30s |
| Serial Data Processing | Mandatory.<br>• If this option is enabled, the system obtains data for the next processing only after the current data processing is complete.<br>• If this option is disabled, as shown in Figure 5-29, you can configure the concurrency (1–80) to limit the number of concurrent asynchronous invocation requests from a DIS trigger. This prevents a single DIS trigger with mass traffic from occupying the concurrency quota and affecting other DIS triggers. (Currently available only in **CN North-Beijing4**)<br>**Figure 5-29** Disabling serial data processing<br>  <br> | Enabled |

**Step 5** Click **OK**.

**----End**

## Modifying a DIS Trigger

Some parameters of DIS triggers can be modified.

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers**, and click **Edit** next to a DIS trigger.

**Step 4** Modify **Max. Fetch Bytes**, **Pull Period**, and **Serial Data Processing** as required, and click **OK**.

**----End**

### Configuring a DIS Test Event to Trigger a Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-50** and click **Save**.

**Table 5-50** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **Data Ingestion Service (DIS)** to use the built-in DIS event template. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **dis-123test**. |
| Event data | The system automatically loads the built-in DIS event template, which is used in this example without modifications. |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

### Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

## 5.6.8 DMS (for Kafka) Trigger

This section describes how to create a Kafka trigger to enable FunctionGraph to periodically poll new messages in a specified topic of a Kafka instance and pass the messages as input parameters to invoke functions.

For details about the Kafka event source, see **Supported Event Sources**.

NOTE

For details about the differences between DMS for Kafka and open-source Kafka, see **Comparing DMS for Kafka and Open-Source Kafka**.

## Notes and Constraints

- The regions and runtimes supported for Kafka triggers are subject to the console.
- The valid payload size of a request body is 6 MB when a Kafka trigger is used.
- In cases of Kafka data processing failure, the Kafka trigger will discard records that are larger than 6 MB.

## Prerequisites

- **Function and configuration:**
  - You have created a function.
  - You have configured DMS agency permissions for the function. For details about how to create an agency, see **Configuring Agency Permissions**.
  - You have enabled VPC access for the function. For details, see **Configuring Networks**.
- **DMS for Kafka:**
  - You have created a Kafka instance. For details, see **Buying an Instance**.
  - You have created a topic under a Kafka instance. For details, see **Creating a Topic**.
  - You have configured the subnet permissions for the Kafka security group. For details, see .

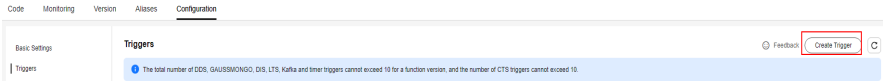
## Creating a Kafka Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-30** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-51** Parameters of a Kafka trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Select **DMS (for Kafka)**. | DMS (for Kafka) |
| Instance | Select a created Kafka instance. If no instance is available, click **Create Instance**. | kafka-fg |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Topic | Select a topic of the Kafka premium instance. You can create a Kafka trigger with multiple topics. You do not need to create one such trigger for each topic in the same instance. If no topic is available, click **Create Topic**. | topic-fg |
| Batch Size | Enter the number of messages consumed from the topic each time. The value ranges from 1 to 1000. | 100 |
| Username | This parameter is mandatory when SSL is enabled for the Kafka instance. Username used for connecting to the Kafka premium instance. | user |
| Password | This parameter is mandatory when SSL is enabled for the Kafka instance. Password used for connecting to the Kafka premium instance. | *** |

**Step 5** Click **OK**.

**----End**

## Configuring a Kafka Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-52** and click **Save**.

**Table 5-52** Test event information

| Parameter | Description | Example Value |
| --- | --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. | Create new test event |
| Event Templates | Select **DMS (for Kafka)** to use the built-in Kafka event template. | DMS (for Kafka) |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. | kafka-123test |
| Event data | The system automatically loads the built-in Kafka event template, which is used in this example without modifications. | - |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

### Helpful Links

- Manage function triggers through APIs. For details, see **Function Trigger APIs**.
- FAQ about Kafka triggers: **Can I Configure a Kafka Trigger in a Different Subnet from My Function?**

## 5.6.9 Kafka (Open-Source) Trigger

This section describes how to create an open-source Kafka trigger to enable FunctionGraph to periodically poll new messages in a specified topic of a Kafka instance and pass the messages as input parameters to invoke functions.

NOTE

For details about the differences between DMS for Kafka and open-source Kafka, see **Comparing DMS for Kafka and Open-Source Kafka**.

### Notes and Constraints

- The regions and runtimes supported for open-source Kafka triggers are subject to the console.
- The valid payload size of a request body is 6 MB when an open-source Kafka trigger is used.
- In cases of Kafka data processing failure, the Kafka trigger will discard records that are larger than 6 MB.

### Prerequisites

- You have created a function.
- You have enabled the VPC access for the function and configured the subnet permissions for the Kafka security group. For details, see **Configuring Networks**. The network configuration must be the same as that of the ECS where Kafka is deployed, including the VPC and subnet.

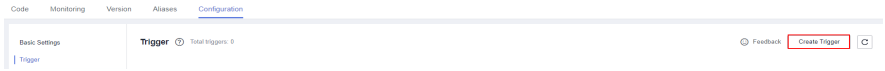
## Creating an Open-Source Kafka Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-31** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-53** Parameters of an open-source Kafka trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **Kafka (Open-Source)**. | Kafka (Open-Source) |
| Connection Address | Mandatory.<br>Addresses of brokers running Kafka. Separate the addresses with commas (,). | 100.85.125.151:9094,100.95.145.47:9094 |
| Topic | Mandatory.<br>Enter the created topic. | topic-1 |
| Batch Size | Mandatory.<br>Enter the maximum number of data records that can be processed by a function at a time. The value ranges from 1 to 10000. | 100 |

**Step 5** Click **OK**.

**----End**

## Enabling a Kafka Trigger

By default, open-source Kafka triggers are disabled. To use such a trigger, click **Enable** on the **Trigger** page.

NOTE

If the trigger fails to be started, **submit a service ticket** to contact technical support.

## Configuring a Kafka Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-54** and click **Save**.

**Table 5-54** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **Kafka (Open-Source)** to use the built-in Kafka event template. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **kafka-123test**. |
| Event data | The system automatically loads the built-in Kafka event template, which is used in this example without modifications. |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.10 DMS (for RabbitMQ) Trigger

This section describes how to create a RabbitMQ trigger (only the fan-out exchange mode is supported) to enable FunctionGraph to periodically poll for new messages in a specific topic bound to the exchange of a RabbitMQ instance and pass the messages as input parameters to invoke functions.

For details about the RabbitMQ event source, see **Supported Event Sources**.

## Notes and Constraints

The regions and runtimes supported for RabbitMQ triggers are subject to the console.

## Prerequisites

- **Function and configuration:**

  - You have created a function.
  - You have configured DMS agency permissions for the function. For details about how to create an agency, see **Configuring Agency Permissions**.
  - You have enabled the VPC access for the function and configured the subnet permissions for the RabbitMQ security group. For details, see **Configuring Networks**.

- **DMS for RabbitMQ:**
  - A RabbitMQ instance has been created. For details, see **Buying an Instance**.
  - A virtual host, exchange, and queue have been created.
    i. To create a virtual host, see **Creating a RabbitMQ Virtual Host**.
    ii. To create an exchange, see **Creating a RabbitMQ Exchange**.
    iii. To create a queue, see **Creating a RabbitMQ Queue**.
    iv. An exchange-queue binding has been configured. For details, see **Binding a RabbitMQ Exchange** and **Binding a RabbitMQ Queue**.

    NOTE

    Virtual hosts (vhost) serve as independent RabbitMQ servers to manage exchanges and queues. A RabbitMQ instance can have multiple virtual hosts, and a virtual host can have multiple exchanges and queues. For details, see **Process of Using RabbitMQ**.
  - The rules of the security group of the instance have been correctly configured.
    i. In the **Network** section on the **Basic Information** tab page, click the name of the security group.
    ii. Click the **Inbound Rules** tab to view the inbound rules of the security group.
      1) SSL disabled

      For intra-VPC access, inbound access through port 5672 must be allowed.

      For public access, inbound access through port 15672 must be allowed.
      2) SSL enabled

      For intra-VPC access, inbound access through port 5671 must be allowed.

      For public access, inbound access through port 15671 must be allowed.

## Creating a RabbitMQ Trigger

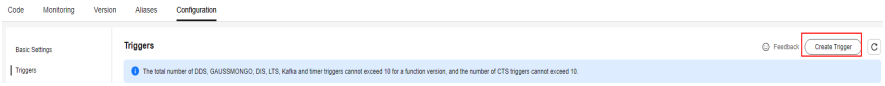
**Step 1** Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-32** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-55** RabbitMQ trigger parameters

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **DMS (for RabbitMQ)**. | DMS (for RabbitMQ) |
| Instance | Mandatory.<br>Select a created RabbitMQ instance. If no instance is available, click **Create Instance**. | rabbitmq-fg |
| Password | Mandatory.<br>Enter the password of the created RabbitMQ instance. | testrabbitmq |
| Exchange | Mandatory.<br>Enter the name of the created exchange. For details, see **ii**. | rabbitmqEX |
| Virtual Host | Optional.<br>Enter the name of the virtual host you have created. For details, see **i**. | test |
| Batch Size | Mandatory.<br>Enter the number of messages consumed from the topic each time. The value ranges from 1 to 1000. | 100 |

**Step 5** Click **OK**.

**----End**

## Configuring a DMS (for RabbitMQ) Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the function to be configured. The function details page is displayed.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-56** and click **Save**.

**Table 5-56** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **DMS (for RabbitMQ)** to use the built-in RabbitMQ event template. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **rabbitmq-123test**. |
| Event data | The system automatically loads the built-in RabbitMQ event template, which is used in this example without modifications. |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.11 GeminiDB Mongo Trigger (Offline Soon)

This section describes how to create a GeminiDB Mongo trigger on the FunctionGraph console. The trigger triggers a function each time a table in the GeminiDB Mongo is updated.

**GeminiDB Mongo triggers are about to be brought offline. You are advised not to create new ones.**

## Notes and Constraints

- The regions and runtimes supported for GeminiDB Mongo triggers are subject to the console.
- The valid payload size of a request body is 6 MB when a GeminiDB Mongo trigger is used.

## Prerequisites

- **Function and configuration:**
  - You have created a function.
  - You have configured the GeminiDB agency permission for the function. For details about how to create an agency, see **Configuring Agency Permissions**.
  - You have enabled VPC access for the function. For details, see **Configuring Networks**.
- **GeminiDB Mongo:**

- An available GeminiDB Mongo instance is available.
- You have configured the subnet permissions for the GeminiDB Mongo security group. For details, see .

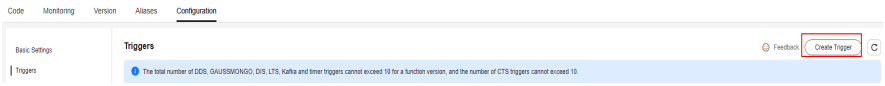
## Creating a GeminiDB Mongo Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-33** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-57** Parameters of a GeminiDB trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **GeminiDB Mongo**. | GeminiDB Mongo |
| GeminiDB Mongo Instance | Mandatory.<br>Select a created GeminiDB Mongo instance. If no instance is available, click **Create Instance**. | geminidb-fg |
| Password | Mandatory.<br>Enter the password of the GeminiDB Mongo instance administrator **rwuser**. | GeminiDB@123 |
| Database | Mandatory.<br>Set the database name. Note that **admin**, **local**, and **config** are reserved database names and cannot be used here. | GeminiDB-test |
| Collection | Mandatory.<br>Enter the database collection name. | GeminiDB-set |
| Batch Size | Mandatory.<br>Enter the number of records read from the database in each batch. The value ranges from 1 to 10000. | 100 |

**Step 5** Click **OK**.

**----End**

### Configuring a GeminiDB Mongo Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the function to be configured. The function details page is displayed.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-58** and click **Save**.

**Table 5-58** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **GeminiDB Mongo**. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **Gemini-123test**. |
| Event data | The system automatically loads the built-in GeminiDB Mongo event template, which is used in this example without modifications. |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

### Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

## 5.6.12 IoT Device Access (IoTDA) Trigger

This section describes how to create an IoTDA trigger on the FunctionGraph console.

For details about the IoTDA event source, see **Supported Event Sources**.

### Notes and Constraints

The regions and runtimes supported for IoTDA triggers are subject to the console.

## Prerequisites

- You have created a function.
- You have created an IoTDA instance. For details, see **Buying an IoTDA Instance**.
- You have created a resource space for the IoTDA instance. For details, see **Creating a Resource Space**.

## Creating an IoTDA Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Step 4** Configure the following parameters.

- **Trigger Type**: Select **IoT Device Access (IoTDA)**.
- **Instance**: Select the created IoTDA instance.
- **Data Source**, **Trigger**, and **Resource Space**: For details, see **Table 5-59**.

**Table 5-59** Configuration

| Data Source | Trigger | Resource Space |
| --- | --- | --- |
| Device | Device added, Device deleted, Device updated | Select a resource space. Multiple resource spaces can be created and one of them is specified as the default resource space. Each account has only one default resource space, which cannot be deleted. |
| Device property | Device property reported | |
| Device message | Device message reported | |
| Device message status | Device message status changed | |
| Device status | Device status changed | |
| Product | Product added, Product deleted, and Product updated | |
| Asynchronous command status of the device | Batch task status changed | |
| Run log | Log reported | |
| Batch task | Batch task status changed | |

**Step 5** Click **OK**.

**----End**

## Triggering the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the function to be configured. The function details page is displayed.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters listed in **Table 5-60** and click **Create**.

**Table 5-60** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event (default) or edit an existing one. |
| Event Templates | Select **Blank Template**. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **iotda-123test**. |
| Event data | {<br>"resource" : "device",<br>"event" : "create",<br>"event_time" : "20240919T011335Z",<br>"event_time_ms" : "2024-09-19T01:13:35.854Z",<br>"request_id" : "75127474-1a26-4578-8847-3128d6101954",<br>"notify_data" : {<br>"body" : {<br>"app_id" : "3d40caf3ddfc4e83815b54b50f13aad7",<br>"app_name" : "DefaultApp_6439vdv2",<br>"device_id" : "66eb7a0ffa8d9c36870c6892_ttytytytytytyt",<br>"node_id" : "ttytytytytytyt",<br>"gateway_id" : "66eb7a0ffa8d9c36870c6892_ttytytytytytyt",<br>"node_type" : "GATEWAY",<br>"auth_info" : {<br>"auth_type" : "SECRET",<br>"secure_access" : false,<br>"timeout" : 0<br>},<br>"product_id" : "66eb7a0ffa8d9c36870c6892",<br>"product_name" : "test",<br>"status" : "INACTIVE",<br>"create_time" : "20240919T011335Z"<br>}<br>}<br>} |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.13 Log Tank Service (LTS) Trigger

This section describes how to create an LTS trigger on the FunctionGraph console.

## Event Description

LTS collects and stores logs, allowing you to query them in real time. If you create a function with an LTS trigger, subscribed logs collected by LTS will be passed as a parameter to invoke the function. Then, the function processes or analyzes the logs, or loads the logs to other systems.

Example event of an LTS trigger

- LTS triggers functions using the following event format. For details about the parameters, see **Table 5-61**.

```
{
    "lts": {
        "data":
"eyJsb2dzIjoiW3tcIm1lc3NhZ2VcIjpcIjIwMTgtMDgtMDgvMDg6MDg6MDggW1dSTl0gW3Rlc3QuZ286MDhdVGhpcyBpcyBhIHRlc3QgbWVzc2FnZS5cIixcInRpbWVcIjoxNTMwMDA5NjUzMDU5LFwiaG9zdF9uYW1lXCI6XCJlY3MtdGVzdFwiLFwiaXBcIjpcIjE5Mi4xNjguMS4xXCIsXCJwYXRoXCI6XCJ2YXIvbG9nL3Rlc3QubG9nXCIsXCJsb2dfdWlkXCI6XCI2NjNkNjkzMC03OTJkLTExZTgtOGIwOC0yODZlZDQ4OGNlNzBcIixcImxpbmVfbm9cIjoxfV0iLCJvd25lciI6IjYyODBlMTcwYmQ5MzRmNjBhNGQ4NTFjZjVjYTA1MTI5IiwibG9nX2dyb3VwX2lkIjoiOTdhOWQyODQtNDQ0OC0xMWU4LThmYTQtMjg2ZWQ0ODhjZTcwIiwibG9nX3RvcGljX2lkIjoiMWE5Njc1YTctNzg0ZC0xMWU4LTlmNzAtMjg2ZWQ0ODhjZTcwIn0="
    }
}
```

**Table 5-61** Parameters of an LTS example event

| Parameter | Type | Example Value | Description |
| --- | --- | --- | --- |
| data | String | Reference example code | Base64-encoded data |

## Notes and Constraints

- The regions and runtimes supported for LTS triggers are subject to the console.
- When the size of an LTS event message exceeds 75 KB, it will be split into multiple messages by 75 KB to trigger the function.

## Prerequisites

- **Function and configuration:**
  - You have created a function.
  - You have created an agency with the **LTS FullAccess** permission. For details about how to create an agency, see **Configuring Agency Permissions**.
- **LTS:**

- You have created a log group, for example, **LogGroup1**. For details, see **Creating a Log Group**.
- You have created a log stream, for example, **LogTopic1**. For details, see **Creating a Log Stream**.
- You have installed and configured an agent for collecting logs from servers, such as ECSs, to a specified log group. For details, see **Installing the ICAgent**.

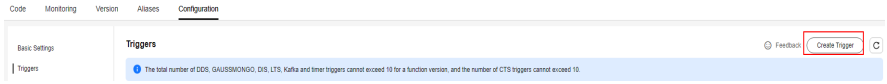
## Creating an LTS Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-34** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **Log Tank Service (LTS)**.
- **Log Group**: Select a log group, for example, **LogGroup1**.
- **Log Stream**: Select a log stream, for example, **LogStream1**.

**Step 5** Click **OK**.

**----End**

## Configuring an LTS Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-62** and click **Save**.

**Table 5-62** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |

| Parameter | Description |
| --- | --- |
| Event Templates | Select **Log Tank Service (LTS)** and use the built-in LTS event template. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **lts-123test**. |
| Event data | The system automatically loads the built-in LTS event template, which is used in this example without modifications. |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.14 Simple Message Notification (SMN) Trigger

This section describes how to create an SMN trigger on the FunctionGraph console to trigger function execution when a message is published.

For details about the SMN event source, see **Supported Event Sources**.

## Notes and Constraints

- The regions and runtimes supported for SMN triggers are subject to the console.
- SMN triggers cannot be disabled and can only be deleted.

## Prerequisites

- You have created a function.
- You have created an SMN topic, for example, **smn-test**. For details, see **Creating a Topic**.

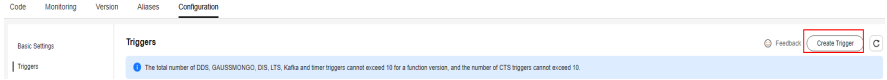
## Creating an SMN Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-35** Creating a trigger



**Step 4** Set the following parameters:

- **Trigger Type**: Select **Simple Message Notification (SMN)**.
- **Topic Name**: Select a topic, for example, **smn-test**.

**Step 5** Click **OK**.

NOTE

After the SMN trigger is created, a subscription is generated for the corresponding topic on the SMN console.

**----End**

### Publishing a Message to Trigger the Function

On the SMN console, publish a message to the **smn-test** topic. For details, see **Publishing a Text Message**. After the message is published, the function is automatically triggered.

**Table 5-63** describes the parameters required for publishing a message.

**Table 5-63** Parameters required for publishing a message

| Parameter | Description |
| --- | --- |
| Subject | Enter **SMN-Test**. |
| Message Format | Select **Text**. |
| Message | Enter **{"message":"hello"}**. |

### Viewing the Execution Result

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click a function to go to the function details page.

**Step 3** Choose **Monitoring** > **Logs** to query function running logs.

**----End**

### Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

## 5.6.15 Object Storage Service (OBS) Trigger

This section describes how to create an OBS trigger on the FunctionGraph console to trigger a function when a file in an OBS bucket is updated.

For details about the OBS event source, see **Supported Event Sources**.

## Supported Event Types

Multiple event types can be applied to the same target object.

Select the event type to be used by referring to **Table 5-64**.

**Table 5-64** Event types supported by OBS

| Event | Description |
| --- | --- |
| ObjectCreated | All kinds of object creation operations, including PUT, POST, COPY, and CompleteMultipartUpload.<br>**Constraints**: If you select **ObjectCreated**, the others are automatically selected and cannot be selected again. |
| Put | Use the PUT method to upload objects. |
| Post | Use the POST method to upload objects. |
| Copy | Use the COPY method to replicate objects. |
| CompleteMultipar-tUpload | Merge parts of multi-part tasks. |
| ObjectRemoved | Indicate all object deletion operations, including **Delete** and **DeleteMarkerCreated**.<br>**Constraints**: If you select **ObjectRemoved**, **Delete** and **DeleteMarkerCreated** are automatically selected and cannot be selected again. |
| Delete | Delete objects by versions. |
| DeleteMarkerCre-ated | Delete objects without specifying versions. |

## Notes and Constraints

- The regions and runtimes supported for OBS triggers are subject to the console.
- OBS triggers cannot be disabled and can only be deleted.
- Ensure that the created function and the OBS bucket are in the same region. Buckets created in different regions are not shared.

## Prerequisites

- **Permission**

  If OBS is used as the event source, you must have the Tenant Administrator permission. For details, see **Assigning Permissions to an IAM User**.
- **Function and its configuration**
  - You have created a function.
  - You have configured the OBS agency permission for the function. For details about how to create an agency, see **Configuring Agency Permissions**.

- **OBS bucket**

  You have created an OBS bucket. The following uses the **eventbucket** bucket as an example. For details, see **Creating a Bucket**.

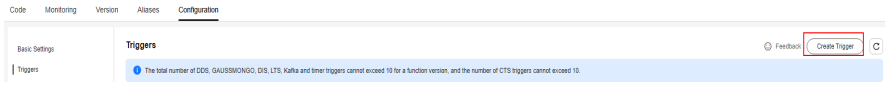
## Creating an OBS Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-36** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-65** Parameters of an OBS trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **Object Storage Service (OBS)**. | Object Storage Service (OBS) |
| Bucket Name | Mandatory.<br>OBS bucket used as the event source. After the bucket is created, it cannot be modified. | obs-cff |
| Events | Mandatory.<br>Select events that will trigger the function. For details about the supported event types, see **Table 5-64**. | **Put**, **Post**, and **Delete**<br>(The function is triggered when files in the OBS bucket are updated, uploaded, or deleted.) |
| Event Notification Name | Optional.<br>Specify the name of the event notification to be sent by SMN when an event occurs. | obs-event-test |
| 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. | - |

| Parameter | Description | Example Value |
| --- | --- | --- |
| 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. | - |

**----End**

### Triggering a Function

On the OBS console, upload a file to the **obs-cff** bucket. For details, see **Uploading a File**.

After the file is uploaded to the **obs-cff** bucket, the function is automatically triggered. You can view the execution result in the function execution logs.

### Viewing the Execution Result

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click a function to go to the function details page.

**Step 3** Choose **Monitoring** > **Logs** to query function running logs.

**----End**

### Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

## 5.6.16 EventGrid Trigger (OBS Application Service)

This section describes how to create an EG trigger (OBS Application Service) on the FunctionGraph console. The EG trigger triggers a function when a file in an OBS bucket is updated.

You can configure OBS Application Service triggers for shared functions in the **LA-Sao Paulo1** region.

For details about the EG event source, see **Supported Event Sources**.

### Notes and Constraints

- The regions and runtimes supported for EventGrid triggers are subject to the console.
- Ensure that the created function and the OBS bucket are in the same region.

## Prerequisites

- **Permission**

  If OBS is used as the event source, you must have the Tenant Administrator permission. For details, see **Assigning Permissions to an IAM User**.
- **Function and configuration:**
  - You have created a function.
  - You have configured the OBS agency permission for the function. For details about how to create an agency, see **Configuring Agency Permissions**.
- **OBS bucket:**

  You have created an OBS bucket. The following uses the **eventbucket** bucket as an example. For details, see **Creating a Bucket**.

## Creating an EG Trigger (OBS Application Service)

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of the function to be configured. The function details page is displayed.
3. Choose **Configuration** > **Triggers** and click **Create Trigger**.

   **Figure 5-37** Creating a trigger

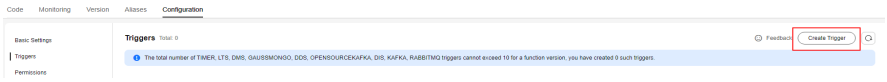
   
   
4. Configure the following parameters.

   **Table 5-66** Parameters for creating an EG trigger (OBS Application Service)

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **OBS Application Service**. | OBS Application Service |
| Trigger Name | Mandatory.<br>Name of a trigger. Only letters, digits, underscores (_), and hyphens (-) are allowed. The value cannot start with a digit or hyphen (-). Maximum length: 128 characters. | eg-obs |
| Bucket Name | Mandatory.<br>Select an OBS bucket. If no OBS bucket is available, click **Create Bucket**. | eventbucket |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Event Type | Mandatory.<br>Select the required trigger event type. Options:<br>• Delete objects without specifying version<br>• Delete objects by version<br>To select this event type, you must enable versioning. For details, see **Deleting Objects from a Bucket with Versioning Enabled**.<br>• Create or override bucket objects via UI or Put request<br>• Merge parts via UI or API request<br>• Create or override bucket objects via Copy request<br>• Create or override bucket objects via Post request | Create or override bucket objects via UI or Put 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. | - |
| 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. | - |
| Object Name Encoding | Mandatory.<br>Specifies whether to encode the object name. This option is enabled by default. | Enabled |

5. Click **OK**.

## Configuring an EG Trigger (OBS Application Service) for a Shared Function

You can configure an EG trigger (OBS Application Service) for a shared function in the **LA-Sao Paulo1** region. For details about the shared function, see **Sharing Functions Based on RAM**.

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. On the **Shared** tab, click the function name to go to the function details page.

3. The subsequent steps are the same as those for creating a common function. For details, see **3**.

## Configuring an EG Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-67** and click **Save**.

**Table 5-67** Test parameters

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **Blank Template**. For details about the code, see "OBS Application Service" in **Supported Event Sources**. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **myobs-123test**. |
| Event data | Use the newly created test event. |

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.17 DMS (for HC .RocketMQ) Trigger (Only for Existing Users)

This section describes how to create a DMS (for HC.RocketMQ) trigger on the FunctionGraph console to trigger function execution using RocketMQ.

**EventGrid triggers (RocketMQ custom event sources) cannot be created. Only existing users can use them.**

## Notes and Constraints

The regions and runtimes supported for EventGrid triggers are subject to the console.

## Prerequisites

- **Function and configuration:**
  - You have created a function.
  - You have enabled VPC access for the function. For details, see **Configuring VPC Access**.
- **EG:**

  You have created an EG event channel. For details, see **Creating an EG Event Channel**.
- **RocketMQ instance:**
  - You have created a RocketMQ instance. For details, see **Buying an Instance**.
  - You have created a RocketMQ topic. For details, see **Creating a Topic**.
  - You have created a RocketMQ consumer group. For details, see **Creating a Consumer Group**.

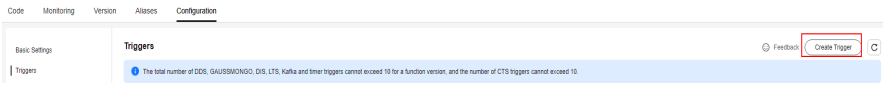
## Creating a DMS (for HC.RocketMQ) Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-38** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-68** Parameters for creating a DMS (for HC.RocketMQ) trigger

| Parameter | Parameter | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **DMS (for HC.RocketMQ)**. | DMS (for HC.RocketMQ) |
| Trigger Name | Mandatory.<br>Name of a trigger. Only letters, digits, underscores (_), and hyphens (-) are allowed. The value cannot start with a digit or hyphen (-). Maximum length: 128 characters. | EG-RocketMQ |

| Parameter | Parameter | Example Value |
| --- | --- | --- |
| Event Channel | Mandatory.<br>Select an existing EG event channel. If no event channel is available, click **Create Event Channel**. | EGtest |
| Instance | Mandatory.<br>Select a created RocketMQ instance. If no instance is available, click **Create Instance**. | rocketmq-fg |
| Topic | Mandatory.<br>Select a created RocketMQ topic. | topic-test |
| Consumer Group | Mandatory.<br>Select a RocketMQ consumer group. | fgtest |

**Step 5** Click **OK**.

**----End**

## Configuring an EG Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-69** and click **Save**.

**Table 5-69** Test parameters

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **DMS (for HC.RocketMQ)** and use the built-in HC.ROCKETMQ event template. |

| Parameter | Description |
| --- | --- |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **rocketmq-123test**. |
| Event data | The system automatically loads the built-in eg-RocketMQ event template, which is used in this example without modifications. |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.18 DMS (for HC .RabbitMQ) Trigger (Only for Existing Users)

This section describes how to create a DMS (for HC.RabbitMQ) trigger on the FunctionGraph console to trigger function execution using RabbitMQ.

**This type of trigger is available only to existing users.**

## Notes and Constraints

The regions and runtimes supported for EventGrid triggers are subject to the console.

## Prerequisites

- **Function and configuration:**
  - You have created a function.
  - You have enabled the VPC access for the function and configured the subnet permissions for the RabbitMQ security group. For details, see **Configuring Networks**.
- **EG:**

  You have created an EG event channel. For details, see **Creating an EG Event Channel**.
- **RabbitMQ instance:**
  - You have created a RabbitMQ instance. For details, see **Buying an Instance**.
  - A virtual host, exchange, and queue have been created.

i. To create a virtual host, see **Creating a RabbitMQ Virtual Host**.
ii. To create an exchange, see **Creating a RabbitMQ Exchange**.
iii. To create a queue, see **Creating a RabbitMQ Queue**.
iv. An exchange-queue binding has been configured. For details, see **Binding a RabbitMQ Exchange** and **Binding a RabbitMQ Queue**.

NOTE

Virtual hosts (vhost) serve as independent RabbitMQ servers to manage exchanges and queues. A RabbitMQ instance can have multiple virtual hosts, and a virtual host can have multiple exchanges and queues. For details, see **Process of Using RabbitMQ**.

- The rules of the security group of the instance have been correctly configured.
  i. In the **Network** section on the **Basic Information** tab page, click the name of the security group.
  ii. Click the **Inbound Rules** tab to view the inbound rules of the security group.
    1) SSL disabled

       For intra-VPC access, inbound access through port 5672 must be allowed.

       For public access, inbound access through port 15672 must be allowed.
    2) SSL enabled

       For intra-VPC access, inbound access through port 5671 must be allowed.

       For public access, inbound access through port 15671 must be allowed.

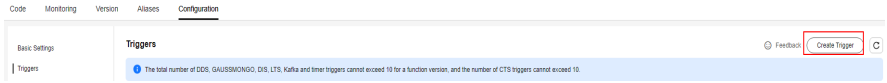
## DMS (for HC.RabbitMQ) Trigger

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

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 5-39** Creating a trigger




**Step 4** Configure the following parameters.

**Table 5-70** Parameters for creating a DMS (for HC.RabbitMQ) trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **DMS (for HC.RabbitMQ)**. | DMS (for HC.RabbitMQ) |
| Trigger Name | Mandatory.<br>Name of a trigger. Only letters, digits, underscores (_), and hyphens (-) are allowed. The value cannot start with a digit or hyphen (-). Maximum length: 128 characters. | EG-RabbitMQ |
| Event Channel | Mandatory.<br>Select an existing EG event channel. If no event channel is available, click **Create Event Channel**. | EGtest |
| Instance | Mandatory.<br>Select a created RabbitMQ instance. If no instance is available, click **Create Instance**. | rabbitmq-fg |
| Username | Mandatory.<br>Enter the username used for accessing the instance. | fgtest |
| Password | Mandatory.<br>Enter the password used for accessing the instance. | testrabbitmq |
| Virtual Host | Mandatory.<br>Enter the virtual host name of the RabbitMQ instance. | rabbitmqEX |
| Queue | Mandatory.<br>Enter the queue of the RabbitMQ instance. | rabbitmqQueue |

**Step 5** Click **OK**.

**----End**

## Configuring an EventGrid Event to Trigger the Function

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the function to be configured to go to the function details page.

**Step 3** On the function details page, select a version.

**Step 4** On the **Code** tab page, click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 5** Set the parameters described in **Table 5-71** and click **Save**.

**Table 5-71** Test parameters

| Parameter | Description |
| --- | --- |
| Configure Test Event | You can choose to create a test event or edit an existing one.<br>Use the default option **Create new test event**. |
| Event Templates | Select **DMS (for HC.RabbitMQ)** to use the built-in RabbitMQ event template. |
| Event Name | The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **rabbitmq-123test**. |
| Event data | The system automatically loads the built-in eg-RabbitMQ event template, which is used in this example without modifications. |

**Step 6** Click **Test**. The function test result is displayed.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.6.19 Managing Function Triggers

This section describes how to view and manage triggers created on the FunctionGraph console.

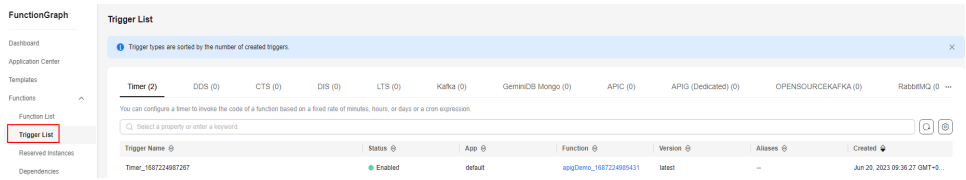
## Viewing Triggers

Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Trigger List**. You can view all created triggers, as shown in **Figure 5-40**. The type of trigger in use will be preferentially displayed.

Click the link in the **Function** column to go to the function details page and disable, enable, or delete the trigger.

**Figure 5-40** Trigger display




## Enabling or Disabling a Trigger

You can enable or disable a trigger as required. **APIG and SMN triggers cannot be disabled and can only be deleted.**

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

**Step 2** Click the name of the desired function.

**Step 3** Choose **Configuration** > **Triggers**. On the displayed page, locate the row that contains the target trigger, and click **Disable** or **Enable**.

**----End**

## Deleting a Trigger

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the desired function.

**Step 3** Choose **Configuration** > **Triggers**. On the displayed page, locate the row that contains the target trigger and click **Delete**.

**----End**

## Helpful Links

Manage function triggers through APIs. For details, see **Function Trigger APIs**.

# 5.7 Debugging Functions Online

This section describes how to manage test events on the FunctionGraph console and debug functions online using test events.

## Scenarios

After the function configuration is complete, you can customize test events on the **Code** tab page of the function details page to verify whether the function can be executed successfully.

## Notes and Constraints

Each function can have a maximum of 10 test events.

## Creating a Test Event

After a test event is configured and saved, it can be used for testing again.

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

**Step 2** Click the name of the desired function.

**Step 3** On the function details page, select a version, and click **Test**. The **Configure Test Event** dialog box is displayed.

**Step 4** In the displayed dialog box, enter the test event information by referring to **Table 5-72**.

**Table 5-72** Test event information

| Parameter | Description |
| --- | --- |
| Configure Test Event | Mandatory.<br>You can create a test event or edit a saved test event. Select **Create new test event**. |
| Event Templates | Mandatory.<br>• If you select **Blank Template**, you can create a test event from scratch.<br>• If you select other common or cloud event templates, the corresponding test events are automatically loaded. **Table 5-73** describes the event templates. |
| Event Name | Mandatory.<br>The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **event-123test**. |
| Event data | Mandatory.<br>Edit the test event code in the code editing area. |

**Table 5-73** Event template description

| Template Name | Description |
| --- | --- |
| API Gateway (Dedicated Gateway) | Simulates a dedicated APIG event to trigger your function. |
| Cloud Trace Service (CTS) | Simulates a CTS event to trigger your function. |
| Document Database Service (DDS) | Simulates a DDS event to trigger your function. |

| Template Name | Description |
| --- | --- |
| GeminiDB Mongo | Simulates a GeminiDB Mongo event to trigger your function. |
| Data Ingestion Service (DIS) | Simulates a DIS event to trigger your function. |
| Log Tank Service (LTS) | Simulates an LTS event to trigger your function. |
| Simple Message Notification (SMN) | Simulates an SMN event to trigger your function. |
| Timer | Simulates a timer event to trigger your function. |
| DMS (for Kafka) | Simulates a Kafka event to trigger your function. |
| Kafka (Open-Source) | Simulates an open-source Kafka event to trigger your function. |
| DMS (for RabbitMQ) | Simulates a RabbitMQ event to trigger your function. |
| Blank Template | The event is **{"key": "value"}**, which can be changed based on requirements. |
| Login Security Analysis | Serves as an input for the **loginSecurity-realtime-analysis-python** function template. |
| Image Classification | Serves as an input for the image classification. |
| Pornographic Image Analysis | Serves as an input for the pornographic image analysis. |
| Speech Recognition | Serves as an input for the voice analysis. |

**Step 5** Click **Create**.

**----End**

## Testing a Function

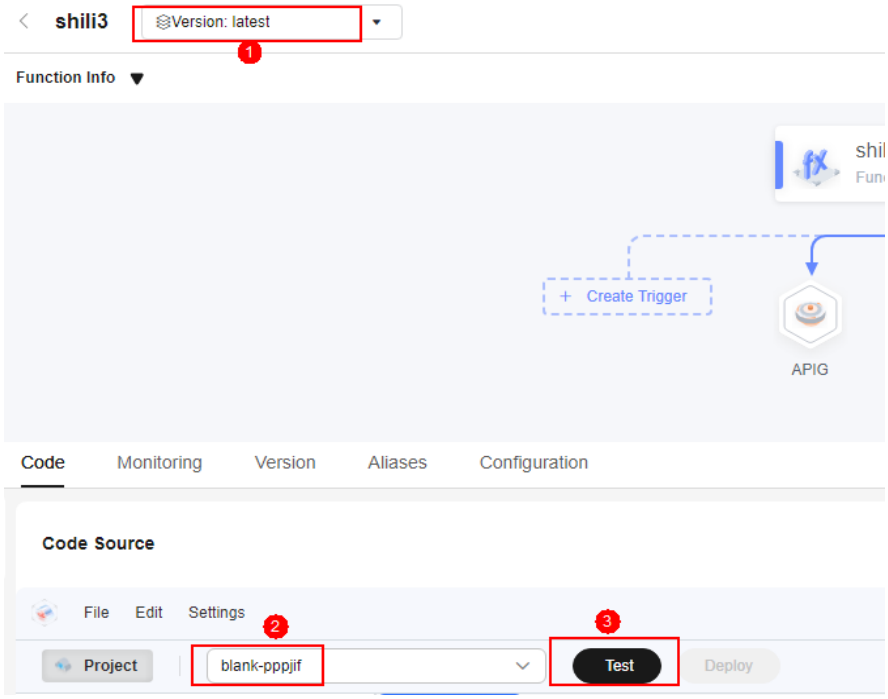
**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the desired function.

**Step 3** On the displayed function details page, select a version and test event, and click **Test** as shown in **Figure 5-41**.

**Figure 5-41** Selecting a test event




**Step 4** Click **Test**. The **Execution Result** window is displayed on the **Code** tab page.

You can view function logs in **Execution Result**. A maximum of 2 KB logs can be displayed. For details about how to view complete logs, see **Configuring Log Groups and Log Streams, and Viewing Function Logs**.

**----End**

## Modifying a Test Event

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the desired function.

**Step 3** On the displayed function details page, click **Configure Test Event**. The **Configure Test Event** dialog box is displayed.

**Step 4** In the **Configure Test Event** dialog box, modify the test event information according to **Table 5-74**.

**Table 5-74** Test event information

| Parameter | Description |
| --- | --- |
| Create new test event | Create a test event. |
| Edit saved test event | Modify an existing test event. |
| Event data | Modify the test event code. |

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

**----End**

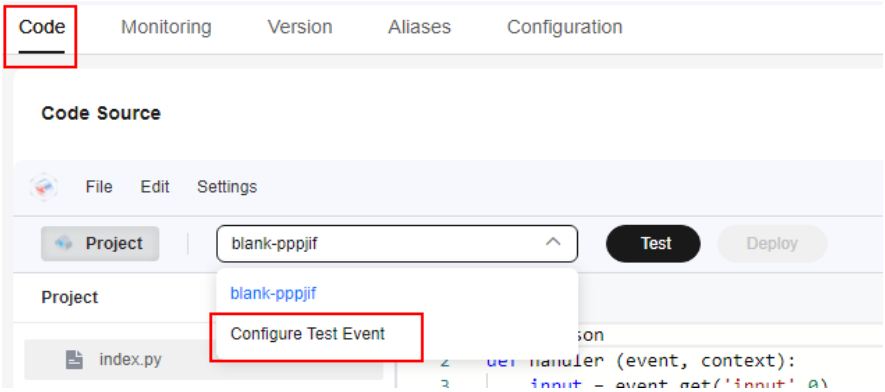
## Deleting a Test Event

**Step 1** Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.

**Step 2** Click the name of the desired function.

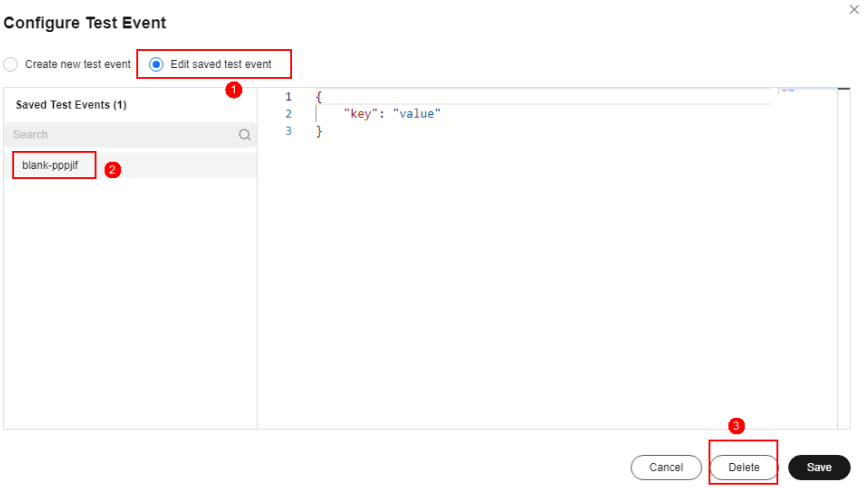
**Step 3** On the **Code** tab page, click **Configure Test Event**. The editing page is displayed, as shown in **Figure 5-42**.

**Figure 5-42** Selecting Configure Test Event




**Step 4** On the **Configure Test Event** page, select **Edit saved test event**. In the **Saved Test Events** list on the left, select the event to be deleted and click **Delete** as shown in **Figure 5-43**.

**Figure 5-43** Deleting a test event




**----End**

## Helpful Links

Manage test events through APIs. For details, see **Function Test Event APIs**.

# 6 Invoking the Function

After a function is created and configured, you can trigger the function execution through RESTful APIs or cloud service events.

- RESTful API: Functions are triggered by calling APIs.
- Cloud service events: Functions are triggered by cloud service triggers.

Functions can be invoked synchronously or asynchronously, depending on whether the client needs to wait for the processing result.

## Synchronous Invocation

Synchronous invocation: After a client invokes a function, FunctionGraph executes the function immediately and returns a response and execution result to the client.

**Scenarios:**

Synchronous invocation is suitable for scenarios that require real-time response.

**Example 1: Real-time data processing and query**

Use synchronous invocation to obtain real-time data (such as order status and payment result) and process requests and return results immediately.

**Example 2: Instant interaction and control**

Use synchronous invocation to obtain operation results (such as message sending success) immediately.

## Asynchronous Invocation

Asynchronous invocation: After a client invokes a function, FunctionGraph queues the requests and directly returns responses to the client without waiting for the function execution result. FunctionGraph processes the queued requests one by one when the server is idle.

In asynchronous invocation scenarios, the client cannot detect the function execution result in real time. For details about how to obtain the request result or set retry upon failure, see **Asynchronous Notification Policy**. You can also use the **asynchronous execution** API instead.

**Scenarios:**

Asynchronous invocation is suitable for tasks that can be processed later, do not require real-time response, and may take a long time or require a large number of resources. This approach enhances system responsiveness and throughput while ensuring reliable task execution.

**Example 1: media processing and conversion**

- Image compression and format conversion: After an image is uploaded, the system asynchronously processes the image in the background, compresses, crops, or converts the image format, and stores the processed image in a specified location.
- Video transcoding and editing: After a video is uploaded, the system asynchronously transcodes, edits, or adds watermarks to the video in the background. Asynchronous invocation frees up system resources during lengthy operations.

**Example 2: data processing and integration**

- ETL tasks (data cleaning and conversion): Extract data from databases periodically, cleanse, convert, and load the data to the target storage service or analysis platform. Reports are generated upon completion with notifications. Asynchronous invocation prevents system blocking and improves processing efficiency.
- Log analysis and statistics: After system logs are collected, log analysis, exception detection, and statistics are performed asynchronously in the background. Results are stored or sent to monitoring systems upon completion. Asynchronous invocation ensures that the system can run properly during peak hours.

## Notes and Constraints

- A function with synchronous invocation is executed for up to 15 minutes.
- A function with asynchronous invocation is executed for up to 3 days. For longer execution duration, **submit a service ticket**.

  You can customize the execution timeout on the function details page. For details, see **Basic Settings**. The execution timeout must be less than the maximum execution duration.
- If the end-to-end execution latency exceeds 90s, you are advised to use asynchronous invocation. If synchronous invocation is used, no responses can be received after 90s due to gateway restrictions.

## Function Trigger Invocation Mode

Triggers can be invoked synchronously or asynchronously. For details about function invocation, see **Invoking the Function**.

- Synchronous invocation: After a client invokes a function, FunctionGraph executes the function immediately and returns a response and execution result to the client.
- Asynchronous invocation: After a client invokes a function, FunctionGraph queues the requests and directly returns responses to the client without waiting for the function execution result. FunctionGraph processes the queued requests one by one when the server is idle.

**Table 6-1** Function trigger invocation

| Trigger | Invocation Mode |
| --- | --- |
| API Gateway (dedicated) | Synchronous. You can change it to asynchronous. For details, see **Configuring Asynchronous Invocation**. |
| API Connect (APIC) | Synchronous. You can change it to asynchronous. For details, see **Configuring Asynchronous Invocation**. |
| Timer | Asynchronous (default and cannot be changed) |
| GeminiDB DynamoDB | Synchronous (default and cannot be changed) |
| Cloud Trace Service (CTS) | Asynchronous (default and cannot be changed) |
| Document Database Service (DDS) | Asynchronous (default and cannot be changed) |
| Data Ingestion Service (DIS) | Asynchronous (default and cannot be changed) |
| DMS (for Kafka) | Asynchronous (default and cannot be changed) |
| Kafka (Open-Source) | Asynchronous (default and cannot be changed) |
| DMS (for RabbitMQ) | Asynchronous (default and cannot be changed) |
| GeminiDB Mongo | Asynchronous (default and cannot be changed) |
| Log Tank Service (LTS) | Asynchronous (default and cannot be changed) |
| Simple Message Notification (SMN) | Asynchronous (default and cannot be changed) |
| EventGrid (EG) | Asynchronous (default and cannot be changed) |

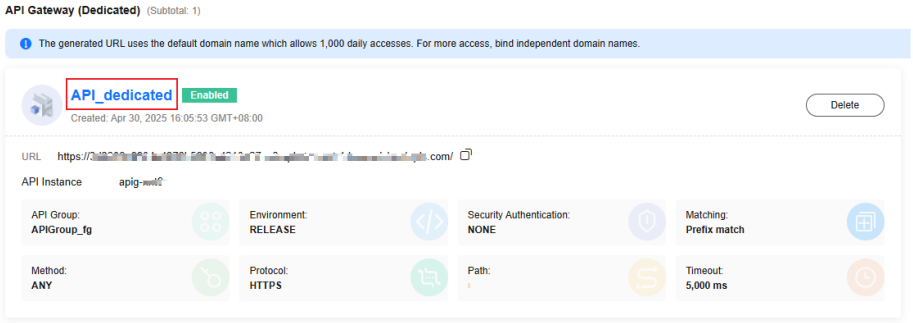
## Configuring Asynchronous Invocation

The following uses an APIG trigger as an example. Ensure you have created a function and **configured an APIG trigger**.

1. Log in to the **FunctionGraph console**, and choose **Functions** > **Function List** in the navigation pane. Click a function name to go to its details page. Choose **Configuration** > **Triggers**.
2. Click the APIG trigger name to go to the APIG console.

**Figure 6-1** Clicking a trigger name

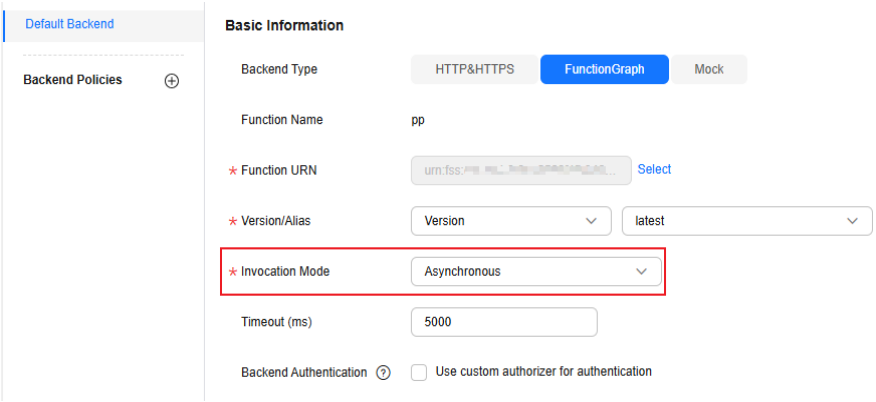



3. Click **Modify** in the upper right.

**Figure 6-2** Clicking Modify


4. Click **Next**. On the **Basic Information** page, change the **Invocation Mode** to **Asynchronous**.

**Figure 6-3** Changing the invocation mode




5. Click **Finish** to save the settings.

## Retry Mechanism

If synchronous or asynchronous invocation fails, do as follows:

- Synchronous invocation

  Try again.
- Asynchronous invocation

  You can configure the maximum number of retries and the maximum message validity period (up to 24 hours) by referring to **Asynchronous Notification Policy**.

  FunctionGraph will retry a function based on these two parameters.

**Idempotency**

In programming, idempotency means that an application or component can identify duplicate events and prevent duplication, inconsistency, and data loss. If you want to keep a function idempotent, you need to design the function logic to correctly handle repeated events.

Idempotent function logic helps reduce the following problems:

- Unnecessary API calls
- Code processing time
- Data inconsistency
- Restrictions
- Latency

Ensure that your function code can process the same event multiple times without causing duplicate transactions or other unnecessary side effects in case of abnormal calls, retry of client, or retry within dependent functions.

## Helpful Links

- Invoke functions using APIs. For details, see **Executing a Function Synchronously** and **Executing a Function Asynchronously**.
- For details about FAQs about function invocation, see **Function Invocation FAQs**.

# 7 Managing Functions

## 7.1 Initialization

This section describes how to configure function initialization on the FunctionGraph console.

### Overview

Function initialization refers to the process of performing setup tasks when a function is invoked, including setting its initial state, allocating resources, and assigning initial values to variables within the function.

The initialization is executed after an instance is started. The instance starts to process requests only after the initialization is executed.

The initialization is executed only once during the lifecycle of a function instance. Initialization will be billed in the same way as function request processing.

The service logic shared by multiple requests can be implemented in the initializer to reduce the latency. For example, the logic of loading a deep learning model with large specifications or building a connection pool for databases. For details about how to use initializers to develop functions, see **Initializer** .

### Notes and Constraints

- **Initialization** is enabled by default for HTTP functions and cannot be disabled.
- Ensure that the function initializer and handler are in the same file.
- Set the initializer in the same way as the handler. For more information about the handler, see **Table 7-2**.
- Initialization is not supported for event functions based on custom images.
- Do not read or write data on disks in the initializer. Otherwise, the function cannot read data on disks.

## Configuring Initialization

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Lifecycle**.
4. On the displayed page, enable **Initialization**, set initialization parameters by referring to **Table 7-1**, and click **Save**.

**Figure 7-1** Enabling initialization

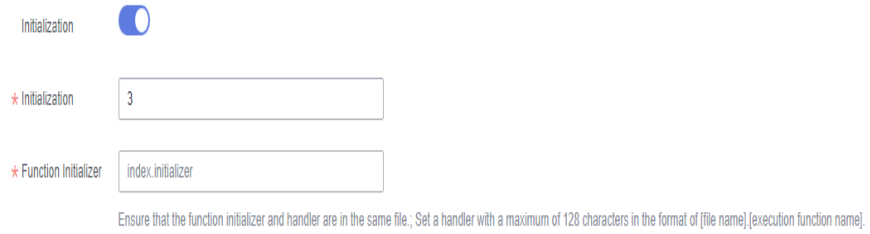



**Table 7-1** Parameter configuration

| Parameter | Description |
| --- | --- |
| Initialization Timeout (s) | The value ranges from 1s to 300s. |
| Initializer | **This parameter is available only for event functions and is not displayed for HTTP functions.**<br>Ensure that the function initializer and handler are in the same file. The initializer must be named in the same way as the handler. For example, for a Node.js or Python function, set an initializer name in the format of [file name].[initialization function name]. The name can contain a maximum of 128 characters.<br>**NOTE**<br>For details on how to configure the code, see **Configuring Function Code**. |

## Initializer Code Example

- Node.js (**Initializer introduction**)

```
exports.initializer = function(context, callback) {
    callback(null, '');
    };
```

- Python (**Initializer introduction**)

```
def my_initializer(context):
    print("hello world!")
```

- Java (**Initializer introduction**)

```
public void my_initializer(Context context)
{
RuntimeLogger log = context.getLogger();
log.log(String.format("ak:%s", context.getSecurityAccessKey()));
}
```

- PHP (**Initializer introduction**)

```
<?php
Function my_initializer($context) {
    echo 'hello world' . PHP_EOL;
    }
?>
```

## Helpful Links

Manage the function lifecycle using APIs. For details, see **Function Lifecycle Management APIs**.

# 7.2 Basic Settings

The basic settings of a function, such as the handler, execution timeout, and memory, are set by default based on the runtime selected during function creation. You can modify the settings as required.

## Configuring Basic Settings

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Basic Settings**.
4. On the displayed page, modify the basic settings by referring to **Table 7-2**.

**Table 7-2** Basic settings

| Parameter | Description |
| --- | --- |
| Function Name | Function name configured during function creation.<br>It is set when a function is created and cannot be modified. |
| Function Version | Function version.<br>The version of the newly created function is v2, which cannot be changed. |
| Application | An application acts like a folder. In the future, functions will be managed by tags for better experience.<br>The application of the newly created function is **default**, which cannot be changed. |
| Runtime | Runtime language of the function.<br>It is set when a function is created and cannot be modified. |

| Parameter | Description |
| --- | --- |
| Enterprise Project | Enterprise project to which this function belongs. You can use different enterprise projects to manage functions. Select a created enterprise project and add the function to it. By default, **default** is selected.<br>This parameter is available only after the enterprise project feature is enabled. For details, see **Enabling the Enterprise Project Function** |
| Handler | Function handler is the entry point specified in the function code for executing logic. Generally, it is a specific function or method. When a function is triggered, the code is executed from the handler.<br>The handler format varies depending on the runtime and cannot exceed 128 characters.<br>• For a Node.js, Python, or PHP function, the handler must be named in the format of *[file name].[function name]*, which must contain a period (.).<br>Default: index.handler<br>• For a Java function, the handler must be named in the format of *[package name].[class name].[execution function name]*.<br>Default: **com.huawei.demo.TriggerTests.apigTest**<br>• For a Go function, the handler name must be the same as the executable file name in the code file.<br>Default: If the executable file is **handler**, set this parameter to **handler**.<br>• For a C# function, the handler must be named in the format of *[assembly]*::*[namespace].[class name]*::*[execution function name]*.<br>Default: CsharpDemo::CsharpDemo.Program::MyFunc |

| Parameter | Description |
| --- | --- |
| Execution Timeout (s) | Timeout interval for executing the function. If the execution times out, the function will be forcibly stopped. If the execution takes longer than 900s, use asynchronous invocation.<br>**NOTE**<br>FunctionGraph has three restrictions on the function execution duration:<br>• Maximum execution duration of a function: The built-in execution duration of a function is limited to prevent the function from executing for a long time. In synchronous invocation scenarios, the maximum function execution duration is 15 minutes. In asynchronous invocation scenarios, the maximum function execution duration is 259,200 seconds (3 days). For details, see **Invoking the Function**.<br>• Execution timeout (s): You can configure the function execution timeout based on service requirements. The timeout must be less than the maximum function execution duration.<br>• Frontend timeout: This refers to the built-in timeout reminder on frontend pages. The frontend page waits for a backend response for a maximum of 90 seconds. If the configured timeout exceeds 90 seconds and the actual function execution exceeds 90 seconds, a timeout message is displayed on the frontend page, but the backend is still running properly. After the execution is complete, you can still view the result in the log.<br>The execution timeout ranges from 3 to 259,200 seconds. The default value is 3. |
| Memory (MB) | Memory size that can be used for function execution. The default value is **128**.<br>The value can be 128, 256, 512, 768, 1,024, 1,280, 1,536, 1,792, 2,048, 2,560, 3,072, 3,584, 4,096, 8,192, or 10,240. |
| Description | Function description. You can customize the description of the function service.<br>Enter a maximum of 512 characters. |

5. Click **Save** to save the configuration.

## Helpful Links

Manage the function lifecycle using APIs. For details, see **Function Lifecycle Management APIs**.

# 7.3 Disk Mounting

This section describes how to mount a disk to a function on the FunctionGraph console to expand the storage space of the function.

## Scenarios

FunctionGraph supports persistent storage by mounting file systems. Advantages:

- Multiple functions can mount and share the same file system.
- Existing storage capabilities of ECSs can be used to dynamically expand compute resources.
- Compared with the temporary storage space **/tmp**, the function storage space can be greatly expanded.

NOTE

The **/tmp** directory can store temporary files. The default size is 512 MB and the maximum size is 10,240 MB.

FunctionGraph supports the following file systems:

**Table 7-3** Disk mounting

| File System Type | Description |
| --- | --- |
| SFS Turbo | SFS Turbo provides a fully hosted shared file storage that can provide petabytes of capacity, sub-millisecond latency, up to tens of millions of IOPS, and hundreds of GB/s of bandwidth. It can provide high availability and durability for workloads dealing with massive small files and applications that require low latency and high IOPS. **For details about SFS Turbo file systems, see SFS Turbo Service Overview.**<br>The advantages of using SFS Turbo for function mounting are as follows:<br>• Ephemeral files can be stored in the SFS Turbo file system. Their size will not be limited by instance local disk space.<br>• Multiple functions can share an SFS Turbo file system. |
| General Purpose File System | Expandable to petabytes of capacity and terabytes of bandwidth, the general purpose file system provides fully hosted shared file storage. It features high availability and durability, and provides support for data-intensive and bandwidth-intensive applications. It supports large-scale data throughput and high-bandwidth workloads such as video transcoding and batch log analytics. **For details about SFS general purpose file system, see SFS General Purpose File System.** |
| ECS | A directory on an ECS is specified as a shared file system by using the network file system (NFS) service. Functions can mount the directory for read and write operations. You can configure user permissions to manage shared data resources. It suitable for use cases that rely on existing ECS resources to rapidly enable lightweight data processing and file-sharing, particularly when such workloads occur only occasionally. |

| File System Type | Description |
| --- | --- |
| SFS Capacity-Oriented File System (Only for Existing Users) | Expandable to petabytes, SFS Capacity-Oriented file systems provide fully hosted shared file storage. It features high availability and durability, and provides support for data-intensive and bandwidth-intensive applications. |

## Notes and Constraints

- **Only existing users can mount SFS Capacity-Oriented file systems. The regions where each file system can be mounted are subject to the console.**
- Do not read or write data on disks in the initializer. Otherwise, the function cannot read data on disks.
- The following file system ports must be enabled: **111**, **445**, **2049**, **2051**, **2052**, and **20048**.

  Another three ports for Ubuntu. To obtain the port numbers, run the following command:

  ```
  rpcinfo -p|grep mountd|grep tcp
  ```

  For details, see **What Resources Does SFS Occupy?**

## Prerequisites

**Table 7-4** Prerequisites for mounting a disk

| File System Source | Prerequisites |
| --- | --- |
| **Mounting an SFS Turbo file system** | • You have created an SFS Turbo file system to be mounted. For details, see **Creating a File System**.<br>You have recorded the VPC and subnet information used by the SFS Turbo file system.<br>• You have created the agency required for mounting an SFS Turbo file system. For details, see **Configuring Agency Permissions**. |
| **Mounting a General Purpose File System** | • You have created an SFS General-purpose system to be mounted.<br>You have recorded the VPC and subnet information used by the SFS General-purpose file system.<br>• You have created the agency required for mounting an SFS General-purpose file system. For details, see **Configuring Agency Permissions**. |

| File System Source | Prerequisites |
| --- | --- |
| **Mounting an ECS shared directory** | • You have created an ECS. For details, see **Purchasing an ECS**. You have recorded the VPC and subnet information used by the ECS.<br>• You have created an NFS shared directory on the ECS. For details, see **Creating an NFS Shared Directory on the ECS**.<br>• You have created the required agency for mounting an ECS shared directory. For details, see **Configuring Agency Permissions**. |
| **Mounting an SFS Capacity-Oriented File System (Only for Existing Users)** | • You have created an SFS Capacity-Oriented file system.<br>• You have created the required agency for mounting an SFS Capacity-Oriented file system. For details, see **Configuring Agency Permissions**. |

## Mounting an SFS Turbo file system

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Permissions** to configure an agency for the function to access the SFS Turbo file system.
   a. Select the agency created in **Prerequisites**.
   b. Click **Save**.
4. Choose **Configuration** > **Network** to configure the VPC for the function to access the SFS Turbo file system.
   a. Enable **VPC Access**.
   b. Set **VPC** and **Subnet** to the VPC and subnet recorded in **Prerequisites**.
   c. Click **Save**.
5. Choose **Configuration** > **File Systems**.
6. Click **Mount File System**. In the displayed dialog box, configure the SFS Turbo file system.

**Table 7-5** Parameters for mounting an SFS Turbo file system

| Parameter | Description | Example Value |
| --- | --- | --- |
| User ID | User ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the user ID.<br>For SFS Turbo file systems, retain the default value **–1**. | -1 |
| Group ID | Group ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the group ID.<br>For SFS Turbo file systems, retain the default value **–1**. | -1 |
| File System Type | Select the type of the file system to be mounted. In this example, select **SFS Turbo**. | SFS Turbo |
| File System | Select the name of the SFS Turbo file system to be mounted. | sfs-turbo-fg |
| Shared Directory | Specify the shared directory of the file system. The path must start with a slash (/), for example, **/a**.<br>If this parameter is not set, the function can access all directories of the file system. If a specific directory path is configured, the function can access only the directory path. | /a |
| Access Path | Specify the directory path used by the function to access files in the file system. The path must start with a slash (/) and contain a maximum of two levels. You are advised to set this parameter to a level-2 directory starting with **/mnt** or **/home**.<br>The directory that already exists in the system cannot be used. Otherwise, the error message **failed to mount exist system path** is displayed.<br>For example, the shared directory path is **/a**, with files **/a/b** and **/a/c/d** under it. If the function access path is **/mnt/test**, then the paths for the function to access file b and file d are **/mnt/test/b** and **/mnt/test/c/d** respectively. | /mnt/test |

7. Click **OK**.

## Mounting a General Purpose File System

Currently, only the **CN North-Ulanqab1** region supports mounting a general purpose file system.

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Permissions** to configure an agency for the function to access the SFS General-purpose file system.
   a. Select the agency created in **Prerequisites**.
   b. Click **Save**.
4. Choose **Configuration** > **Network** to configure the VPC for the function to access the SFS General-purpose file system.
   a. Enable **VPC Access**.
   b. Set **VPC** and **Subnet** to the VPC and subnet recorded in **Prerequisites**.
   c. Click **Save**.
5. Choose **Configuration** > **File Systems**.
6. Click **Mount File System**. In the displayed dialog box, configure the SFS General-purpose file system.

**Table 7-6** Parameters for mounting an SFS General-purpose file system

| Parameter | Description | Example Value |
| --- | --- | --- |
| User ID | User ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the user ID.<br>For SFS Turbo file systems, retain the default value **–1**. | -1 |
| Group ID | Group ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the group ID.<br>For SFS Turbo file systems, retain the default value **–1**. | -1 |
| File System Type | Select the type of the file system to be mounted. In this example, select **General purpose**. | General purpose |
| File System | Select the name of the SFS General-purpose file system to be mounted. | sfs-fg |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Access Path | Specify the directory path used by the function to access files in the file system. The path must start with a slash (/) and contain a maximum of two levels. You are advised to set this parameter to a level-2 directory starting with **/mnt** or **/home**.<br>The directory that already exists in the system cannot be used. Otherwise, the error message **failed to mount exist system path** is displayed.<br>For example, the shared directory path is **/a**, with files **/a/b** and **/a/c/d** under it. If the function access path is **/mnt/test**, then the paths for the function to access file b and file d are **/mnt/test/b** and **/mnt/test/c/d** respectively. | /mnt/test |

7. Click **OK**.

## Mounting an ECS Shared Directory

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Permissions** to configure the agency for the function to access the ECS shared directory.
   a. Select the agency created in **Prerequisites**.
   b. Click **Save**.
4. Choose **Configuration** > **Network** to configure the VPC for the function to access the ECS shared directory.
   a. Enable **VPC Access**.
   b. Set **VPC** and **Subnet** to the VPC and subnet recorded in **Prerequisites**.
   c. Click **Save**.
5. Choose **Configuration** > **File Systems**.
6. Click **Mount File System**. In the displayed dialog box, configure the ECS shared directory.

**Table 7-7** Parameters for mounting an ECS shared directory

| Parameter | Description | Example Value |
| --- | --- | --- |
| User ID | User ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the user ID.<br>● For ECSs on Windows, retain the default value **–1**.<br>● For ECSs on Linux, use the system uid. For example, if the Linux username is **test-user**, you can run the **id test-user** command to query the uid and gid. | -1 |
| Group ID | Group ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the group ID.<br>● For ECSs on Windows, retain the default value **–1**.<br>● For ECSs on Linux, use the system gid. For example, if the Linux username is **test-user**, you can run the **id test-user** command to query the gid. | -1 |
| File System Type | Select the file system type to be mounted. In this example, select **ECS**, which indicates the ECS shared directory. | ECS |
| ECS | Select the name of the ECS to be mounted. | ecs-fg |
| Shared Directory | Enter an existing directory path starting with a slash (/), for example, **/a**.<br>If this parameter is not set, the function can access all directories in the file system. | /a |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Access Path | Specify the directory path used by the function to access files in the file system. The path must start with a slash (/) and contain a maximum of two levels. You are advised to set this parameter to a level-2 directory starting with **/mnt** or **/home**.<br>The directory that already exists in the system cannot be used. Otherwise, the error message **failed to mount exist system path** is displayed.<br>For example, the shared directory path is **/a**, with files **/a/b** and **/a/c/d** under it. If the function access path is **/mnt/test**, then the paths for the function to access file b and file d are **/mnt/test/b** and **/mnt/test/c/d** respectively. | /mnt/test |

7. Click **OK**.

## Mounting an SFS Capacity-Oriented File System (Only for Existing Users)

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Permissions** to configure an agency for the function to access the SFS Capacity-Oriented file system.
   a. Select the agency created in **Prerequisites**.
   b. Click **Save**.
4. Choose **Configuration** > **File Systems**.
5. Click **Mount File System**. In the displayed dialog box, configure the SFS Capacity-Oriented file system.

**Table 7-8** Parameters for mounting an SFS Capacity-Oriented file system

| Parameter | Description | Example Value |
| --- | --- | --- |
| User ID | User ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the user ID.<br>For SFS Capacity-Oriented file systems, retain the default value **–1**. | -1 |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Group ID | Group ID to be used for the function to access the specified file system. The value can be **–1** (default) or an integer ranging from 1 to 65534, excluding 1000 and 1002. **–1** indicates that the system automatically allocates the group ID.<br>For SFS Capacity-Oriented file systems, retain the default value **–1**. | -1 |
| File System Type | Select the type of the file system to be mounted. In this example, select **SFS**. | SFS |
| File System | Select the name of the SFS Capacity-Oriented file system to be mounted. | SFS-fg |
| Shared Directory | Specify the shared directory of the file system. The path must start with a slash (/), for example, **/a**.<br>If this parameter is not set, the function can access all directories in the file system. | /a |
| Access Path | Specify the directory path used by the function to access files in the file system. The path must start with a slash (/) and contain a maximum of two levels. You are advised to set this parameter to a level-2 directory starting with **/mnt** or **/home**.<br>The directory that already exists in the system cannot be used. Otherwise, the error message **failed to mount exist system path** is displayed.<br>For example, the shared directory path is **/a**, with files **/a/b** and **/a/c/d** under it. If the function access path is **/mnt/test**, then the paths for the function to access file b and file d are **/mnt/test/b** and **/mnt/test/c/d** respectively. | /mnt/test |

6. Click **OK**.

## Creating an NFS Shared Directory on the ECS

Linux (CentOS, SUSE, EulerOS, Fedora, and OpenSUSE)

- Procedure
  a. Configure a YUM repository.
     i. Create a file named **euleros.repo** in the **/etc/yum.repos.d** directory. Ensure that the file name end with **.repo**.
     ii. Run the following command to edit the **euleros.repo** file.
     ```
     vi /etc/yum.repos.d/euleros.repo
     ```
     iii. Add the information to the **euleros.repo** file.
     The EulerOS 2.0 SP3 YUM configuration is as follows:

```
[base]
name=EulerOS-2.0SP3 base
baseurl=http://repo.huaweicloud.com/euler/2.3/os/x86_64/
enabled=1
gpgcheck=1
gpgkey=http://repo.huaweicloud.com/euler/2.3/os/RPM-GPG-KEY-EulerOS
```

The EulerOS 2.0 SP5 YUM configuration is as follows:

```
[base]
name=EulerOS-2.0SP5 base
baseurl=http://repo.huaweicloud.com/euler/2.5/os/x86_64/
enabled=1
gpgcheck=1
gpgkey=http://repo.huaweicloud.com/euler/2.5/os/RPM-GPG-KEY-EulerOS
```

**Table 7-9** Parameter description:

| Parameter | Description |
| --- | --- |
| name | Repository name. |
| baseurl | Repository URL.<br>• HTTP-based network address: **http://path/to/repo**<br>• Local repository address: **file:///path/to/local/repo** |
| gpgcheck | Indicates whether to enable the GNU privacy guard (GPG) to check the validity and security of RPM package resources.<br>• **1**: The GPG check is enabled.<br>• **0**: The GPG check is disabled.<br>• If this option is not specified, the GPG check is enabled by default. |

iv. Press **Esc** to exit the input mode, enter **:wq**, and press **Enter** to save the **euleros.repo** file and exit.

v. Run the following command to clear the cache.

```
yum clean all
```

b. Run the following command to install **nfs-utils**.

```
yum install nfs-utils
```

c. Set the shared directory.

i. Run the following command to edit the **/etc/exports** file.

```
vi /etc/exports
```

ii. Add the following information to the **exports** file.

```
/sharedata  192.168.0.0/24(rw,sync,no_root_squash)
```

**/sharedata** indicates the path of the directory to be shared. **192.168.0.0/24** indicates that the directory is shared to other servers in this CIDR block.

iii. Press **Esc** to exit the input mode, enter **:wq**, and press **Enter** to save the **exports** file and exit.

d. Run the following commands to start the NFS service.

```
systemctl start rpcbind
service nfs start
```

e. Run the following command to check whether the setting is successful.
```
exportfs -v
```
If the shared directory path is displayed in the command output, the setting is successful.

f. Modify the shared directory.

To modify or add a shared directory, modify the configuration file by referring to **c**. After the modification, run the following command to restart the NFS service.
```
service nfs restart
```

g. (Optional) Enable automatic startup of the rpcbind service.

Run the following command:
```
systemctl enable rpcbind
```

Linux (Ubuntu)

- Procedure

a. Run the following commands to install **nfs-kernel-server**.
```
sudo apt-get update
sudo apt install nfs-kernel-server
```

b. Set the shared directory.

i. Run the following command to edit the **/etc/exports** file.
```
vim /etc/exports
```

ii. Add the following information to the **exports** file.
```
/sharedata  192.168.0.0/24(rw,sync,no_root_squash)
```
**/sharedata** indicates the path of the directory to be shared. **192.168.0.0/24** indicates that the directory is shared to other servers in this CIDR block.

iii. Press **Esc** to exit the input mode, enter **:wq**, and press **Enter** to save the **exports** file and exit.

c. Run the following commands to start the NFS service.
```
service nfs-kernel-server restart
```

d. Run the following command to check whether the setting is successful.
```
exportfs -v
```
If the shared directory path is displayed in the command output, the setting is successful.

e. Modify the shared directory.

To modify or add a shared directory, modify the configuration file by referring to **b**. After the modification, run the following command to restart the NFS service.
```
service nfs restart
```

f. (Optional) Enable automatic startup of the rpcbind service.

Run the following command:
```
systemctl enable rpcbind
```

Windows

- Procedure

For details about how to install NFS and share files, see the **official document** on Windows.

Windows OS (haneWIN NFS Server software)

- Procedure
  a. Obtain the haneWIN NFS Server software from the official website.

     **Note: haneWIN NFS Server is a paid software.**
  b. Run the **.exe** file in the software package as the Windows system administrator to install haneWIN NFS Server.
  c. After the installation is complete, open the **NFS Server** software and choose **Edit** > **Preferences**.
  d. Click the **Exports** tab, click **Edit exports file**, configure the shared directory, retain the default settings on the **NFS**, **Server**, and **PortMapper** tab pages, and click **Save**.

     The shared directory format can be referenced as **D:\share -public -name:nfs**, which means to set the permission on the **share** folder to **public** and define an alias **nfs**.
  e. Click **OK**.
  f. Disable all firewalls, including the **Domain network**, **Private network**, and **Public network**. Enable them after the entire configuration is complete.

  NOTE

  Run the following command in Linux to mount the directory and check whether the file sharing is successful:

  ```
  mount -t nfs -o nolock 192.168.xxx.xxx:/nfs /mnt
  ```

  - **192.168.***xxx.xxx* indicates the IP address of the Windows system.
  - **nfs** is the alias created when the shared directory is configured.
  - **/mnt** is the local directory where the remote directory is mounted.

## Helpful Links

For details about the storage types supported by FunctionGraph, see **Function Storage Selection**.

# 7.4 Environment Variables

This section describes how to configure function environment variables on the FunctionGraph console.

## Scenarios

Environment variables allow you to pass dynamic parameters to a function without modifying code.

Scenarios:

- Environment distinguishing: Configure different environment variables for the same function logic. For example, use environment variables to configure testing and development databases.

- Configuration encryption: Configure encrypted environment variables to dynamically obtain authentication information (account, password) required to access other services.
- Dynamic configuration: Configure environment variables for parameters that need to be dynamically adjusted, including query period and timeout, in function logic.

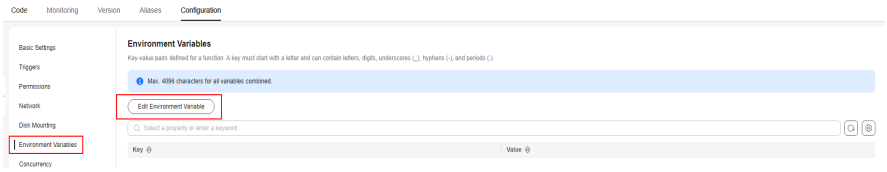
## Notes and Constraints

When you define environment variables, FunctionGraph displays all your input information in plain text. For security purposes, do not include sensitive information.

## Configuring Environment Variables

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Environment Variables** and click **Edit Environment Variable**.

   **Figure 7-2** Adding environment variables

4. On the displayed dialog box, click **Add** and configure the environment variable information.

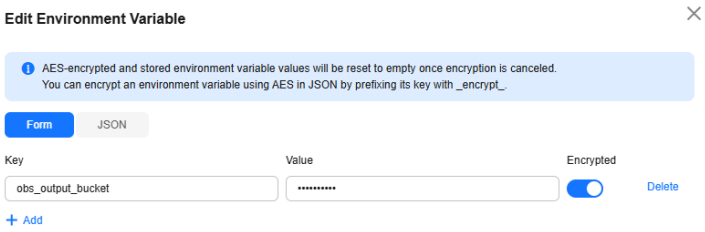
   Editing environment variables using a form

   – For details about the parameters for editing environment variables using a form, see **Table 7-10**.

   **Table 7-10** Parameters for configuring environment variables

| Parameter | Description |
| --- | --- |
| Key | Key of the environment variable.<br>The key must start with a letter and can contain a maximum of 4,096 characters, including letters, digits, underscores (_), hyphens (-), and periods (.).<br>Some environment variables are preset in the system. You cannot configure environment variables with the same name as the preset environment variables. For details about the preset environment variables, see **Preset Environment Variables**. |
| Value | Value of the environment variable.<br>The total length of the key and value cannot exceed 4,096 characters. |

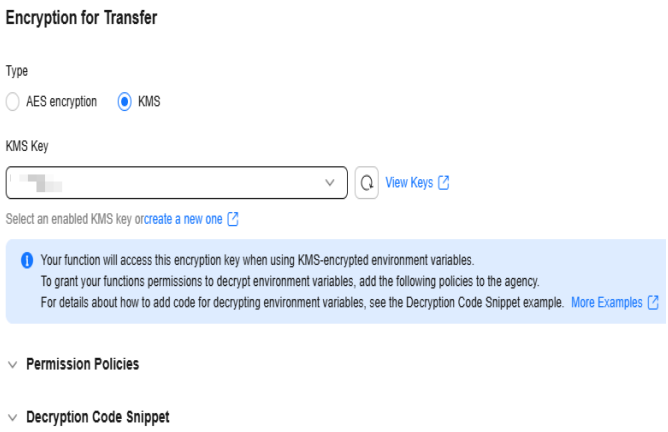
  

| Parameter | Description |
| --- | --- |
| Encrypted | If enabled, the environment variable value is encrypted and displayed as an asterisk (*). During parameter transmission, the key value is also encrypted.<br>Encrypted and saved environment variable values will be reset to empty if the encryption is disabled.<br><br>**Figure 7-3** Editing environment variables using a form<br>  <br> |

| Parameter | Description |
| --- | --- |
| Encryption for Transfer | **This parameter is available only in the LA-Sao Paulo1 region.**<br>**CAUTION**<br>DEW charges you on a pay-per-use basis. For details, see **DEW Billing**.<br>Click **Enable** to encrypt the environment variable. The following encryption types are supported:<br>• **AES**: Encrypt environment variables using AES. You do not need to manually create a key.<br>• **KMS**: You can select an existing KMS key to encrypt the function environment variable.<br>**Prerequisites**<br>Before using KMS to encrypt environment variables, make sure you have granted the function the permission to decrypt environment variables. For details, see **Creating a Custom Policy in JSON View**. Go to the IAM console and add the permission policy to the function agency. For details, see **Configuring an Agency**.<br>**Selecting a KMS key**<br>Select the enabled customer master key (CMK). If no key is available, create one by referring to **Creating a Custom Key**.<br>**CAUTION**<br>Do not delete the CMK used for function encryption in DEW. Otherwise, the function execution will fail because encrypted data cannot be decrypted.<br>**Decrypting environment variables**<br>After KMS is used to encrypt environment variables, add **Decryption Code Snippet** to the function code to decrypt the environment variables. Replace {endpoint} in the code with the **endpoint** of the corresponding region. |

| Parameter | Description |
| --- | --- |
| | **Figure 7-4** Encryption during transmission<br>  <br>If the parameter is enabled, the environment variable value is encrypted and displayed as an asterisk (*). During parameter transmission, the key value is also encrypted.<br>Encrypted and saved environment variable values will be reset to empty if the encryption is disabled. |

Editing environment variables in JSON

- The following is an example of editing environment variables in JSON format.

```
{
  "obs_output_bucket": "obs_image"
}
```

To use AES to encrypt environment variables, add the prefix **_encrypt_** to the key.

```
{
  "_encrypt_obs_output_bucket": "obs_image"
}
```

5. Configure **Static Encryption** for the environment variables. (This step is optional and available only in the **LA-Sao Paulo1** region.)

   To use static encryption with KMS, add the permission policy to the agency of the function. The following KMS static encryption modes are supported:

   - **functiongraph/default (default)**: The function automatically creates a default key in DEW.
   - **Customer master key (CMK)**: Select the CMK to encrypt the code. For details about how to create a CMK, see **Creating a Custom Key**.

> **CAUTION**
>
> If you select **Customer master key (CMK)**, do not delete the CMK used for function encryption in DEW. Otherwise, the function execution will fail because encrypted data cannot be decrypted.

6. After the configuration is complete, click OK to save the environment variables.

   After the environment variables are configured, you can use the environment variables in functions. For details, see **Application Example of Environment Variables**.

## Preset Environment Variables

**Table 7-11** lists the preset environment variables.

**Table 7-11** Preset environment variables

| Environment Variable | Description | Obtaining Method |
| --- | --- | --- |
| RUNTIME_PROJECT_ID | Project ID | Obtain the value from a Context interface or a system environment variable. |
| RUNTIME_FUNC_NAME | Function name | Obtain the value from a Context interface or a system environment variable. |
| RUNTIME_FUNC_VERSION | Function version | Obtain the value from a Context interface or a system environment variable. |
| RUNTIME_HANDLER | Handler | Obtain the value from a system environment variable. |
| RUNTIME_TIMEOUT | Execution timeout allowed for a function | Obtain the value from a system environment variable. |
| RUNTIME_USERDATA | Value passed through an environment variable | Obtain the value from a Context interface or a system environment variable. |
| RUNTIME_CPU | CPU usage of a function. The value is in proportion to **MemorySize**. | Obtain the value from a Context interface or a system environment variable. |

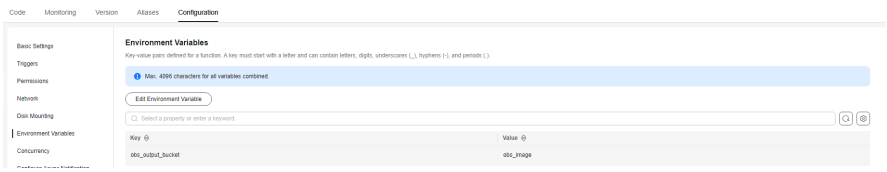
| Environment Variable | Description | Obtaining Method |
| --- | --- | --- |
| RUNTIME_MEMORY | Memory size configured for a function, in MB | Obtain the value from a Context interface or a system environment variable. |
| RUNTIME_MAX_RESP_BODY_SIZE | Maximum size of the return value (default: 6,291,456 bytes) | Obtain the value from a system environment variable. |
| RUNTIME_INITIALIZER_HANDLER | Initializer | Obtain the value from a system environment variable. |
| RUNTIME_INITIALIZER_TIMEOUT | Initialization timeout of a function | Obtain the value from a system environment variable. |
| RUNTIME_ROOT | Path of the runtime package (default: **/home/snuser/runtime**) | Obtain the value from a system environment variable. |
| RUNTIME_CODE_ROOT | Path of the code in the container (default: **/opt/function/code**) | Obtain the value from a system environment variable. |
| RUNTIME_LOG_DIR | Directory for storing system logs in the container (default: **/home/snuser/log**) | Obtain the value from a system environment variable. |

## Application Example of Environment Variables

You can use environment variables to configure which directory to install files in, where to store outputs, and how to store connection and logging settings. These settings are decoupled from the application logic, so you do not need to update your function code when you change the settings.

1. Using environment variable **obs_output_bucket**, you can flexibly set the OBS bucket used for storing output images.

   **Figure 7-5** Environment variables

2. Use environment variables in function code.

   In the following code snippet, **obs_output_bucket** is the bucket used for storing processed images.

NOTE

- Non-HTTP functions use **context.getUserData ('xxx')** to obtain environment variables.
- HTTP functions use system methods to obtain environment variables. For example, Python functions use **os.Environ['xx']**, and Node.js functions use **process.env.xx**. Node.js functions use **process.env.RUNTIME_USERDATA** to obtain encrypted environment variables.

– **Python**

```
def handler(event, context):
    srcBucket, srcObjName = getObsObjInfo4OBSTrigger(event)
    obs_address = context.getUserData('obs_address')
    outputBucket = context.getUserData('obs_output_bucket')
    if obs_address is None:
        obs_address = '{obs_address_ip}'
    if outputBucket is None:
        outputBucket = 'casebucket-out'

    ak = context.getSecurityAccessKey()
    sk = context.getSecuritySecretKey()
    st = context.getSecurityToken()

    # download file uploaded by user from obs
    # TODO: Replace with actual implementation
    GetObject(obs_address, srcBucket, srcObjName, ak, sk, st)

    outFile = watermark_image(srcObjName)

    # Upload converted files to a new OBS bucket.
    # TODO: Replace with actual implementation
    PostObject(obs_address, outputBucket, outFile, ak, sk, st)

    return 'OK'
```

– **Node.js**

```
exports.handler = async (event, context) => {
    let bucket = context.getUserData('obs_output_bucket');
    console.log(bucket);
    const output =
    {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }
    return output;
}
```

## Helpful Links

- Manage the function lifecycle using APIs. For details, see **Function Lifecycle Management APIs**.
- For FAQs about function environment variables, see **Function Configuration FAQs**.

# 7.5 Asynchronous Notification Policy

In asynchronous mode, FunctionGraph queues requests, does not wait for the execution result, and directly returns a response to the client. To set the

asynchronous request retry upon failure or obtain the asynchronous request result, you can configure the asynchronous notification policy.

- Retry: By default, FunctionGraph does not retry if a function fails due to a code error. If your function needs retry, for example, if third-party services often fail to be invoked, configure retry to improve the success rate.
- Result notification: FunctionGraph automatically sends the asynchronous execution result of a function to downstream services for further processing. For example, storing execution failure information in OBS for cause analysis, or pushing execution success information to DIS or triggering the function again.

## Notes and Constraints

**Table 7-12** Restrictions of asynchronous notification

| Restricted Scenario | Description |
| --- | --- |
| Using functions that are invoked asynchronously | The return value of a function in asynchronous mode cannot exceed 256 KB; otherwise, an empty value will be returned. |
| Configuring async notification to notify the target service | To avoid cyclic invocation, do not set two functions as asynchronous execution targets of each other. |

## Prerequisites

- You have configured an agency with operation permissions on the target service for the function to notify the target service in asynchronous mode. For details about how to configure an agency, see **Configuring Agency Permissions**.

## Configuring an Asynchronous Notification Policy

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Async Notification**.
4. On the displayed page, click **Edit**.

**Figure 7-6** Setting parameters

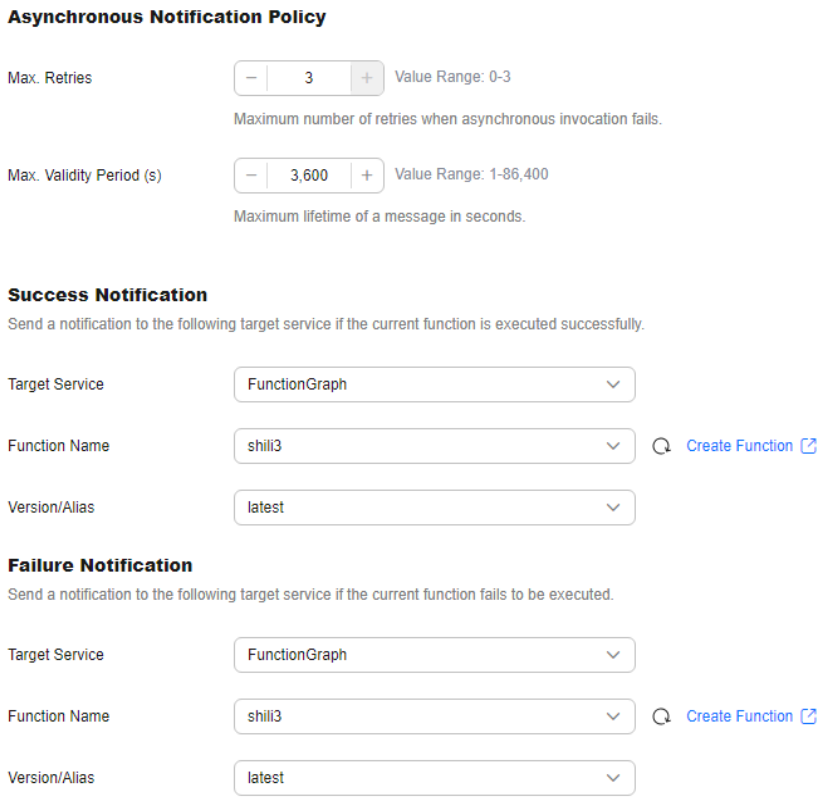



**Table 7-13** Asynchronous notification parameters

| Parameter | Description |
| --- | --- |
| Asynchronous Execution Notification Policy | • **Max. Retries**: maximum number of retries when asynchronous invocation fails. Value range: 0–3. Default value: **3**.<br>• **Max. Validity Period (s)**: maximum lifetime of a message in seconds. Value range: 1–86,400.<br>• **Asynchronous Invocation Status Persistence**: This parameter is displayed after you enable LTS. For details, see **Enabling Asynchronous Invocation Status Persistence**. |

| Parameter | Description |
| --- | --- |
| Success Notification | **Target Service**: to which a notification will be sent if a function is executed successfully.<br>• FunctionGraph<br>– **Function Name**: name of the function that will receive the notification.<br>– **Version/Alias**: Select the version or alias of the function.<br>• OBS<br>– **Target Bucket Directory**: Select the OBS bucket directory for storing notification messages.<br>– **Object Prefix Directory**: Enter an object prefix directory to filter the object files for storing notification messages.<br>– **Object Validity Period (Days)**: Configure the time duration after which an object will expire. The OBS server automatically clears expired objects. The value ranges from 0 to 365. The value 0 indicates that the object does not expire.<br>• DIS<br>**Target Stream**: Select the stream that receives notification messages.<br>• SMN<br>**Topic Name**: Select the topic that receives notification messages. |

| Parameter | Description |
| --- | --- |
| Failure Notification | **Target Service**: to which a notification will be sent if a function fails to be executed.<br>• FunctionGraph<br>– **Function Name**: name of the function that will receive the notification.<br>– **Version/Alias**: Select the version or alias of the function.<br>• OBS<br>– **Target Bucket Directory**: Select the OBS bucket directory for storing notification messages.<br>– **Object Prefix Directory**: Enter an object prefix directory to filter the object files for storing notification messages.<br>– **Object Validity Period (Days)**: Configure the time duration after which an object will expire. The OBS server automatically clears expired objects. The value ranges from 0 to 365. The value 0 indicates that the object does not expire.<br>• DIS<br>**Target Stream**: Select the stream that receives notification messages.<br>• SMN<br>**Topic Name**: Select the topic that receives notification messages. |

5. Click **OK**.

## Notification Message Structure

The following is an example of an asynchronous invocation notification sent by FunctionGraph to the target.

```
{
      "timestamp": "2020-08-20T12:00:00.000Z+08:00",
   "request_context": {
          "request_id": "1167bf8c-87b0-43ab-8f5f-26b16c64f252",
          "function_urn": "urn:fss:xx-xxxx-x:xxxxxxx:function:xxxx:xxxx:latest",
          "condition": "",
          "approximate_invoke_count": 0
    },
      "request_payload": "",
      "response_context": {
          "status_code": 200,
          "function_error": ""
     },
    "response_payload": "hello world!"
}
```

**Table 7-14** Message parameters

| Parameter | Description |
| --- | --- |
| timestamp | Execution start time |
| request_context | Request context. |
| request_context.request_id | ID of an asynchronous invocation request. |
| request_context. function_urn | URN of the function that is to be executed asynchronously. |
| request_context.condition | Invocation error type. |
| request_context. approximate_invoke_count | Number of asynchronous invocation times. If the value is greater than 1, function execution has been retried. |
| request_payload | Original request payload. |
| response_context | Response context. |
| response_context.statusCode | System return code of the invoked function. If the code is not 200, a system error occurred. |
| response_context.function_error | Invocation error information. |
| response_payload | Payload returned after function execution. |

## Enabling Asynchronous Invocation Status Persistence

With asynchronous invocation status persistence enabled, you can query the execution status of each asynchronous invocation in logs reported to LTS.

NOTE

By default, you do not have the permission to enable **Asynchronous Invocation Status Persistence**. You need to **submit a service ticket** to add a whitelist.

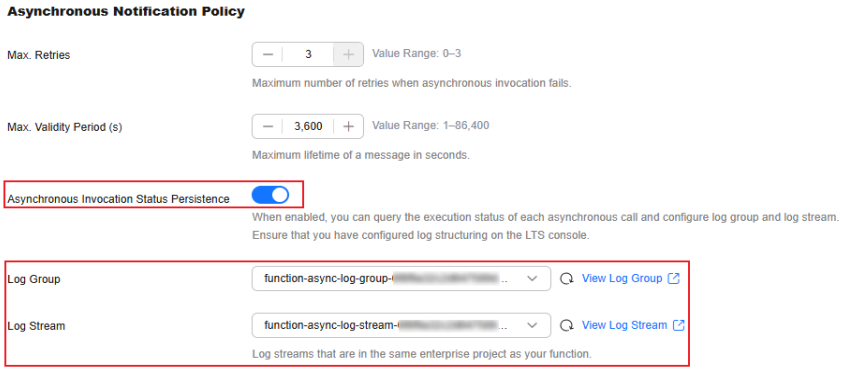
1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function to go to the function details page. Choose **Configuration** > **Async Notification**.
3. On the displayed page, click **Enable LTS** in the upper part of the page to enable LTS. (Skip this step if LTS has been enabled.)

   During this process, the system automatically calls the LTS API to create a log group (named starting with **function-async-log-group**) and a log stream (named starting with **function-async-log-stream**) on the LTS console.
4. After the LTS is enabled, click **Edit**. The **Asynchronous Invocation Status Persistence** parameter is displayed.

5. Enable **Asynchronous Invocation Status Persistence** and select the log group and log stream automatically created in **3**.

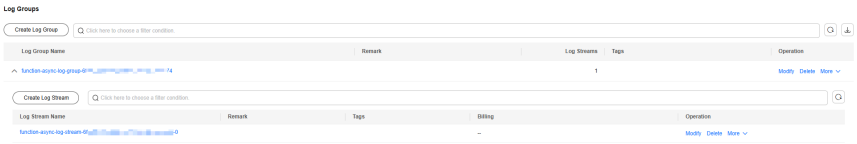
**Figure 7-7** Enabling Asynchronous Invocation Status Persistence




6. Configure log structuring on the LTS console.

   a. Log in to the LTS console. In the navigation pane, choose **Log Management**. You can view the automatically created log group and log stream in the log group list.

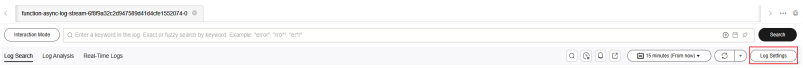
   **Figure 7-8** Log group list

   b. In the displayed log stream list, click the name of the log stream you want to view.

   c. On the log stream details page, click **Log Settings** in the upper right corner, as shown in **Figure 7-9**.

   **Figure 7-9** Log settings

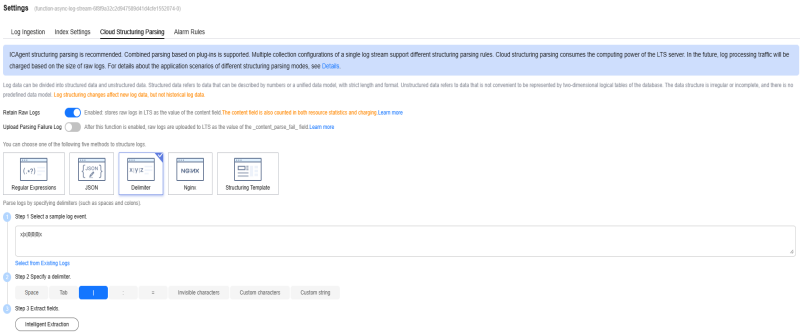
   d. In the displayed dialog box, click the **Cloud Structuring Parsing** tab and configure log structuring, as shown in **Figure 7-10**.

      - Select **Delimiter** method.
      - Enter **x|x|0|0|0|x** in the sample log event area.

        The placeholder indicates that the first two fields are of the string type, the middle three fields are of the long type, and the last field is of the string type.
      - Select the vertical bar (|) as the delimiter.

**Figure 7-10** Cloud structuring parsing

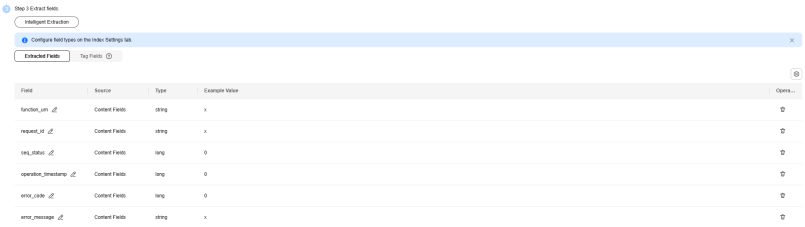


e. Click **Intelligent Extraction** in step 3 to generate fields, and click  to change the field names by referring to **Table 7-15**. **Figure 7-11** shows the modified fields.

**Table 7-15** Modifying fields

| Field | Modified Field |
| --- | --- |
| field1 | function_urn |
| field2 | request_id |
| field3 | seq_status |
| field4 | operation_timestamp |
| field5 | error_code |
| field6 | error_message |

**Figure 7-11** Modifying extraction fields



f. Click **Save**.

## Helpful Links

- Manage asynchronous notification for functions using APIs. For details, see **Asynchronous Execution Notification APIs**.
- If the message indicating insufficient permissions is displayed when you modify the asynchronous notification, add the **FunctionGraph Administrator** permission. For details, see **Creating a User and Granting Permissions**.

# 7.6 Concurrency

## Overview

By default, each function instance processes only one request at a time. If FunctionGraph receives concurrent requests, for example, three concurrent requests, FunctionGraph starts three function instances to process the requests. To address this issue, FunctionGraph has launched the single-instance multi-concurrency feature, allowing multiple requests to be processed concurrently on one instance.

This feature is suitable for functions which spend a long time to initialize or wait for a response from downstream services.

The feature has the following advantages:

- Fewer cold starts and lower latency: Usually, FunctionGraph starts three instances to process three requests, involving three cold starts. If you configure the concurrency of three requests per instance, only one instance is required, involving only one cold start.
- Shorter processing duration and lower cost: Normally, the total duration of multiple requests is the sum of each request's processing time. With this feature configured, the total duration is from the start of the first request to the end of the last request.

FunctionGraph automatically scales in or out function instances based on the number of requests. If the number of concurrent requests increases, FunctionGraph allocates more function instances to process the requests. If that number decreases, FunctionGraph allocates fewer function instances accordingly.

Number of function instances = Function concurrency/Concurrency per instance

- Function concurrency: the number of requests concurrently executed by a function at a certain time point.
- Concurrency per instance: the maximum number of concurrent requests allowed by a single instance. This is equivalent to the **Max. Requests per Instance** parameter on the **Concurrency** page.

## Comparison

If a function takes 5s to execute each time and you set the number of requests that can be concurrently processed by an instance to 1, three requests need to be processed in three instances, respectively. Therefore, the total execution duration is 15s.

When you set **Max. Requests per Instance** to **5**, if three requests are sent, they will be concurrently processed by one instance. The total execution time is 5s.

NOTE

If the maximum number of requests per instance is greater than 1, new instances will be automatically added when this number is reached. The maximum number of instances will not exceed **Max. Instances per Function** you set.

**Table 7-16** Comparison

| Comparison Item | Single-Instance Single-Concurrency | Single-Instance Multi-Concurrency |
| --- | --- | --- |
| Log printing | None | To print logs, Node.js uses the **console.info()** function, Python uses the **print()** function, and Java uses the **System.out.println()** function. In this mode, current request IDs are included in the log content.<br>However, when multiple requests are concurrently processed by an instance, the request IDs are incorrect if you continue to use the preceding functions to print logs. In this case, use **context.getLogger()** to obtain a log output object. For example, in Python:<br>`log = context.getLogger()`<br>`log.info("test")` |
| Shared variables | None | Modifying shared variables will cause errors. Mutual exclusion protection is required when you modify non-thread-safe variables during function writing. |
| Monitoring metrics | Perform monitoring based on the actual situation. | Under the same load, the number of function instances decreases significantly. |
| Flow control error | None | When the number of requests exceeds the processing capability of instances, FunctionGraph performs flow control on the requests. The error code in the body is **FSS.0429**, the status in the response header is **429**, and the error message is **Your request has been controlled by overload sdk, please retry later**. |

## Notes and Constraints

- This feature is supported only by FunctionGraph v2.
- **This feature is supported only for HTTP functions created from scratch,using function templates, or created based on container images**.
  - If the number of concurrent requests per instance is set to **1** for an existing function created in another way, the value cannot be changed.
  - To use the single-instance multi-concurrency feature for functions created in other ways, **submit a service ticket** to apply for whitelist access.

- For Python functions, threads on an instance are bound to one core due to the Python Global Interpreter Lock (GIL) lock. As a result, concurrent requests can only be processed using the single core, not multiple cores. The function processing performance cannot be improved even if larger resource specifications are configured.
- For Node.js functions, the single-process single-thread processing of the V8 engine results in processing of concurrent requests only using a single core, not multiple cores. The function processing performance cannot be improved even if larger resource specifications are configured.

## Configuring Single-Instance Multi-Concurrency

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Concurrency**.
4. Configure the concurrency for the function.

**Figure 7-12** Concurrency configuration

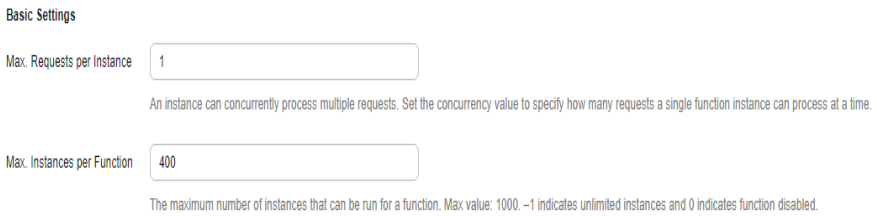



**Table 7-17** Concurrency parameters

| Parameter | Description |
| --- | --- |
| Max. Requests per Instance | Number of concurrent requests supported by a single instance.<br>**Restrictions:**<br>• This parameter is only available for HTTP functions created from scratch, HTTP functions based on templates, and container image-based functions.<br>• If this parameter is set to 1, this parameter is not displayed and cannot be modified. If this parameter is set to a value greater than 1, this parameter is still displayed.<br>**Value range**:<br>1–1000<br>**Default value**:<br>1 |

| Parameter | Description |
| --- | --- |
| Max. Instances per Function | Maximum number of on-demand instances that can be enabled for a function.<br>**Restrictions:**<br>• Requests that exceed the processing capability of instances will be discarded.<br>• Errors caused by excessive requests will not be displayed in function logs. You can obtain error details by referring to **Asynchronous Notification Policy**.<br>**Value range**:<br>-1 or an integer ranging from 1 to 1000. The value **–1** indicates that the number of instances is not limited.<br>**Default value**:<br>400 |

5. Click **Save**.

## Helpful Links

For details about function instance types and their usage modes, see **Function Instance Types and Usage Modes**.

# 7.7 Versions

FunctionGraph allows you to publish one or more versions throughout the development, test, and production processes to manage your function code. Each version is a complete snapshot of the function at a specific point in time, corresponding to a tag in the code. It includes independent code, configuration, and dependency, making it convenient for rollback, parallel deployment, or feature validation.

After a function is created, the default version is **latest**. Each function has the **latest** version.

After a function version is published, you can modify the version configuration as required. However, the code of the version cannot be updated to ensure version stability and traceability.

## Notes and Constraints

- You can release a maximum of 20 versions (including the latest version) for a function.
- No triggers, mounting disks, or reserved instances are configured in a new version by default.
- The **latest** version of a function cannot be deleted.
- If a function version associated with aliases is deleted, the aliases will also be deleted.

- Deleting a version will permanently delete the associated code, configuration, alias, and event source mapping, but will not delete logs. Deleted versions cannot be recovered. Exercise caution when performing this operation.

## Publishing a Version

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. On the **Versions** tab, click **Publish New Version**. The new version is published as the **latest** version.

**Figure 7-13** Parameters for publishing a new version

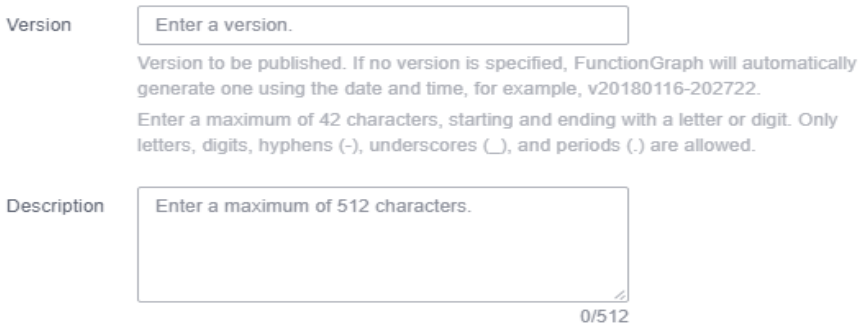



**Table 7-18** Version configuration parameters

| Parameter | Description |
| --- | --- |
| Version | Enter a version number.<br>The value can contain a maximum of 42 characters, including letters, digits, hyphens (-), underscores (_), and periods (.). It must start and end with a letter or digit. If no version number is specified, the system automatically generates a version number based on the current date, for example, **v20220510-190658**. |
| Description | Description of the version. The value can contain a maximum of 512 characters. |

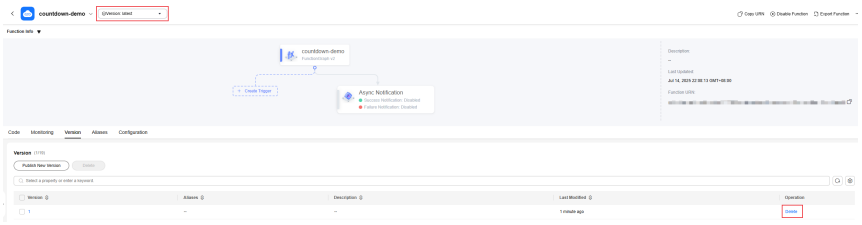
4. Click **OK**. The system automatically publishes a version. Then you will be redirected to the new version.

## Deleting a Function Version

1. On the function details page, select the **latest** version.
2. On the **Versions** tab page, you can view the created version list.
3. Click **Delete** on the right of the version. In the displayed dialog box, enter **DELETE** and click **OK**.

**Figure 7-14** Deleting a Version



4. Click **OK** to delete the version.

## Helpful Links

Manage function versions using APIs. For details, see **Function Version and Alias APIs**.

# 7.8 Aliases

FunctionGraph allows you to create aliases for functions and associate aliases with specified function versions. When you invoke a function using an alias, the function of the specified version is invoked. In actual service scenarios, you can update or roll back a function version by modifying the version configured for an alias. The client is unaware of the change.

An alias can point to up to two versions with different weights for dark launch.

## Notes and Constraints

- You can create up to 10 aliases for a function.
- A maximum of two function versions can be configured for an alias.

## Creating an Alias

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. On the **Aliases** tab page, click **Create Alias**. Create an alias to point to the current function version. You can use the alias to invoke this version.

**Table 7-19** Parameters for creating an alias

| Parameter | Description |
| --- | --- |
| Alias | Custom alias name. The name can contain a maximum of 63 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. |
| Version | Select the function version to be associated. Only one alias can be created for each version. |

| Parameter | Description |
| --- | --- |
| Traffic Shifting | Choose whether to enable traffic shifting. If this function is enabled, you can distribute a specific percentage of traffic to the additional version. |
| Additional Version | This parameter is mandatory only when **Traffic Shifting** is enabled.<br>Select an additional version to be associated. The **latest** version cannot be used as an additional version. For details about how to create a version, see **Versions**. |
| Shift By | This parameter is mandatory only when **Traffic Shifting** is enabled.<br>Select the traffic shifting type of the additional version.<br>• **Percentage**: Set a weight to shift the corresponding percentage of requests to the additional version. For example, if you set the percentage to 5%, FunctionGraph will forward 5% of requests to the additional version and the remaining to the main version.<br>• **Rule**: Requests that meet the specified rules will be forwarded to the additional version. **This mode is supported only by HTTP functions or functions using the APIG trigger.** |
| Weight | This parameter is mandatory only when **Traffic Shifting** is enabled and **Shift By** is set to **Percentage**.<br>Percentage of requests to be forwarded to the additional version. |
| Rule Type | This parameter is mandatory only when **Traffic Shifting** is enabled and **Shift By** is set to **Rule**.<br>Select a pattern that meets the rule conditions.<br>• **All rules met**: A request is sent to the additional version only when it meets all the rules in the rule list.<br>• **Any rule met**: A request is sent to the additional version as long as it meets any rule in the rule list. |

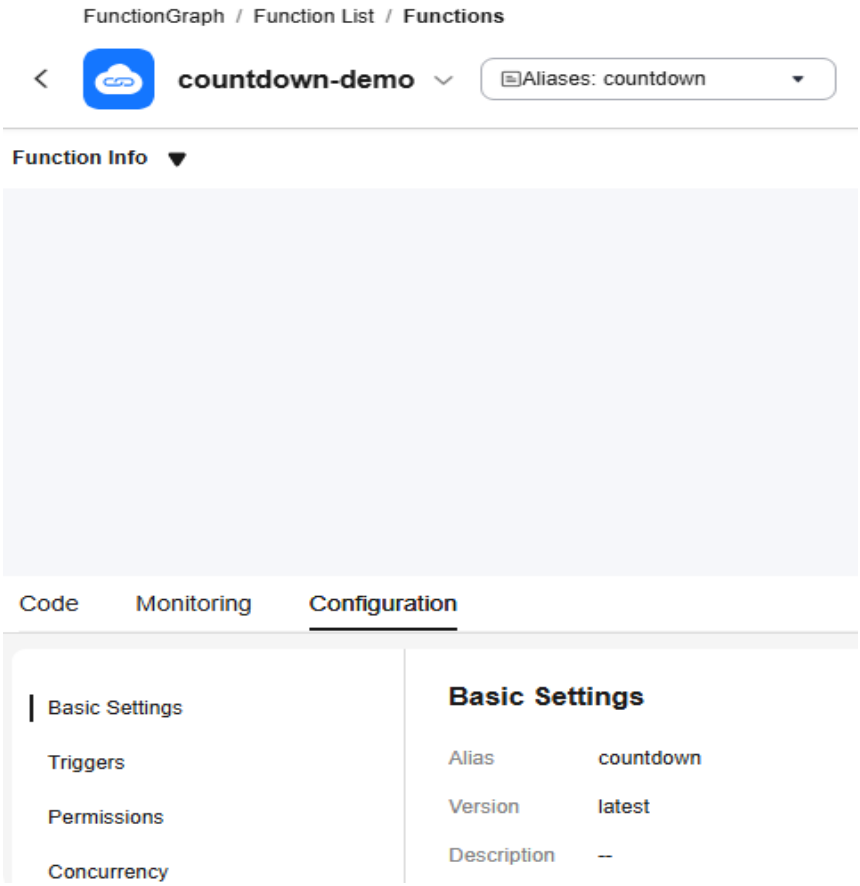
  

| Parameter | Description |
| --- | --- |
| Rules | This parameter is mandatory only when **Traffic Shifting** is enabled and **Shift By** is set to **Rule**.<br>Set the rule conditions that the function request must meet.<br>• **Parameter Type**: The default value is **Header**, indicating the header parameter in an HTTP request.<br>• **Parameter**: name of the header parameter in an HTTP request. The value is case insensitive.<br>• **Condition**: condition type that the header parameter must meet. Options: **=** and **in**.<br>• **Value**: value of the header parameter. The value is a character string.<br>– If the condition is set to **=**, the condition is met only when the value of the header parameter in the request is the same as the specified value.<br>– If the condition is set to **in**, you can set multiple values and separate them with commas (,). The condition is met when the value of the header parameter meets one of the values.<br>Rule 1: **Parameter** is set to **aaa**, **Condition** is set to **=**, and **Value** is set to **123**; Rule 2: **Parameter** is set to **bbb**, **Condition** is set to **in**, and **Value** is set to **111,222,333**. If the HTTP request contains the header parameter **aaa=123**, rule 1 is met. If the HTTP request contains the header parameter **bbb=222**, rule 2 is met. If the HTTP request contains the header parameters **aaa=123** and **bbb=111**, both rule 1 and rule 2 are met. |
| Description | Alias description. The value can contain up to 512 characters. |

4. Click **OK**. The alias function details page is displayed.

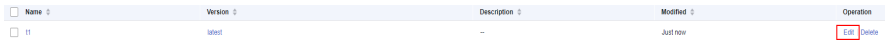
**Figure 7-15** Creating an alias




## Managing Function Aliases

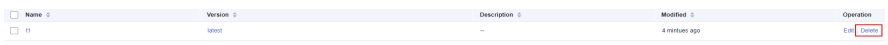
1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. On the function details page, select the **latest** version.
4. On the **Aliases** tab, you can view the created aliases.
   - Click **Edit** on the right of an alias to modify the alias configuration.

     **Figure 7-16** Modifying an Alias

   - Click **Delete** on the right of the alias. In the displayed dialog box, click **OK**.

     **Figure 7-17** Deleting an Alias

## Helpful Links

Manage function versions using APIs. For details, see **Function Version and Alias APIs**.

# 7.9 Tags

Tags are used to identify and classify functions. A tag can be associated with multiple functions, and a function can be associated with multiple tags.

You can configure function tags from the following two tag sources:

- Global resource tags created in Tag Management Service (TMS)
- Tags created in FunctionGraph

## Notes and Constraints

- Each function can have a maximum of 20 tags.
- If your organization has configured tag policies for FunctionGraph, add tags to functions based on the policies. If a tag does not comply with the tag policies, function creation may fail. Contact your administrator to learn more about tag policies.

## Prerequisites

To use the predefined tags, enable TMS. For details, see **Permissions**.

## Adding a Tag

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Tags**.
4. Click **Edit Tag**. In the displayed dialog box, click **Add**.
5. Enter a tag key and value.

   Each tag consists of a key-value pair. Each key must be unique and have only one value.

   **Table 7-20** Tag naming rules

| Parameter | Rule |
| --- | --- |
| Key | • Cannot be empty.<br>• Cannot start with _**sys**_ or a space, or end with a space.<br>• Can contain UTF-8 letters, digits, spaces, and the following characters: _.:=+-@<br>• Must be unique and cannot exceed 128 characters. |

| Parameter | Rule |
| --- | --- |
| Value | • Can be an empty string.<br>• Can contain UTF-8 letters, digits, spaces, and the following characters: _.:/=+-@<br>• Max. 255 characters. |

6. Click **OK**.

### Searching for Functions by Tag

1. Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. In the search box, select the **Tag** filter, and then select one or more key-value pairs.
3. Press **Enter** and you can view the search result in the function list.

### Helpful Links

In addition to using the console, you can query function tags using APIs. For details, see **Querying Function Tags**.

## 7.10 Dynamic Memory

This section describes how to configure dynamic memory for functions on the FunctionGraph console to reduce costs.

### Scenarios

By default, a function is bound with only one resource specification. Dynamic memory allows you to set the resource specifications of the processing function instance when processing a specified request. If no resource specification is specified, the function uses the default.

Take video transcoding as an example. The size of a video file ranges from MB to GB. Different encoding formats and resolutions require different computing resources. To ensure performance, you usually need to configure a large resource specification, which however will result in a waste during low-resolution video (such as short video) processing. To solve this problem, implement the transcoding service with two functions: metadata obtaining and transcoding. Configure a specification for the transcoding function according to the metadata information to minimize the resources and cost.

### Notes and Constraints

- The value of dynamic memory must be 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2560, 3072, 3584, 4096, 8192 or 10240, in MB.
- If dynamic memory is not enabled, the memory size set in **Configuring Basic Settings** is used when the API is called to execute the function.

- If dynamic memory is enabled but the memory size is not set, the memory size set in **Configuring Basic Settings** is used when the API for executing a function asynchronously or synchronously is called.
- If dynamic memory is enabled but the memory size is not within the specified range. When the API is called, error code FSS.0406 is returned.

### Configuring Dynamic Memory

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Advanced Settings**.
4. Enable **Dynamic Memory** and click **Save**.

### Helpful Links

For details about how to call an API for executing a function synchronously or asynchronously, see **Executing a Function Synchronously** and **Executing a Function Asynchronously**.

## 7.11 Heartbeat Function

This section describes how to configure a heartbeat function on the FunctionGraph console to detect function exceptions.

### Scenarios

The heartbeat function monitors user functions for issues like deadlocks, memory overflow, or network errors.

After a heartbeat function is configured, FunctionGraph sends a heartbeat request to the function instance every 5 seconds. If the response is abnormal, FunctionGraph terminates the function instance.

The heartbeat request timeout is 3s. If no response is returned for six consecutive times, the function instance will be stopped.

### Notes and Constraints

- Currently, heartbeat detection can be configured for Java functions.
- The heartbeat function entry must be in the same file as your function handler.

  Java heartbeat function format:

  ```
  public boolean heartbeat() {
      // Custom detection logic
      return true
  ```

- The heartbeat function has no input parameter and its return value is Boolean.

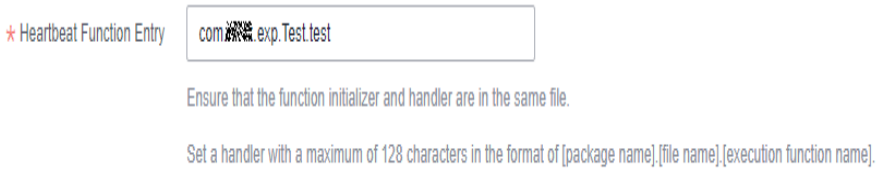
  

### Configuring Heartbeat Function

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Advanced Settings**.
4. Enable **Heartbeat Function** and set the function entry.

   The heartbeat function entry must be in the same file as the handler. Format: *[package name].[class name].[execution function name]*. The value can contain a maximum of 128 characters.

   **Figure 7-18** Configuring Heartbeat Function

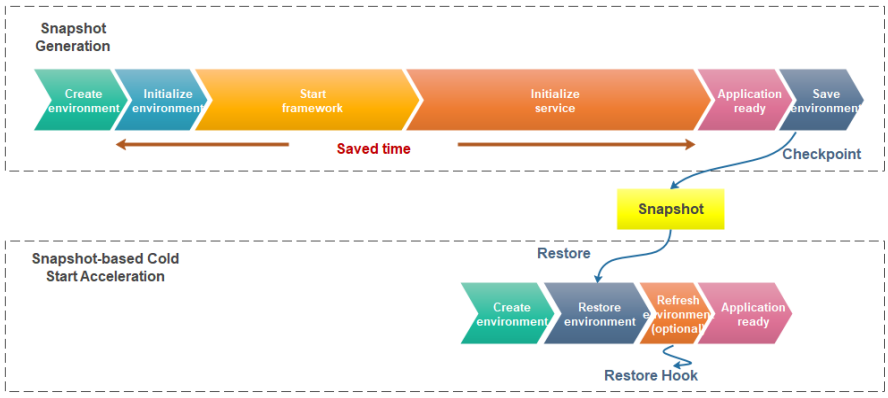
5. Click **Save**.

## 7.12 Snapshot-based Cold Start

### Overview

The snapshot-based cold start solution is a performance optimization service. It requires only simple configurations and a few code changes.

After snapshot-based cold start is enabled for a Java function, FunctionGraph executes the initialization code of the function, captures a snapshot of the context, and caches the snapshot after encryption. For subsequent invocations, it **restores the execution environment from the snapshot** instead of repeating the initialization process, significantly increasing the startup speed.




### Notes and Constraints

- Only Java functions support snapshot-based cold start.
- For functions that strongly depend on states, consider using Restore Hook to refresh their states.

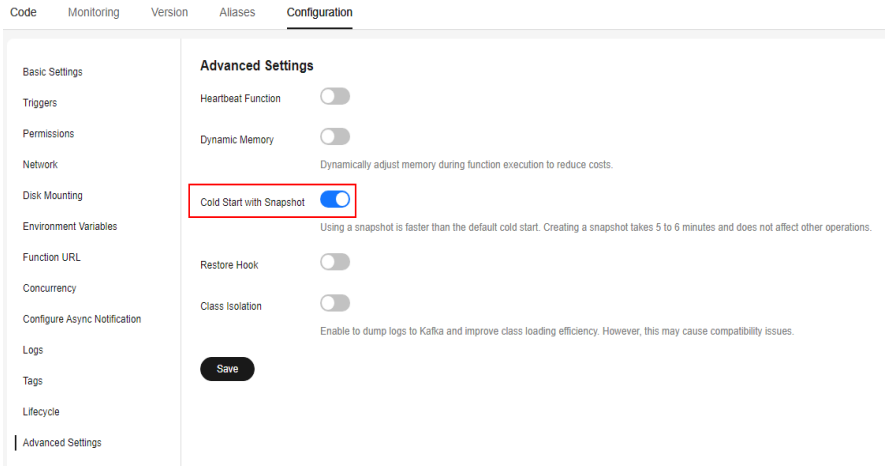
  

- For functions that strongly depend on CPU instruction sets, contact technical support by **submitting a service ticket** to determine whether snapshot-based cold start is supported.
- When functions that depend on the hard-coded host environment (such as hostname or hostip) are migrated to other hosts, snapshot-based cold start may fail. **Submit a service ticket** to contact technical support for confirmation. You are advised not to depend on these variables.
- Currently, snapshot-based cold start supports only applications developed using x86 machines.

## Configuring Snapshot-based Cold Start

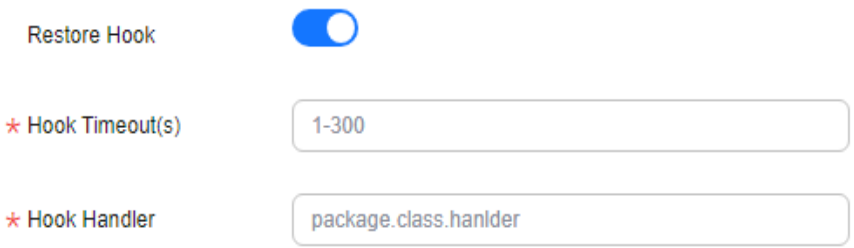
1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function. Choose **Configuration** > **Advanced Settings**.
3. Enable **Cold Start with Snapshot**.

**Figure 7-19** Enabling Snapshot-based Cold Start




4. (Optional) If the function strongly depends on state, configure Restore Hook to update the state and implement the corresponding hook logic in the function code.
   - **Hook Timeout (s)**: timeout interval for executing the hook handler.
   - **Hook Handler**: Enter the hook handler in the format of *[Package name].[Class name].[Execution function name]*. The value can contain a maximum of 128 characters.

**Figure 7-20** Enabling Restore Hook



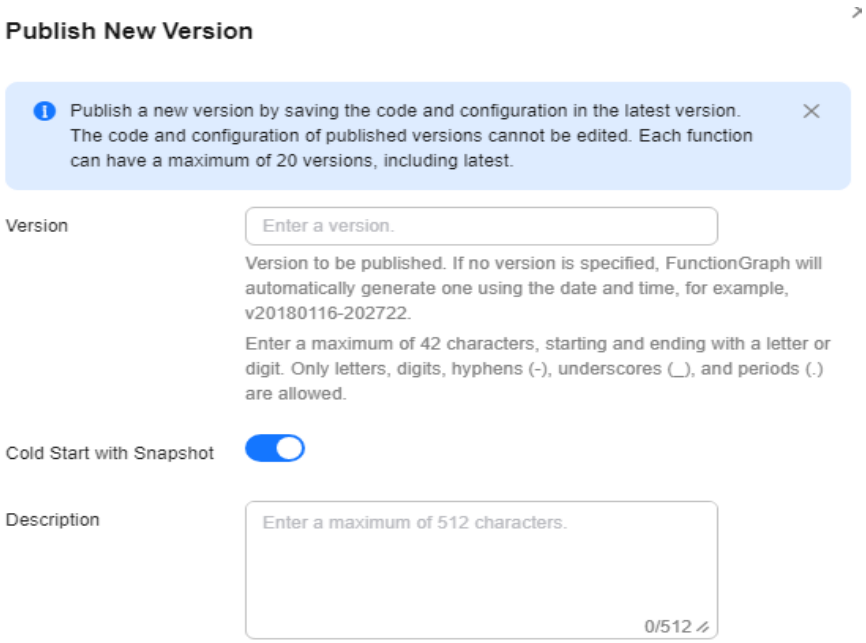

  

The following is an example of Restore Hook.

```
public class App
{
    public void restore(Context context) {
        System.setProperty("myKey", "restoreValue");
    }

    public void init(Context context) {
        System.setProperty("myKey", "initValue");
    }

    2 usages
    public void print(APIGTriggerEvent event, Context context) {
        System.out.println("value: " + System.getProperty("myKey"));
    }
```

5. Click **Save**.
6. Publish a function version by referring to **Publishing a Version**. Enable **Cold Start with Snapshot** to trigger snapshot creation.

**Figure 7-21** Publishing a new version




7. Wait until the snapshot is created (timeout: 5 minutes).

**Figure 7-22** Creating snapshot...

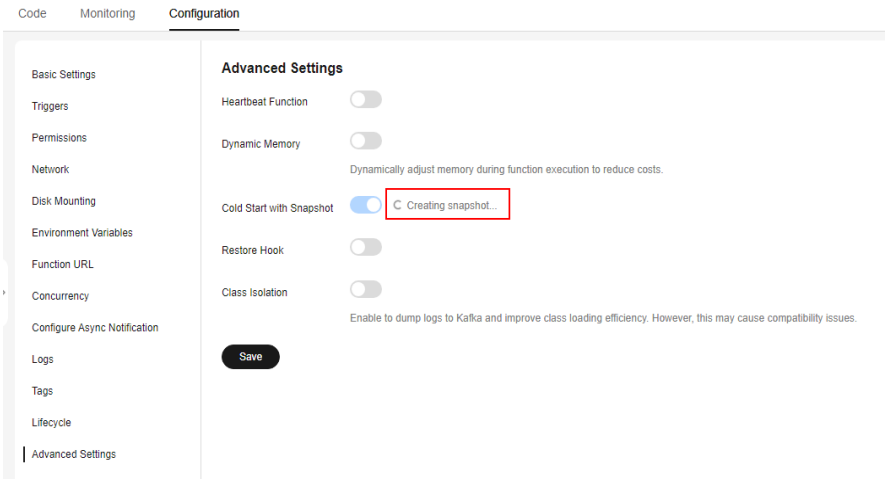
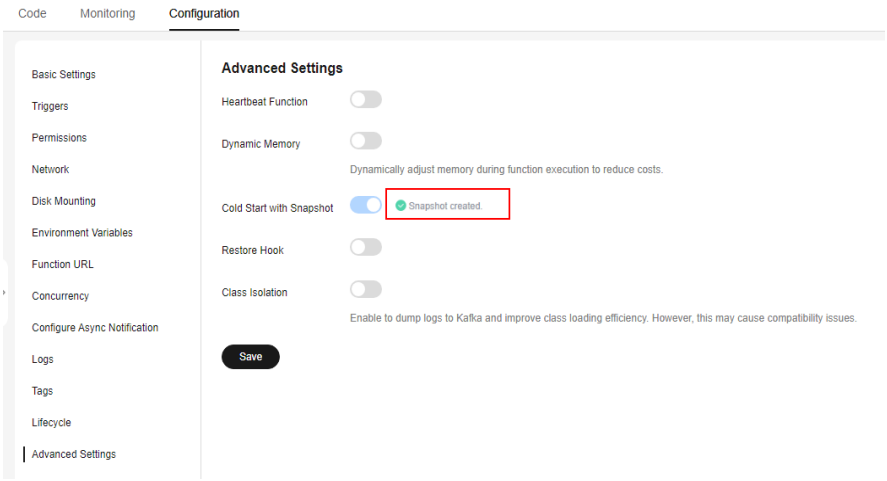



**Figure 7-23** Snapshot created




8. Invoke the Java function with snapshot cold start enabled and view the snapshot cold start information in the execution log, as shown in **Figure 7-24**.

**Figure 7-24** Startup logs

### Helpful Links

- Manage the function lifecycle using APIs. For details, see **Function Lifecycle Management APIs**.
- For details about how to optimize cold starts, see **FunctionGraph Cold Start Optimization**.

## 7.13 WebSocket

This section describes how to configure WebSocket connections on the FunctionGraph console.

### Overview

In FunctionGraph, you can configure an APIG trigger to enable a function to respond to WebSocket requests. Once configured, the associated function can act as a web server to handle WebSocket connections and messages, returning results to clients. This solution is suitable for scenarios requiring persistent connections, real-time messaging, and live data monitoring, enabling efficient real-time communication and data interaction.

The billing of WebSocket is the same as that of HTTP. You can consider WebSocket as an HTTP call with a longer connection time. For details on billing, see **FunctionGraph Billing Overview**.

### Notes and Constraints

- **WebSocket connection is only supported by HTTP functions.**
- Currently, WebSocket connection can be configured only in the **CN East-Shanghai1** region.
- The configured execution timeout applies equally to both WebSocket and HTTP requests. If the WebSocket connection duration exceeds the execution timeout, the connection will be closed and the client will receive status code **1006**.

### Step 1: Creating an HTTP Function

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

**Step 2** On the **Function List** page, click **Create Function** in the upper right corner.

**Step 3** Click **Create from scratch** and configure the function information by referring to **Table 7-21**.

**Table 7-21** Function configuration

| Parameter | Description | Example Value |
| --- | --- | --- |
| Function Type | Select the function type. To configure WebSocket connection, you need to create an HTTP function.<br>The HTTP functions process HTTP requests. You can send an HTTP request to a URL to trigger the function and use your web services. This feature is supported only by FunctionGraph v2. | HTTP Function |
| Region | Select a region where you will deploy your code.<br>Currently, WebSocket connection can be configured only in the **CN East-Shanghai1** region. | CN East-Shanghai1 |
| Function Name | Name of the function, which must meet the following requirements:<br>Consists of 1 to 60 characters, and can contain letters, digits, hyphens (-), and underscores (_). Start with a letter and end with a letter or digit. | WebSocket-demo |
| Enterprise Project | 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. If you have not enabled the enterprise management service, this parameter is not displayed. For details, see **Enabling the Enterprise Project Function**. | default |
| Agency | Select an agency for the function. An agency is used to authorize FunctionGraph to access other cloud services. For example, if FunctionGraph needs to access LTS or VPC, you must select an agency with required service permissions. If FunctionGraph does not access any cloud services, you do not need to select an agency.<br>By default, **Use no agency** is used. You can select an existing agency.<br>If no default agency is available, FunctionGraph allows you to quickly create a default agency named **fgs_default_agency**. For details, see **Default Agency**. | Use no agency |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Advanced Settings | • **Public Access**: When enabled, functions can access services on the public network. The public network access bandwidth is shared among users.<br>• **VPC Access**: When enabled, functions will use the NICs bound to the configured VPC for network access and the default FunctionGraph NIC will be disabled. | **Public Access**: Enabled.<br>**VPC Access**: Disabled. |

**Step 4** Click **Create Function**. The function details page is displayed.

**----End**

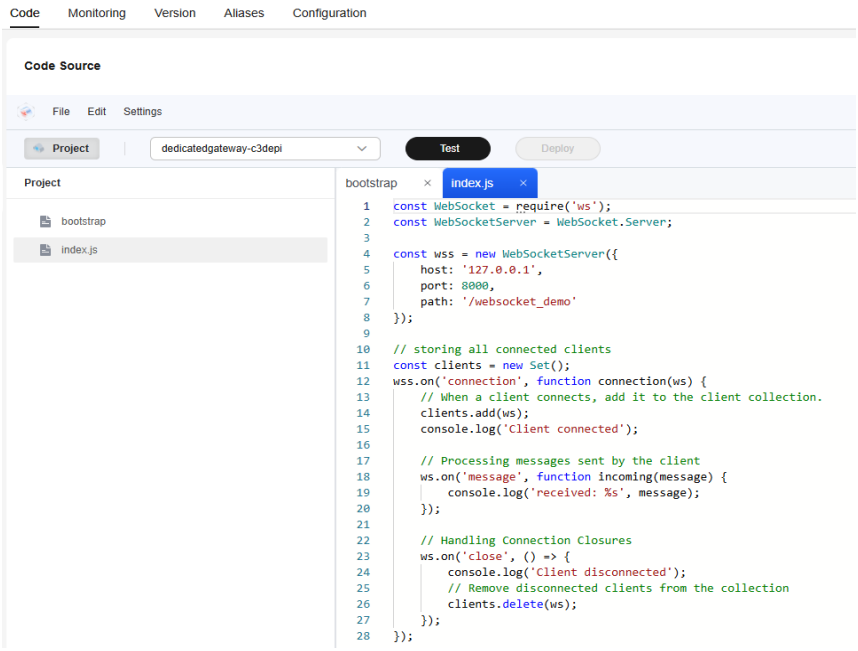
## Step 2: Configuring Function Code

**Step 1** On the **Code** tab page of the function details page, open the **index.js** file, copy and deploy the following code, as shown in **Figure 7-25**.

```
const WebSocket = require('ws');
const WebSocketServer = WebSocket.Server;

const wss = new WebSocketServer({
    host: '127.0.0.1',
    port: 8000,
    path: '/websocket_demo'
});

// Store all connected clients.
const clients = new Set();
wss.on('connection', function connection(ws) {
    //Add the connected client to the client set.
    clients.add(ws);
    console.log('Client connected');

    //Process messages sent by the client.
    ws.on('message', function incoming(message) {
        console.log('received: %s', message);
    });

    //Process the connection closure.
    ws.on('close', () => {
        console.log('Client disconnected');
        //Remove the disconnected client from the set.
        clients.delete(ws);
    });
});

// Send messages to all clients at a scheduled time.
setInterval(() => {
    // Traverse all clients and send messages.
    for (let ws of clients) {
        if (ws.readyState === WebSocket.OPEN) {
            ws.send('Server time: ' + new Date().toTimeString());
        }
    }
}, 2000); // Executed every 2 seconds.

console.log('WebSocket server is running on ws://127.0.0.1:8000/websocket_demo');
```

**Figure 7-25** index.js file



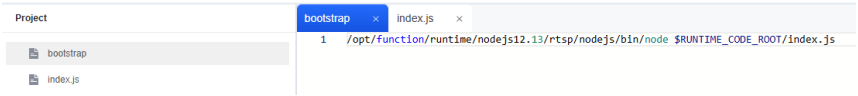

**NOTE**

The listening IP address of the WebSocket server is **127.0.0.1**, and the default listening port is **8000**.

**Step 2** Add the following content to the **bootstrap** file, as shown in **Figure 7-26**.

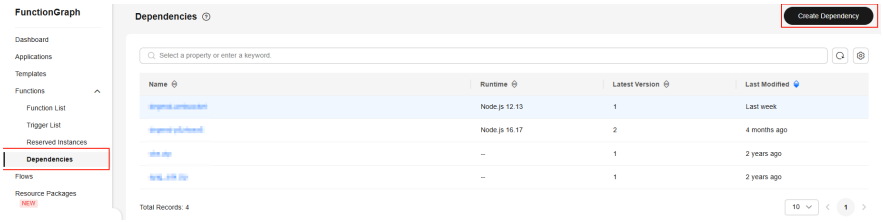
```
/opt/function/runtime/nodejs12.13/rtsp/nodejs/bin/node $RUNTIME_CODE_ROOT/index.js
```

**Figure 7-26** bootstrap file




**Step 3** In the navigation pane, choose **Functions** > **Dependencies**, and click **Create Dependency**, as shown in **Figure 7-27**.

**Figure 7-27** Dependency management




**Step 4** Download the **wss-nodejs12** file, upload and configure the WebSocket dependency package on the **Create Dependency** page, as shown in **Figure 7-28**, and click **Create**.

**Figure 7-28** Creating a dependency

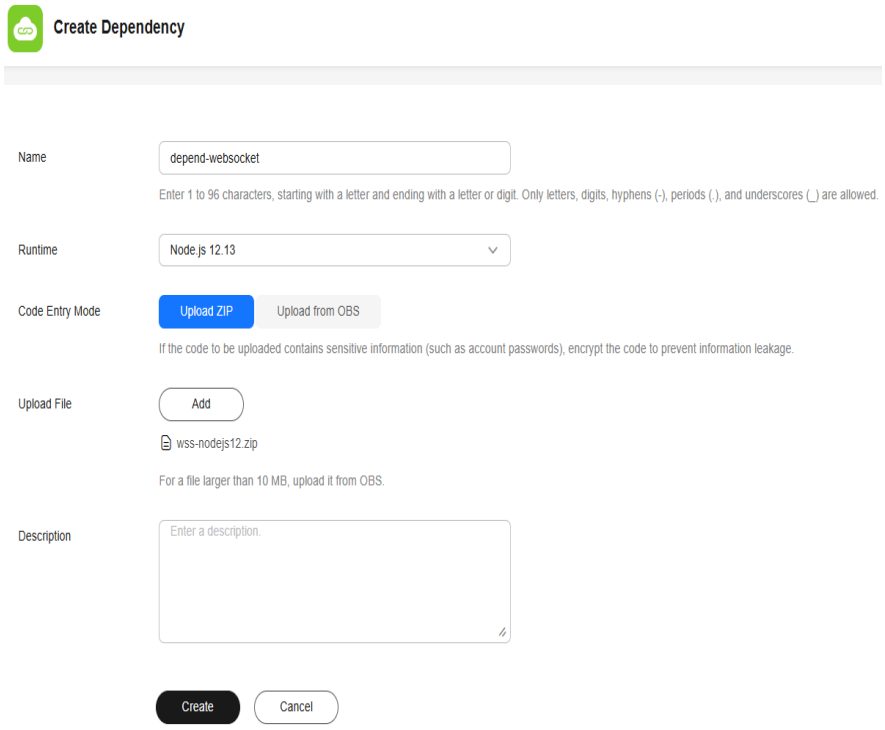



**Table 7-22** Parameters for creating a dependency

| Parameter | Description | Example Value |
| --- | --- | --- |
| Name | Enter a dependency name.<br>Start with a letter and end with a letter or digit.<br>The value can contain a maximum of 96 characters, including letters, digits, underscores (_), periods (.), and hyphens (-). | depend-websocket |
| Runtime | Select the runtime of the dependency. | Node.js 12.13 |
| Code Entry Mode | Select a code upload mode. You can select **Upload ZIP** or **Upload from OBS**. | Upload ZIP |

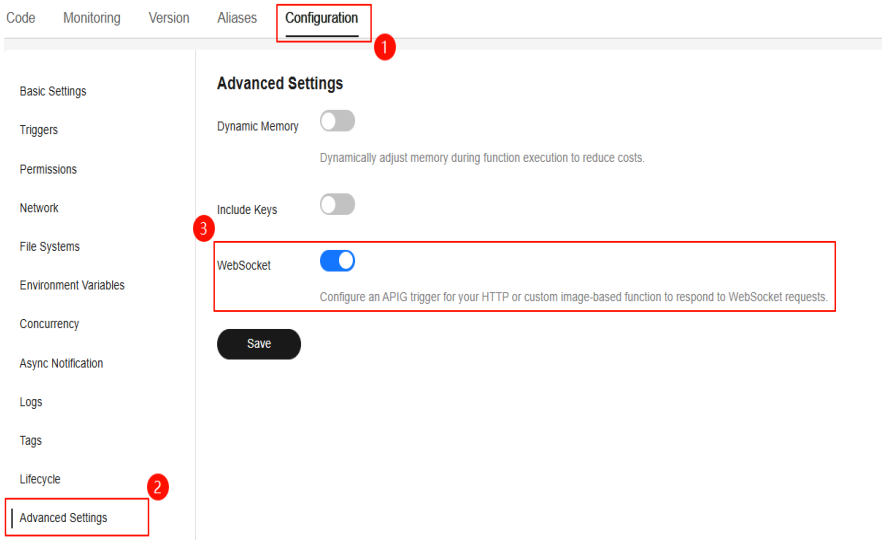
| Parameter | Description | Example Value |
| --- | --- | --- |
| Upload File/OBS Link URL | • If **Upload ZIP** is selected, click **Add** to upload the dependency packaged in ZIP format. The size of the ZIP file to be uploaded cannot exceed 10 MB. If the file size exceeds 10 MB, upload it from OBS.<br>• If **Upload from OBS** is selected, enter the OBS object URL that points to the code file object. The object must be in ZIP format and must be stored in the OBS bucket in the same region as the function. Copy the URL of the code file object by referring to **Sharing Objects with Anonymous Users Using URLs**. | Add a file and upload the **wss-nodejs12** file downloaded in **Step 4**. |
| Description | Enter the description of the dependency. | - |

**Step 5** Return to the **WebSocket-demo** function details page. On the **Code** tab, click **Add** in the **Dependencies** area at the bottom.

**Step 6** In the displayed dialog box, set **Type** to **Private**, add the **depend-websocket** dependency created in **Step 4**, and click **OK**.

**Step 7** Choose **Configuration** > **Advanced Settings** and enable **WebSocket**, as shown in **Figure 7-29**.

**Figure 7-29** Enabling WebSocket




**----End**

## Step 3: Creating 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:

**Table 7-23** Notes for creating a gateway

| Parameter | Description |
| --- | --- |
| Region | Select the region where the function is created. |
| Public Inbound Access | • If you perform a local test as shown in **Step 5: Testing the Function**, enable **Public Inbound Access** and select the inbound bandwidth based on your requirements.<br>• If your function is configured with VPC access and WebSocket connection in the production environment, you do not need to enable **Public Inbound Access**. |
| Network | If you use the WebSocket feature in the production environment, you can configure **VPC Access** for the function and configure the same VPC and subnet for the gateway. |

## Step 4: Creating an APIG Trigger

**Step 1** Choose **Configuration** > **Triggers**, click **Create Trigger**, and create a dedicated APIG trigger by referring to **Table 7-24**.

**Table 7-24** Parameters for creating a dedicated APIG trigger

| Parameter | Description | Example Value |
| --- | --- | --- |
| Trigger Type | Mandatory.<br>Select **API Gateway (Dedicated)**. | API Gateway (Dedicated) |
| API Instance | Mandatory.<br>Select an APIG dedicated gateway. If no gateway is available, click **Create Instance**. | apig-ws |
| API Name | Mandatory.<br>Name of a dedicated APIG trigger. Enter 3 to 64 characters, starting with a letter. Only letters, digits, and underscores (_) are allowed. | API_websocket_demo |

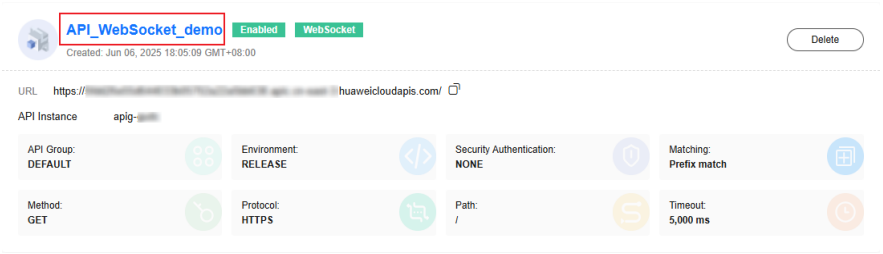
| Parameter | Description | Example Value |
| --- | --- | --- |
| API Group | Mandatory.<br>Select an API group. An API group is a collection of APIs. You can manage APIs by API group.<br>If no group is created, click **Create API Group**.<br>After the group is created, click  on the right. | APIGroup_ws |
| Environment | Mandatory.<br>The environment where the API is published. An API can be called in different environments, such as production, test, and development environments only when the parameter is set to **RELEASE**.<br>If no environment is available, click **Create Environment**. | RELEASE |
| Security Authentication | Mandatory.<br>API authentication modes are as follows:<br>• **App**: AppKey and AppSecret authentication. This mode is of high security and is recommended. For details, see **App Authentication**.<br>• **IAM**: IAM authentication. This mode grants access permissions to IAM users only and is of medium security. For details, see **IAM Authentication**<br>• **None**: No authentication. This mode grants access permissions to all users.<br>**CAUTION**<br>In this example, **None** is selected. You are advised to select a high-security authentication mode for actual services. | None |
| Protocol | Mandatory.<br>When WebSocket is used, HTTPS corresponds to WSS, and HTTP corresponds to WS.<br>There are two types of API request protocols:<br>• **HTTP**: Data is not encrypted during transmission.<br>• **HTTPS**: Data is encrypted during transmission. | HTTPS |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Method | Mandatory.<br>To use the WebSocket protocol, the request method must support at least GET.<br>Options: GET, POST, DELETE, PUT, PATCH, HEAD, OPTIONS, and ANY. | GET |
| Timeout (ms) | Mandatory.<br>API backend timeout in milliseconds. Range: 1–60,000.<br>The backend timeout indicates the maximum idle connection time of WebSocket. For example, if the timeout is set to 5000 ms, the connection is disconnected when WebSocket does not receive or send messages for more than 5000 ms. | 5000 |

**Step 2** After the trigger is created, click the trigger name to access the APIG console, as shown in **Figure 7-30**.

**Figure 7-30** APIG trigger




**NOTE**

After the APIG trigger is created, its URL will not change.

**Step 3** On the API details page, click **Modify**, as shown in **Figure 7-31**. The page for modifying an API is displayed.

**Figure 7-31** Modifying an API


**Step 4** Click **Next** to go to the **Default Backend** tab, as shown in **Figure 7-32**, set API parameters, and click **Finish**.

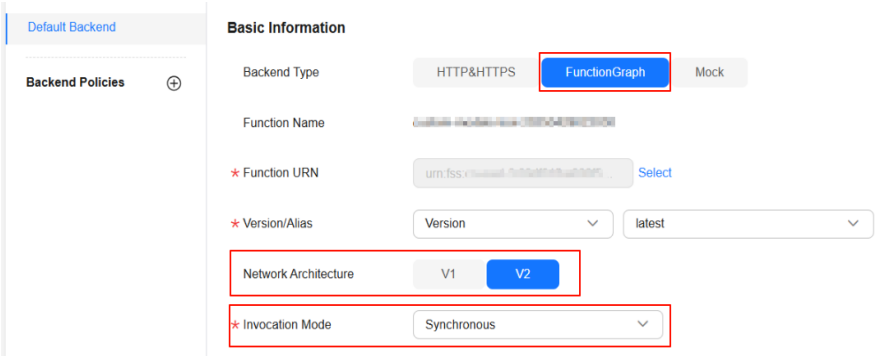
- **Backend Type**: Select **FunctionGraph**.
- **Network Architecture**: Select **V2**. (If this parameter is not available, **submit a service ticket** to enable the whitelist.)

- **Invocation Mode**: Select **Synchronous**. In WebSocket scenarios, only the synchronous invocation mode is supported.

  Retain default values for other parameters.

**Figure 7-32** Modifying the default backend of the API




**Step 5** Return to the API details page and click **Publish Latest Version**, as shown in **Figure 7-33**.

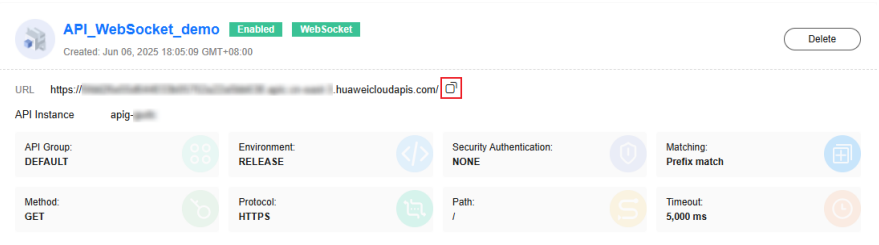
**Figure 7-33** Publishing the API of the latest version


**----End**

## Step 5: Testing the Function

**Step 1** Return to the FunctionGraph console, go to the function details page, choose **Configuration** > **Triggers**, and copy the URL of the APIG trigger.

**Figure 7-34** Copying the URL of the APIG trigger




**Step 2** Use Postman to test the function.

1. Create a WebSocket request in Postman. Copy the URL of the APIG trigger to Postman and change the scheme from HTTPS to WSS.
2. Configure Params and Headers parameters as required.
3. Connect to WebSocket. After the connection is successful, you can send messages and view the message receiving status.

4. After the execution times out, the connection to the WebSocket server is disconnected.

**----End**

## WebSocket Description

- **Connection Keepalive and Timeout Reconnection**

  After a WebSocket connection is established, FunctionGraph does not intervene in any logic unless the connection duration exceeds the specified execution timeout. If no data is transmitted through your WebSocket connection within a specified period, the connection may be closed by an intermediate network node (such as a NAT gateway). In this scenario, you need to use the ping and pong frames of the WebSocket protocol to keep the connection active or verify the validity of the WebSocket connection.

  If the service requirements exceed the maximum request timeout provided by FunctionGraph, or if the application needs to maintain logical connection stability during running, you are advised to add a timeout reconnection mechanism to the client code. You can use the **Reconnecting-WebSocket** or **SocketIO** library.

- **Session Affinity**

  FunctionGraph is a serverless computing platform. Its function instances use request-triggered lifecycle management. During high concurrency, the system automatically scales by creating multiple instance replicas but cannot guarantee routing consecutive client requests to the same instance. For WebSocket applications that require session state maintenance, use external storage systems such as Redis, Kafka, and databases to implement cross-instance state synchronization.

  For example, in a chat room application, all users may not be connected to the same function instance at the same time. You can use the Redis publish/subscribe function to broadcast messages. When a user joins a chat room, the function subscribes to the channel of the chat room. When the user sends a message, the function publishes the message to the channel of the chat room in Redis. Because all users in the chat room have subscribed to the channel, all users receive the message.

# 7.14 Streaming Response

## Overview

To meet the requirements of web and AI applications for real-time data transmission and large packet transmission, you can configure the streaming response for the function to return response packets to the client in HTTP streaming mode.

**Advantages:**

- Supports larger response packets (up to 200 MB).
- Reduces memory usage by processing and transmitting data in batches instead of transferring the entire packets at once.
- Enables faster time to first byte through streaming transmission.

**Scenarios:**

- Processing large files or big data sets (such as images and videos)
- Real-time data processing (for example, AI applications using Server-Sent Events (SSE) for data transmission)

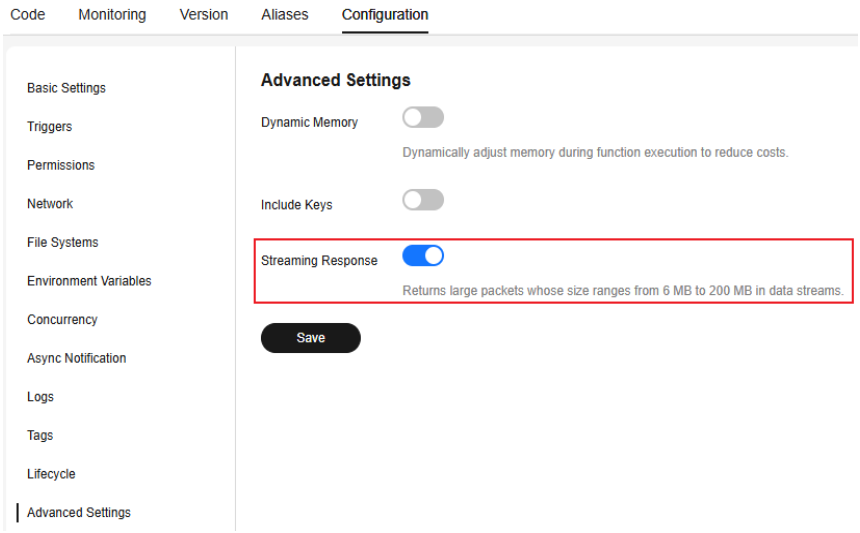
## Notes and Constraints

- Currently, streaming response is supported only in the **CN Southwest-Guiyang1** and **CN North-Beijing4** regions.
- **Streaming response is supported only for container image-based functions, custom runtime functions, and HTTP functions.**
- The maximum size of a streaming response is 200 MB.
- Streaming responses support only synchronous invocation, and the maximum execution timeout is 300s.

## Configuring Streaming Response

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function. Choose **Configuration** > **Advanced Settings**.
3. Enable **Streaming Response** and click **Save**.

**Figure 7-35** Enabling streaming response




## Compiling Code for a Function with Streaming Response Enabled

The following are code examples of container image-based functions, HTTP functions, and custom runtime functions.

## Sample Code of a Container Image-based Function

Create a container image-based function by referring to **Creating a Function with an Image** and enable streaming response. Note that the **function execution timeout** must be less than 300s. Otherwise, the function will fail to be saved.

To ensure that the function can correctly return streaming data, write streaming data in batches during the generation of the function response body to avoid memory waste. The following is an example of a simple Java Spring Boot application returning streaming data:

```
@SpringBootApplication
@RestController
public class SpringBootDemoHelloworldApplication {
    private ExecutorService executor = Executors.newCachedThreadPool();
    public static void main(String[] args) {
        SpringApplication.run(SpringBootDemoHelloworldApplication.class, args);
    }
    /**
     * Stream Response
     * @return
     */
    @PostMapping(value = "/invoke")
    public ResponseEntity<ResponseBodyEmitter> invoke(@RequestBody Map<String, Object> body) {
        ResponseBodyEmitter emitter = new ResponseBodyEmitter();
        executor.submit(() -> {
            try {
                emitter.send("hello \n", MediaType.TEXT_EVENT_STREAM);
                emitter.send("world \n", MediaType.TEXT_EVENT_STREAM);
                emitter.complete();
            } catch (Exception ex) {
                emitter.completeWithError(ex);
            }
        });
        HttpHeaders responseHeaders = new HttpHeaders();
        responseHeaders.setContentType(MediaType.TEXT_EVENT_STREAM);
        return new ResponseEntity(emitter, responseHeaders, HttpStatus.OK);
    }
}
```

## Sample Code of an HTTP Function

Create an HTTP function by referring to **Creating an HTTP Function** and enable streaming response. Note that the **function execution timeout** must be less than 300s. Otherwise, the function will fail to be saved.

For details about the sample code of an HTTP function, see **Sample Code of a Container Image-based Function**.

## Sample Code of a Custom Runtime Function

To return streaming data, the custom runtime function needs to call the POST method at **http://$RUNTIME_API_ADDR/v1/runtime/invocation/response/$REQUEST_ID**. Below is a simple Shell bootstrap example.

```
#!/bin/sh
set -o pipefail
#Processing requests loop
while true
do
  HEADERS="$(mktemp)"
  # Get an event
  EVENT_DATA=$(curl -sS -LD "$HEADERS" -X GET "http://$RUNTIME_API_ADDR/v1/runtime/invocation/
```

```
request")
  # Get request id from response header
  REQUEST_ID=$(grep -Fi x-cff-request-id "$HEADERS" | tr -d '[:space:]' | cut -d: -f2)
  if [ -z "$REQUEST_ID" ]; then
    continue
  fi
  # Send streamdata file content back to client
  curl -X POST "http://$RUNTIME_API_ADDR/v1/runtime/invocation/response/$REQUEST_ID" -H "Content-
Type:application/octet-stream" --data-binary "@code/streamdata.txt"
done
```

### Testing Streaming Response Functions

1. On the **Code** tab page of the function details page, click **Test**. The **Configure Test Event** dialog box is displayed.
2. Select **Blank Template**, and click **Create**.
3. Select the test event created in **2** and click **Test** again.
4. After the function is executed, the download is automatically started. The download file name is the request ID, and the file content is the return value of the request.

**Figure 7-36** File download


## 7.15 Class Isolation and Pre-stop for Java Functions

Class isolation is used to load your code and dependencies using an independent class loader if they conflict with the runtime dependencies.

Pre-stop is used to call a callback function before FunctionGraph stops the current function instance.

### Notes and Constraints

Only Java functions can be configured with class isolation and pre-stop.

### Configuring Class Isolation for a Java Function

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Advanced Settings**.
4. Enable **Class Isolation** and click **Save**.

### Configuring Pre-stop for a Java Function

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

2. Click the name of a function.
3. On the **Configuration** tab, click **Lifecycle**.
4. Enable **Pre-stop** and set the related parameters.

**Table 7-25** Pre-stop configuration

| Parameter | Description |
| --- | --- |
| Pre-stop Timeout (s) | Timeout for executing the callback function before the current function instance is stopped. The value is an integer ranging from 1 to 90. |
| Pre-stop Handler | Handler of the callback function, which can contain a maximum of 128 characters in the format of *[package name].[class name].[execution function name]*. |

5. Click **Save**.

# 7.16 Transferring Secret Keys Through the Request Header

## Transferring Secret Keys Through the Request Header

The key of an HTTP function can be transferred only through the request header. To obtain the AK, SK, and token of an HTTP function as shown in **Table 7-26**, perform the following steps.

1. Log in to the **FunctionGraph console** and go to the details page of the HTTP function to be configured.
2. Choose **Configuration** > **Advanced Settings**, enable **Include Keys**, and click **Save**.

**Figure 7-37** Transferring secret keys through the request header

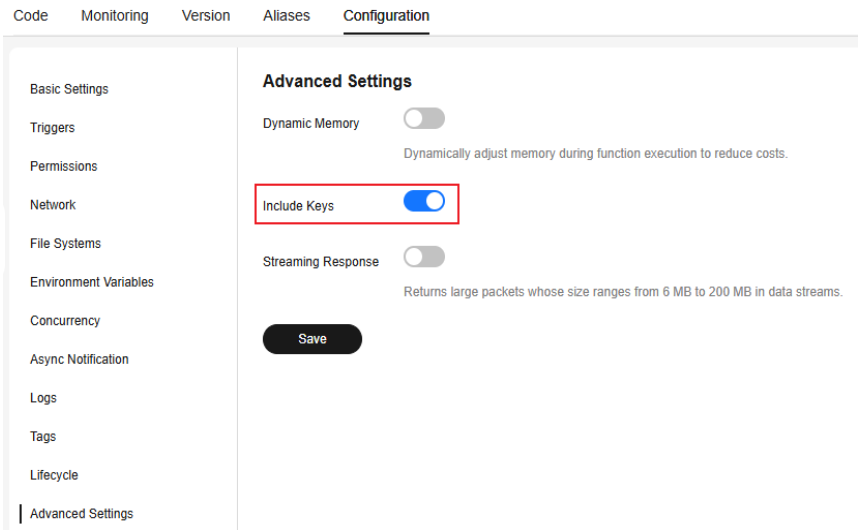


**Table 7-26** Transferring secret keys through the request header

| Field | Description |
| --- | --- |
| X-CFF-Auth-Token | A token is an access credential issued to an IAM user to bear its identity and permissions. |
| X-CFF-Security-Access-Key | A temporary access key and SecurityToken are issued by the system to IAM users.<br>The temporary AK/SK and SecurityToken follow the principle of least privilege.<br>A temporary AK/SK and SecurityToken must be used together. |
| X-CFF-Security-Secret-Key | |
| X-CFF-Security-Token | |

## Helpful Links

For details about HTTP functions, see **Creating an HTTP Function**.

# 7.17 Importing and Exporting Functions

FunctionGraph supports function import and export. You can export a function as a file to your local PC and import it to the FunctionGraph console of another region or user to migrate function configuration data.

## Notes and Constraints

**Table 7-27** Restrictions on function import and export

| Item | Description |
| --- | --- |
| Restrictions on function export | • The exported function resource package cannot exceed 50 MB.<br>• The exported function resource package does not include alias information.<br>• Only functions of the **latest** version can be exported from the function list.<br>• Encrypted environment variables can be exported. |

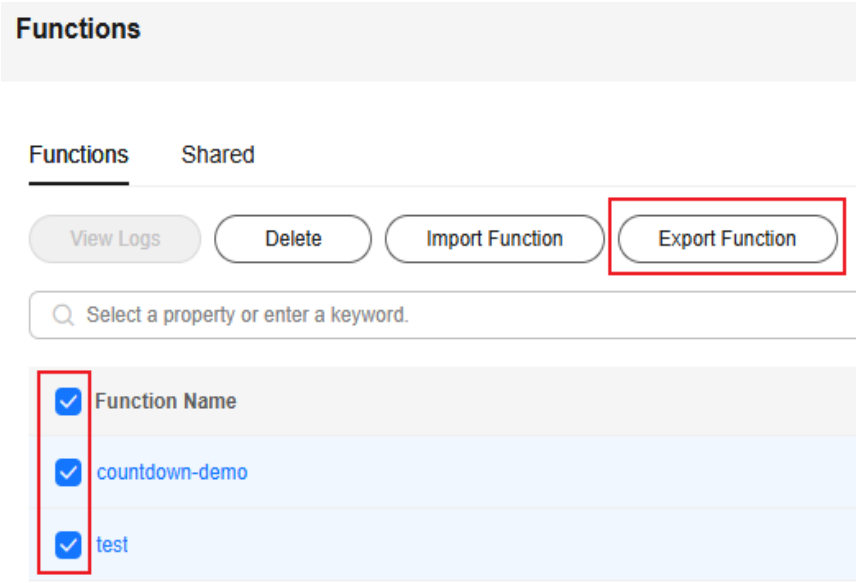
| Item | Description |
| --- | --- |
| Restrictions on function import | • When a function is imported across regions (for example, from **CN East-Shanghai1** to **CN North-Beijing4**), the encryption environment variable of the function may fail to be decrypted because the keys may be different in different regions. In this case, you need to reconfigure the encryption environment variables.<br>• If the code to be uploaded contains sensitive information (such as account passwords), encrypt the sensitive information to prevent leakage.<br>• The maximum file size is 10 MB.<br>• The uploaded file is a ZIP package that contains the function code (ZIP) and function configuration file (YAML). |

## Exporting a Function

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. On the **Functions** page, you can export functions using either of the following methods:
   - Exporting functions in batches: Select the functions to be exported from the function list and click **Export Function**. The exported functions are downloaded as a ZIP file.

     **Only the latest version of each function can be exported.**

     **Figure 7-38** Exporting functions in batches

   - Exporting a single function:

i. Click the name of the desired function.

ii. On the function details page, select the version of the function to be exported.

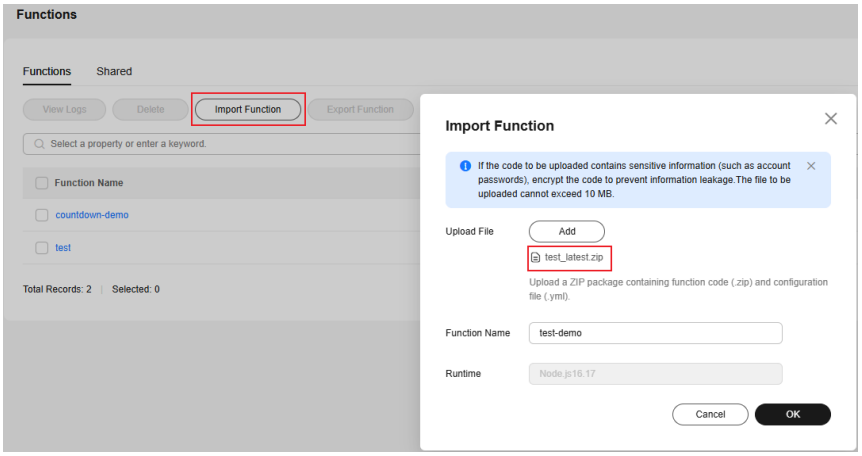
iii. In the upper right corner of the function details page, click **Export Function**.

**Figure 7-39** Exporting a function

## Importing a Function

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click **Import Function**. In the displayed dialog box, click **Add** and select the exported ZIP function file.

**Figure 7-40** Importing a function




3. The system automatically reads and configures the runtime of the function. Edit the function name and click **OK**.

## Helpful Links

Import and export functions using APIs. For details, see **Function Import and Export APIs**.

# 7.18 Enabling or Disabling Functions

FunctionGraph supports enabling and disabling functions. Disabled functions cannot be invoked.

## Notes and Constraints

- Only functions of the **latest** version can be disabled.

- Versions published based on the disabled **latest** version of a function are also disabled and can never be enabled.
- After disabling a function, you can modify its code but cannot execute the function.

### Disabling a Function

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of the desired function.
3. In the upper right corner of the function details page, click **Disable Function**. In the displayed dialog box, click **Yes**.

### Enabling a Function

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of the disabled function.
3. In the upper right corner of the function details page, click **Enable Function**.

## 7.19 Reserved Instances

### Overview

FunctionGraph provides on-demand and reserved instances.

- On-demand instances are created and released by FunctionGraph based on actual function usage. When receiving requests to invoke functions, FunctionGraph automatically allocates execution resources to the requests.
- Reserved instances are created and released by yourself. If a reserved instance is created for a function, FunctionGraph will forward invocation requests to the reserved instance. If the request peak exceeds the processing capability of the reserved instances, FunctionGraph sends the excess requests to on-demand instances and automatically allocates the required environments.

  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. **Do not execute one-time services using the initializer of reserved instances.**

FunctionGraph supports **Configuring a Fixed Number of Reserved Instances**, **Configuring a Scheduled Scaling Policy**, and **Configuring an Intelligent Recommendation Policy**.

NOTE

By default, you do not have permission to use the metric and intelligent recommendation policies. To use them, **submit a service ticket**.

## Notes and Constraints

- Reserved instances cannot be configured for both a function alias and the corresponding version. For example, if the alias of the latest version is **1.0** and reserved instances have been configured for this version, no more instances can be configured for alias **1.0**.
- After the idle mode is enabled, reserved instances are initialized and the mode change needs some time to take effect. You will still be billed at the price of reserved instances for non-idle mode in this period.
- If the function concurrency is greater than the number of reserved instances, the excess requests will be allocated to on-demand instances, which involve a cold start.

## Billing Description

You will be billed for reserved instances on a pay-per-use basis. For details about the billing rules and items, see **FunctionGraph Billing Items**.

## Prerequisites

You have configured an agency with permissions for querying AOM metrics and function configurations for the function. For details, see **Configuring Agency Permissions**.

## Configuring a Fixed Number of Reserved Instances

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of a function.
3. Choose **Configuration** > **Concurrency**.
4. Click **Add** and configure the reserved instance policy.

   The number of reserved instances cannot exceed the concurrency quota and **Max. Instances per Function**.

**Figure 7-41** Basic settings

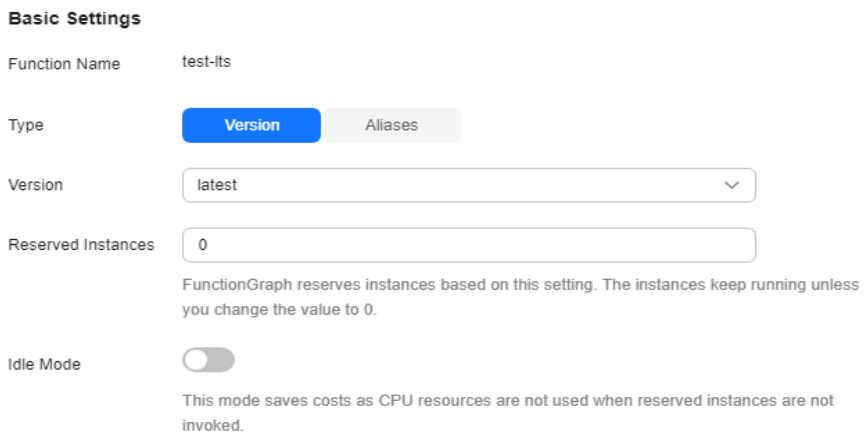


**Table 7-28** Basic settings

| Parameter | Description |
| --- | --- |
| Function Name | Name of the current function. |
| Type | Whether to configure reserved instances for a function version or alias. |
| Version | **This parameter is mandatory only when Type is set to Version.**<br>Select the version of the function for which you want to configure reserved instances. |
| Alias | **This parameter is mandatory only when Type is set to Aliases.**<br>Select the alias of the function for which you want to configure reserved instances. **Aliases of additional versions are supported.** |
| Reserved Instances | Number of reserved instances. The value is an integer ranging from 1 to 1000. After it is configured, FunctionGraph creates a fixed number of function instances for the function and keeps them running. |
| Idle Mode | This mode saves costs as CPU resources are not used when reserved instances are not invoked. |
| Auto Scaling Policies | If the number of reserved instances is fixed, you do not need to configure this parameter. |

5. Click **OK**.

## Configuring a Scheduled Scaling Policy

You can configure a scheduled policy for reserved instances. The number of reserved instances can be updated within a specified period. After this period, the number of reserved instances is restored to the **Reserved Instances** specified in the basic settings.

1. For details about how to configure the basic settings of a reserved instance policy, see **Configuring a Fixed Number of Reserved Instances**.
2. In the dialog box for adding reserved instance policies, click **Add Policy** and add an auto scaling policy by referring to **Table 7-29**.

**Table 7-29** Parameters for configuring a scheduled policy

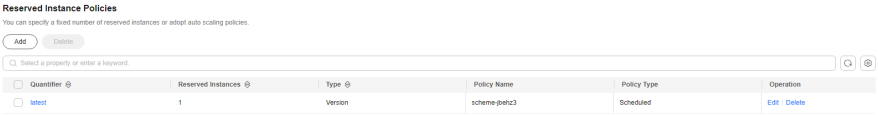
| Parameter | Description |
| --- | --- |
| Policy Type | Select **Scheduled**. |

| Parameter | Description |
| --- | --- |
| Policy Name | Name of a custom policy. The name can contain a maximum of 60 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. |
| Cron Expression (UTC) | Use a Cron expression to set the time when the number of reserved instances in the scheduled policy takes effect. For details about how to set a Cron expression, see **Cron Expression Rules**. |
| Validity | Time window when the scheduled policy takes effect. The scheduled policy takes effect only when the time is within the time window. If no scaling policy in the reserved instance policy takes effect, the number of reserved instances is restored to the value in the **Basic Settings**. |
| Reserved Instances | Number of reserved instances that need to be created when the scheduled policy takes effect. The value must be greater than or equal to the number of reserved instances configured in **Configuring a Fixed Number of Reserved Instances**. |

3. Click **OK**. The scheduled policy is added.

**Figure 7-42** Policy list




4. Click **OK**. The reserved instance policy is configured.
   - Multiple scheduled policies can be configured. For example, the number of reserved instances at 08:00 and 21:00 is updated to 100 and 10 respectively.
   - After the configuration is complete, you can click the name of a reserved instance policy in the **Reserved Instance Policies** area, and click the name of an auto scaling policy to view the number of instances concurrently executed by the function.

     NOTE

     Multiple scheduled policies can be configured. For example, the number of reserved instances at 08:00 and 21:00 is updated to 100 and 10 respectively.

## Configuring an Intelligent Recommendation Policy

Intelligent recommendation policies are based on feature profiling and load prediction technologies, dynamically adjusting reserved instances for peak and off-peak demands.

You can add an intelligent recommendation policy to reserved instances. FunctionGraph will update the number of reserved instances based on function load.

Intelligent recommendation policies are available in three options: high performance, balance, and low cost. The system dynamically adjusts the number of reserved instances based on load prediction to adapt to the peak and off-peak loads. The cost and performance of reserved instances are displayed. (Intelligent recommendation policies cannot coexist with other types of policies. A function version or alias can have only one such policy.)

1. For details about how to configure the basic settings of a reserved instance policy, see **Configuring a Fixed Number of Reserved Instances**.
2. In the dialog box for adding reserved instance policies, click **Add Policy** and add an auto scaling policy by referring to **Table 7-30**.

**Figure 7-43** Intelligent recommendation


**Table 7-30** Parameters for configuring an intelligent recommendation policy

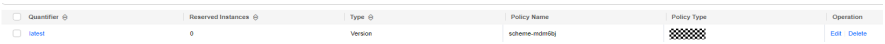
| Parameter | Description |
| --- | --- |
| Policy Type | Select **Intelligent recommendation**. |

| Parameter | Description |
| --- | --- |
| Policy Name | Name of a custom policy. The name can contain a maximum of 60 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. |
| Principle | Select the execution mode of the intelligent policy based on service requirements. The options are **High performance**, **Balance**, and **Low cost**.<br>• **High performance**: Average CPU usage of 40%; high performance during peak hours<br>• **Balance**: Average CPU usage of 40%; affordable cost during peak hours<br>• **Low cost**: Average CPU usage of 70% and low cost |

3. Click **OK**. The intelligent recommendation policy is added.

   **Figure 7-44** Reserved Instance Policies

4. Click **OK**. The reserved instance policy is configured.

   After the configuration is complete, you can click the name of a reserved instance policy in the **Reserved Instance Policies** area, and click the name of an auto scaling policy to view the cost and performance statistics of reserved instances.

## Helpful Links

- For details about function instance types and their usage modes, see **Function Instance Types and Usage Modes**.
- Manage reserved instances using APIs. For details, see **Reserved Instances APIs**.

# 7.20 Sharing Functions Based on RAM

Based on RAM, FunctionGraph allows you to share functions across accounts. After accepting a sharing invitation, principals can access and use the shared functions.

Function owners can specify sharing permissions based on the least privilege principle and usage requirements, so that principals can only access functions within permissions. This improves function security. For more information about the RAM service, see **What Is RAM?**.

If your account is managed by Huawei Cloud Organizations, you can share functions more easily. If your account is in an organization, you can share functions either with individual accounts or with all accounts in the organization or in an organization unit (OU) without having to enumerate each account. For details, see **Enabling Sharing with Organizations**.

## Notes and Constraints

- Only the owner can share functions with other accounts. Principals cannot share functions shared by other accounts with other accounts. Principals can only view and invoke shared functions.
- You need to enable **Sharing with Organizations** to share functions with your organizations or OU. For more information, see **Enabling Sharing with Organizations**.
- A principal can accept a maximum of 50 shared functions.
- To specify resource share details, set the **Resource Type** to **functiongraph:function**.
- After the function share is created, principals need to accept the share invitation within a specified period before using the functions. For details, see **Responding to a Resource Sharing Invitation**.
- Currently, FunctionGraph supports only sharing of functions.

## Creating a Function Share

The function owner needs to create a resource share on the RAM management console. For details, see **Creating a Resource Share**.

## Viewing and Using Shared Functions

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. On the **Shared** tab, you can view the function resources shared by other accounts.
   - If you are a function owner, you can find the corresponding share on the RAM management console based on the share name and view the resource status, permissions, and principals. For details, see **Viewing a Resource Share**.
   - If you are a function principal, you can find the corresponding share on the RAM management console based on the share name and view the resource status, permissions, and owners. For details, see **Viewing Resources Shared with You**.

   **Figure 7-45** Viewing shared functions

   
   
3. You can click the function name to view the function details and invoke the function.

## Stopping Function Sharing

- If a function share is no longer needed, the owner can delete it at any time. The shared functions will not be deleted. After a function share is deleted, the specified principals cannot use the functions in the share. For details, see **Deleting a Resource Share**.

- The owner can update a function share at any time, including updating its name, description, tags, shared functions, permissions, and principals. For details, see **Updating a Resource Share**.
- You can leave the function share if you do not need it. But the functions in the share will no longer be accessible then.

  You can leave a function share only if your account is an individual Huawei Cloud account instead of a part of an organization. For details, see **Leaving a Resource Share**.

# 8 Configuring Function Flows

## 8.1 Creating a Flow

Flows are distributed serverless applications. Visually orchestrate multiple independent serverless functions into an application in sequential, branch, or parallel mode using Low Code technology. In addition, diagnose and debug your applications through a monitoring and management platform.

This section describes how to create and orchestrate a flow. You can create a standard or express flow for your service requirements.

- Standard flow: better for common services that take a long time to execute. Standard flows can only be invoked asynchronously, and their execution records are persisted for query.
- Express flow: better for services that take less than 5 minutes to execute and require ultimate performance. Express flows can be invoked synchronously or asynchronously, but their execution records (such as execution node history) are not persisted. The synchronous execution interface returns results, and the execution logs reported to LTS are displayed on the logs page.

  NOTE

  Express flows are available for free trial as a time-limited offer.

### Notes and Constraints

Flows are available only in CN East-Shanghai1 and AP-Singapore.

### Creating a Flow

**Procedure**

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Flows**.
2. Click **Create** next to **Standard Flow** or **Express Flow**.
3. Orchestrate the flow by dragging components as in **Configuring Function a Flow Component**.

A flow must be a directed acyclic graph (DAG). It has a start node followed only by one node (except the error handling and stop nodes) and ends after another node. You can end a flow in the following ways:

- Do not connect any conditional, parallel, or start node to the last node of the flow.
- Add a stop node as the last node of the flow.

4. After configuring all nodes, click **Save** in the upper right corner, set parameters, and click **OK**.

**Table 8-1** Parameter description

| Parameter | Description |
| --- | --- |
| Name | Enter the name of the function flow. |
| Enterprise Project | Select an enterprise project. |
| Logs | Required when creating an express flow.<br>• **ALL**: All events will be recorded.<br>• **ERROR**: Only errors will be recorded.<br>• **NONE**: Log recording is disabled. |
| Combine Input | Whether to combine the previous node's output with the current node's input. |
| Streaming Response | Required when creating an express flow. If this option is enabled, the flow returns data in a stream. For details, see stream file processing.<br>**The code of function nodes in the flow will call the streaming data API.** |
| Description | Description about the flow. |

## Expression Description

The JSONPath expression used for **Retry Condition (JSONPath)** in the Error Handling component and for **Input Filter Expression** and **Output Filter Expression** in the Conditional Branch component follows the structure: [JSONPath] + [Logical operator] + [Comparison data], for example, $.age >= 20.

**Table 8-2** JSONPath description

| Operator | Supported or Not | Description |
| --- | --- | --- |
| $ | Yes | The root element to query. This starts all regular expressions. |
| @ | Yes | The current node being processed. |

| Operator | Supported or Not | Description |
| --- | --- | --- |
| . | Yes | Subnode. |
| [ (, )] | Yes | Array indexes. |
| [start:end] | Yes | Array slice operator. |
| [?()] | Yes | Filter expression, which must be evaluated to a Boolean value. |

**Example**

```
{
    "fruits": [ "apple", "orange", "pear" ],
    "vegetables": [{
    "veggieName": "potato",
    "veggieLike": true
     },
     {
         "veggieName": "broccoli",
         "veggieLike": false
     }]
}
```

- Simple value

  The **$.fruits** expression obtains all values under **fruits**.

  The result of **$.fruits** is **["apple","orange","pear"]**.

- Simple filtering

  Expression: **$.vegetables[?(@.veggieLike == true)].veggieName**

  Meaning: Obtains all values of the key **vegetables** and outputs the value of **veggieName** whose **veggieLike** is **True**.

  ```
  Result: [potato]
  ```

## Operators

The following data is used as input parameters in the examples:

```
{
  "name" : "apple",
  "weight": 13.4,
  "type": [3,4,6,8],
  "obj": {
        "a" : 1
   }
}
```

**Table 8-3** Logical operators

| Operator | Description | Example | Return Value | Remarks |
| --- | --- | --- | --- | --- |
| == | Equal to | $.name == 'apple' | true | Supported data types: int, float, string, bool, and nil |
| != | Not equal to | $.name != 'apple' | false | Supported data types: int, float, string, bool, and nil |
| < | Less than | $.weight < 12 | false | Only numbers supported |
| > | Greater than | $.weight > 12 | true | Only numbers supported |
| <= | Less than or equal to | $.weight <= 13.4 | true | Only numbers supported |
| >= | Greater than or equal to | $.weight >= 13.4 | true | Only numbers supported |
| '*' | Wildcard | $.weight == '*' | true | Must be used together with ==. |
| \|\| | Or | $.name == 'apple' \|\| $.weight < 12 | true | Used together with () for complex AND/OR logic. |
| && | And | $.name == 'apple' && $.weight < 12 | false | Used together with () for complex AND/OR logic. |

- A string constant must be enclosed with single quotation marks (''). For example, **'apple'**.
- JSONPath expressions cannot contain =, !=, <, >, |, or &.

# 8.2 Starting a Flow

After configuring a flow, start it to activate the service function.

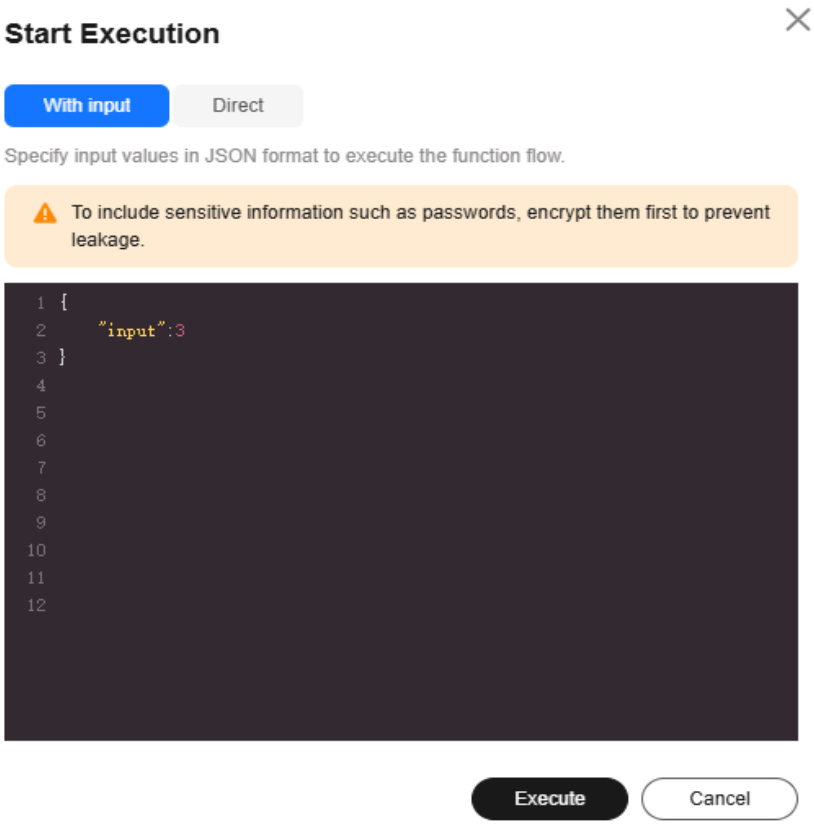
### Notes and Constraints

If you modify the flow configuration, save the changes before starting the flow.

### Starting a Flow

1. On the function flow orchestration page, click **Start** in the upper right corner. Alternatively, on the function flow list page, click **More** > **Start**.
2. On the **Start Execution** page, specify input or start the flow directly. In this example, select **With input**. (**The input values must be in JSON format.**)

   **Figure 8-1** Starting execution




3. Click **Execute**. A message indicating that the flow is started successfully is displayed in the upper right corner.

## 8.3 Configuring Function a Flow Component

## 8.3.1 Configuring a Function Component

Function flows support the function component. You can associate a created function with the component to meet service requirements.

## Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- **The data returned by the configured function must be in JSON format. Otherwise, it cannot be parsed.**
- Synchronous invocation does not support long-term running functions and is max. 15 minutes.
- Asynchronous invocation supports long-term running functions. The max. duration of a function node is the same as that supported by FunctionGraph.
- A flow can have zero to 99 functions.
- A function connected to an error handling node can be connected to another node that is not start or error handling.
- A function that is not connected to an error handling node can be connected to only one non-start node.

## Prerequisites

- You have created a function on the FunctionGraph console. For details, see **Creating a Function from Scratch**.
- Before using flows, learn about **Expression Description** and **Operators**.

## Configuration Description

Click the function node to edit it. Set the function parameters. For details, see **Table 8-4**.

**Figure 8-2** Configuring a function node

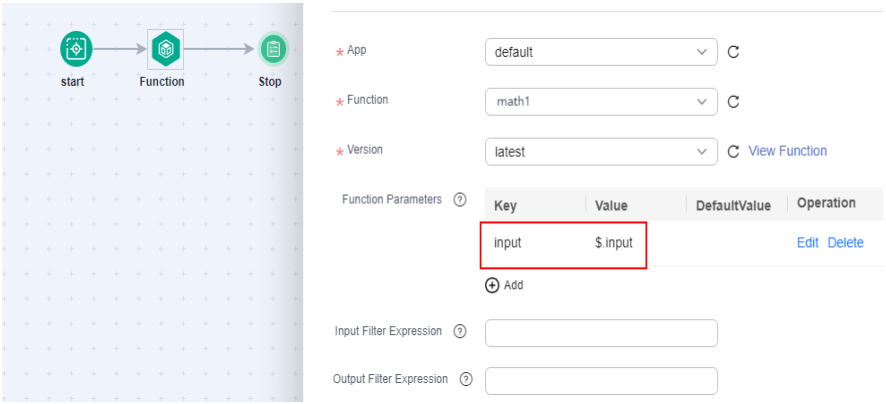



**Table 8-4** Function parameters

| Parameter | Description |
| --- | --- |
| App | App to which the desired function belongs. You can create multiple functions under an app. |
| Function | Function you want to use. |
| Version | Function version you want to use. |

| Parameter | Description |
| --- | --- |
| CallMode | Required when creating a standard flow. By default, function nodes in a flow are invoked synchronously. |
| Function Parameters | Key-value pairs passed for executing a function. Set constants, or obtain values from the input defined during function flow startup or the output of the former node using a JSONPath expression. To include sensitive information such as passwords, encrypt them first to prevent leakage.<br>Parameters that will be passed in JSON format in the body.<br>**Key**: parameter name<br>**Value**: parameter value<br>**DefaultValue**: used if no value is specified for the parameter<br>**Operation**: to edit or delete the parameter |
| Input Filter Expression | JSONPath expression used to filter the input information of the node. |
| Output Filter Expression | JSONPath expression used to filter the output information of the node. |
| Turn on StandbyOperation | When enabled, the current node name cannot be the same as other function node names. |

### Example

This section uses Python 3.9 to demonstrate function implementation. The code and function are shown below. For detailed instructions on creating functions, see **Creating a Function from Scratch**.

Function: Returning the value of the event input

```
import json
def handler (event, context):
    input = event.get('input',0)
    return {
        "result": input
        }
```

## 8.3.2 Configuring a Subflow Processor

Multiple function flows can be combined as a new flow and reused to avoid repeated orchestration.

### Notes and Constraints

Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.

### Prerequisites

- You have created a function on the FunctionGraph console as a subflow.
- Before using flows, learn about **Expression Description** and **Operators**.

### Configuration Description

On the function orchestration page, click the subflow node and set the parameters by referring to **Table 8-5**.

**Table 8-5** Subflow parameters

| Parameter | Description |
| --- | --- |
| Subflow | All created flows are displayed in the drop-down list. |
| Proceed Until Subflow Completed | Enabled by default. |
| Input Filter Expression | JSONPath expression used to filter the input information of the node. |
| Output Filter Expression | JSONPath expression used to filter the output information of the node. |

## 8.3.3 Configuring a Parallel Branch Processor

The parallel branch processor enables multiple branches of a flow to execute simultaneously. You can determine the next step of each branch.

### Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- A parallel branch can connect between 1 and 20 nodes, excluding error handling, start, or stop nodes.

### Prerequisites

Before using flows, learn about **Expression Description** and **Operators**.

### Configuration Description

On the function orchestration page, click and configure the Parallel Branch node.

**Figure 8-3** Configuring a parallel branch node

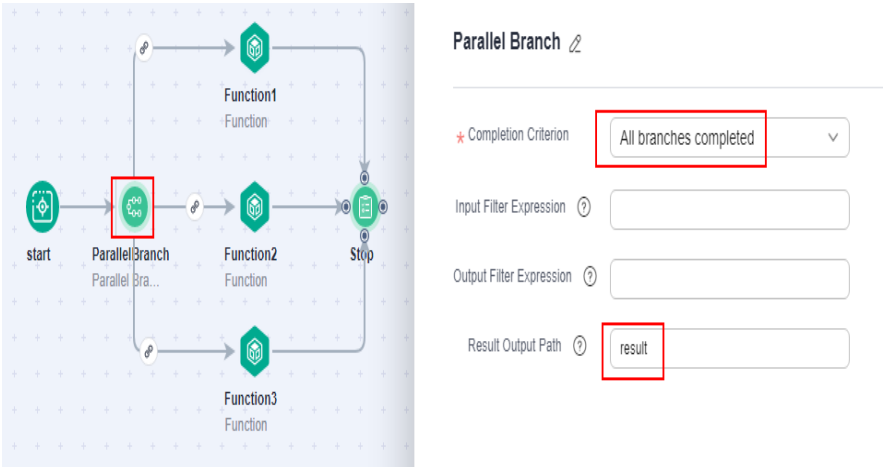



**Table 8-6** Parallel Branch parameters

| Parameter | Description |
| --- | --- |
| Completion Criterion | You can select any of the following options:<br>• All branches completed<br>• 1 branch completed<br>• Set number of branches completed<br>**Set Branches** is required for the third option. Specify a maximum of 20 branches. |
| Input Filter Expression | JSONPath expression used to filter the input information of the node. |
| Output Filter Expression | JSONPath expression used to filter the output information of the node. |
| Result Output Path | Path to which the result of this parallel branch will be output in the JSON format. The path is a key and the result is a value. If this parameter is not set, the default path is **result**. |

## 8.3.4 Configuring a Start Processor

The start node serves as the the start of a function flow.

### Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- Each function flow must start with only one Start node and end with an End node.
- The start node must be followed by a node that is not stop or error handling.
- Only triggers can be configured for the Start node.

- Flow triggers include **APIG (dedicated)** and **timer** triggers.
- A flow can have a maximum of 10 triggers.
- Triggers must be included in the start node.
- Triggers cannot be connected to any other node.

## Timer Trigger Configuration Description

Click the **Start** node. On the displayed page, click **Create Trigger**, set **Trigger Type** to **Timer**, and set other parameters, as shown in **Table 8-7**.

**Figure 8-4** Timer trigger

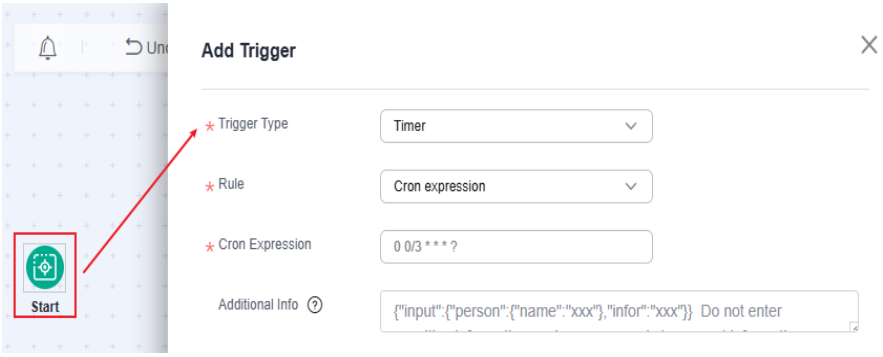



**Table 8-7** Timer trigger information

| Parameter | Description |
| --- | --- |
| Trigger Type | Select **Timer**. |
| Rule | Triggering rule of the timer. Currently, only cron expressions are supported. |
| Cron Expression | Specifies the date and time when the flow will be scheduled. For details, see **Cron Expression Rules**. |
| Additional Info | Must be in JSON format and contain an input. The value of input will be passed to the flow. |

## APIG (Dedicated) Trigger Configuration Description

- APIG flow triggers support only IAM authentication.
- Ensure that you have created a dedicated gateway before this operation. For details, see **Buying a Dedicated Gateway**.

Click the **Start** node. On the displayed page, click **Create Trigger**, set **Trigger Type** to **API Gateway (dedicated)**, and set other parameters, as shown in **Table 8-8**.

**Figure 8-5** API Gateway (dedicated) trigger

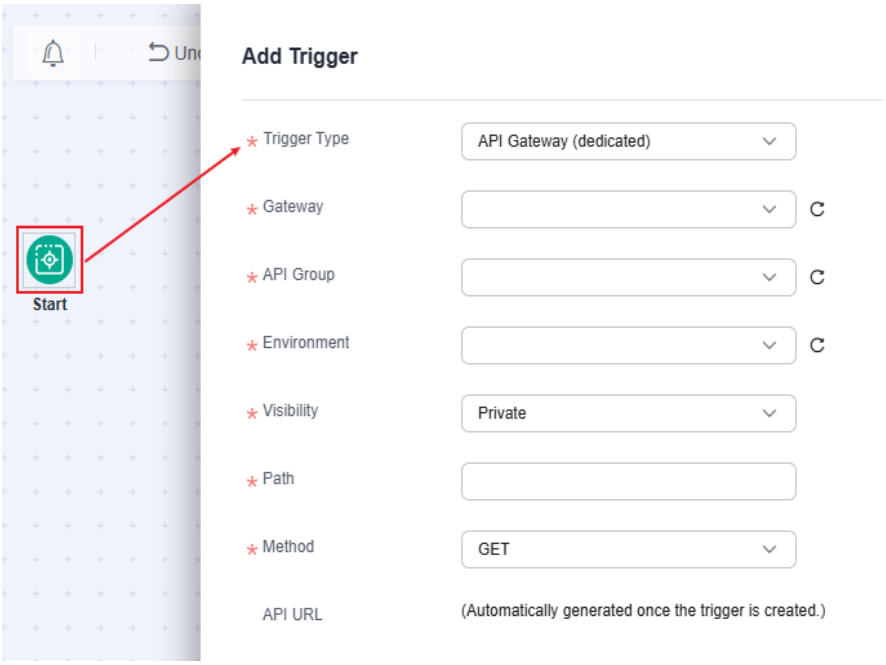



**Table 8-8** APIG (dedicated) trigger information

| Parameter | Description |
| --- | --- |
| Trigger Type | Select **API Gateway (dedicated)**. |
| Gateway | Select a dedicated gateway. For details, see **Creating a Gateway**. |
| API Group | An API group is a collection of APIs. You can manage APIs by API group. For details, see **Creating an API Group**. |
| Environment | An API can be called in different environments, such as production, test, and development environments. APIG supports environment management, which allows you to define different request paths for an API in different environments. To ensure that the API can be called, select **RELEASE**. |
| Visibility | Options:<br>• **Public**: The API can be released to KooGallery.<br>• **Private**: The API will be excluded when the API group to which it belongs is released on KooGallery. |
| Path | Path for requesting the API.<br>Format: **/users/projects** |
| Method | The API calling method. Options: **GET**, **POST**, **DELETE**, **PUT**, **PATCH**, **HEAD**, **OPTIONS**, and **ANY**.<br>**ANY** indicates that the API can be called using any request method. |

# 8.3.5 Configuring an Error Handling Processor

The Error Handling processor component controls the next step after a function fails to be executed. An error handling node determines whether to retry when the function it follows fails. There are two types of function failures: execution error and internal service error with an error code (**200**: success; **500** and **404**: failure).

## Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- An error handling processor can be connected to a maximum of 10 nodes that are not start, stop, or error handling.

## Configuration Description

On the function orchestration page, click the error handling node. On the page displayed on the right, configure the parameters.

**Figure 8-6** Configuring error handling

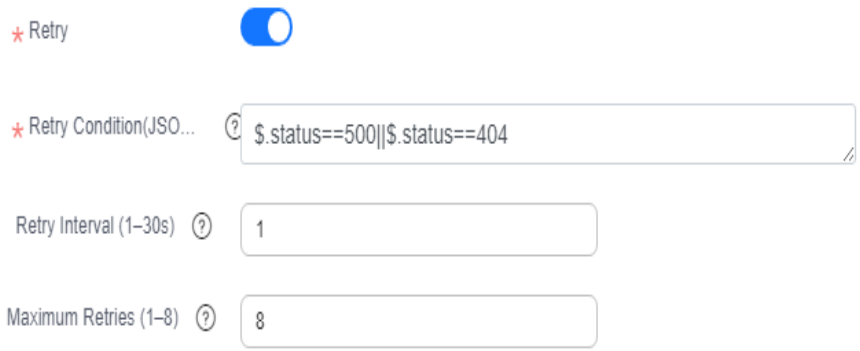



**Table 8-9** Error handling parameters

| Parameter | Description |
| --- | --- |
| Retry | Disabled by default. |
| Retry Condition (JSONPath) | Required when **Retry** is enabled. This parameter specifies whether to retry the task at the configured retry interval and up to the maximum retry count when the retry condition is met. If the condition still holds after reaching the maximum retries, the workflow proceeds to the next node. If the retry condition is not met during or after the maximum retries, an alternate branch node is executed. |

| Parameter | Description |
| --- | --- |
| Retry Interval (1–30s) | Required when **Retry** is enabled. The default retry interval is 1s. |
| Maximum Retries (1–8) | Required when **Retry** is enabled. The default retry count is 3. |

## 8.3.6 Configuring a Loop Processor

The Loop processor iterates over each element in the input array, executing a subprocess or function once per loop.

### Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- The subflows in a loop must satisfy the following rules:
  - Start with a function or wait node.
  - Only include function, wait, and error handling nodes.

### Prerequisites

- You have created a function on the FunctionGraph console. For details, see **Creating a Function from Scratch**.
- Before using flows, learn about **Expression Description** and **Operators**.

### Configuration Description

In the flow orchestration area, click the loop component and configure it.

**Figure 8-7** Configuring the loop component

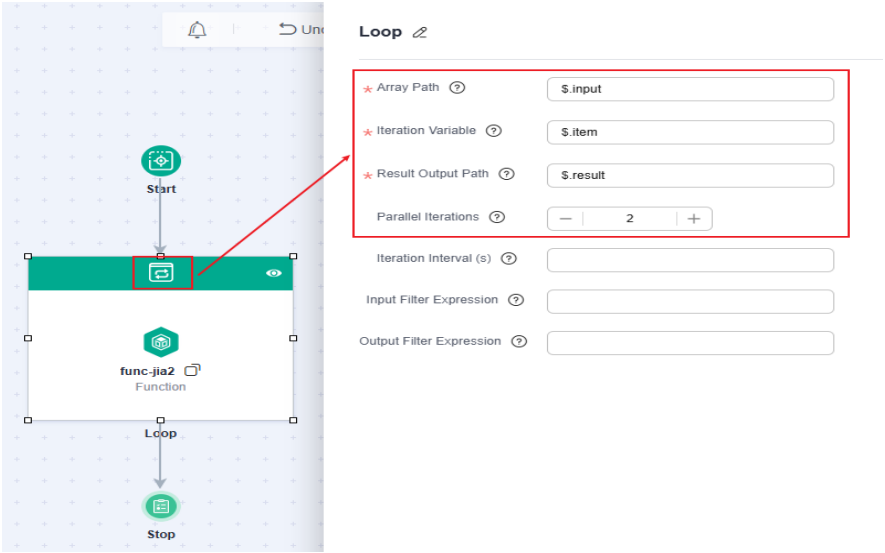


**Table 8-10** Loop component parameters

| Parameter | Description |
| --- | --- |
| Array Path | JSONPath expression used to obtain an array from the input. For an input of {"arr": [1,2,3]}, set **Array Path** to **$.arr** to obtain the array. |
| Iteration Variable | Temporary variable for each element in the array during the traversal of the array. For example, if **item** is used as the temporary variable, set this parameter to **$.item**. |
| Result Output Path | JSONPath expression used to output results in JSON format. For a Result Output Path of $.result, the output result will be {"result": [2,3,4]}. |
| Parallel Iterations | Range: 0 to 100, where 0 means no limit on the number of parallel iterations. |
| Iteration Interval (s) | Interval between parallel iterations. |
| Input Filter Expression | JSONPath expression used to filter the input information of the node. |
| Output Filter Expression | JSONPath expression used to filter the output information of the node. |

## 8.3.7 Configuring a Wait Processor

The Wait processor controls the current subflow or function to call the next subflow or function after a specified delay.

### Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- The Wait processor can be connected to zero or one node that is not start or error handling.

### Prerequisites

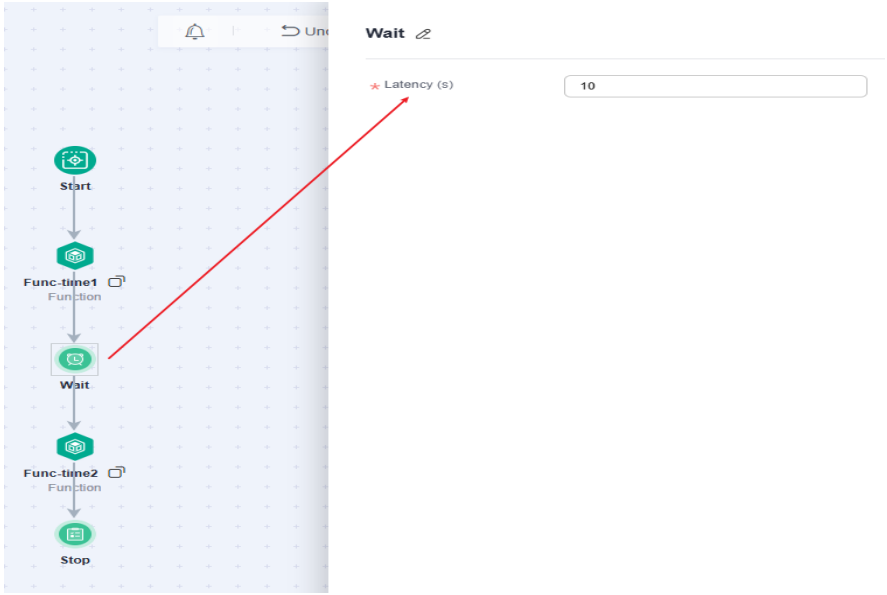
Before using flows, learn about **Expression Description** and **Operators**.

### Configuration Description

In the flow orchestration area, click the Wait processor and change the value of **Latency (s)** (the value must be between **0.001** and **86400**).

**Figure 8-8** Configuring the Wait processor




## 8.3.8 Configuring a Service Processor

The Service processor combines function operations into an atomic node for easy management.

It supports both serial and parallel execution modes.

### Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- The Service processor consists of multiple functions and can be connected to a stop node or an error handling node.

### Prerequisites

Before using flows, learn about **Expression Description** and **Operators**.

### Configuration Description

On the function orchestration page, click and configure the service node.

**Figure 8-9** Serial

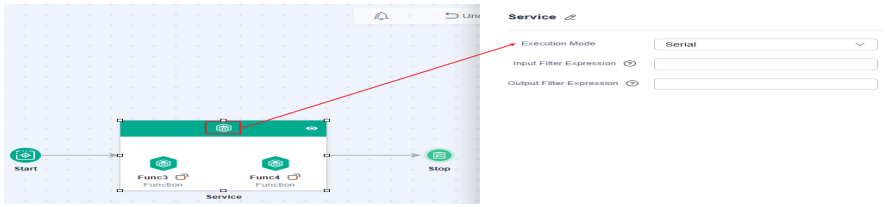


**Figure 8-10** Parallel

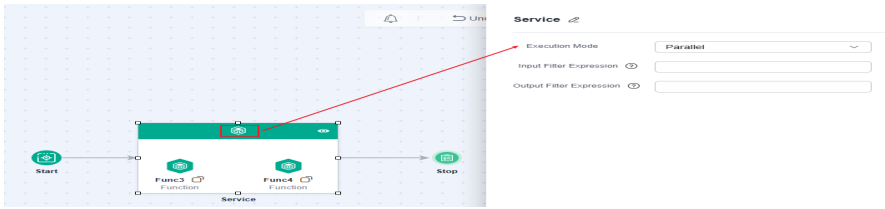



**Table 8-11** Service component parameters

| Parameter | Description |
| --- | --- |
| Execution Mode | Select **Serial** or **Parallel** from the drop-down list.<br>• **Serial**: Functions in the Service component must be connected.<br>• **Parallel**: Functions in the Service component do not need to be connected. |
| Input Filter Expression | JSONPath expression used to filter the input information of the node. |
| Output Filter Expression | JSONPath expression used to filter the output information of the node. |

## 8.3.9 Configuring a Conditional Branch Processor

The Conditional Branch processor of a function flow determines whether to execute other branches based on the configured conditions.

### Notes and Constraints

- Function flows created in Data Workroom (DWR) can only be viewed on the FunctionGraph console; editing and deletion must be done within DWR.
- The Conditional Branch processor connects 2 to 20 nodes that are not start, stop, or error handling.

### Prerequisites

Before using flows, learn about **Expression Description** and **Operators**.

### Configuration Description

In the flow orchestration area, configure the **Conditional Branch** and **Edge** parameters.

**Figure 8-11** Configuring the Conditional Branch processor

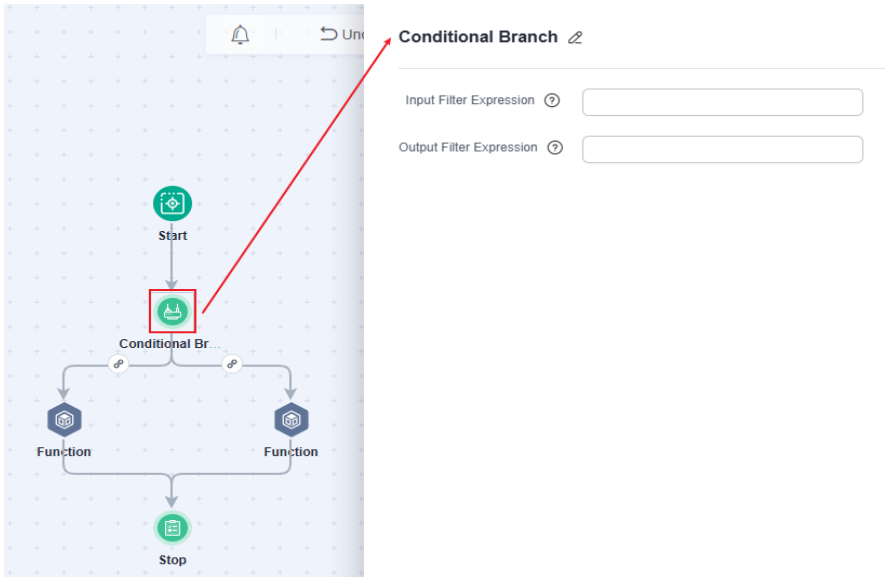



**Figure 8-12** Configuring the **Edge**

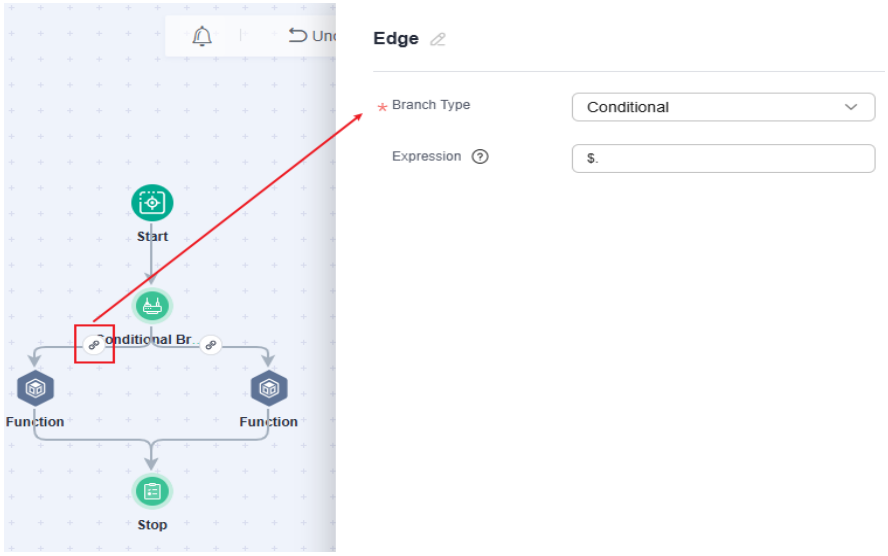



**Table 8-12** Conditional Branch processor parameters

| Parameter | Description |
| --- | --- |
| Input Filter Expression | JSONPath expression used to filter the input information of the node. |
| Output Filter Expression | JSONPath expression used to filter the output information of the node. |

**Table 8-13** Edge parameters

| Parameter | Description |
| --- | --- |
| Branch Type | Select **Conditional** or **Default** from the drop-down list. |
| Expression | Required when **Branch Type** is set to **Conditional**. This parameter determines whether the current branch meets the execution condition. For example, **$.input<2** means the conditional branch runs when the input is less than 2, otherwise the default branch runs. |

# 8.4 Managing Function Flows

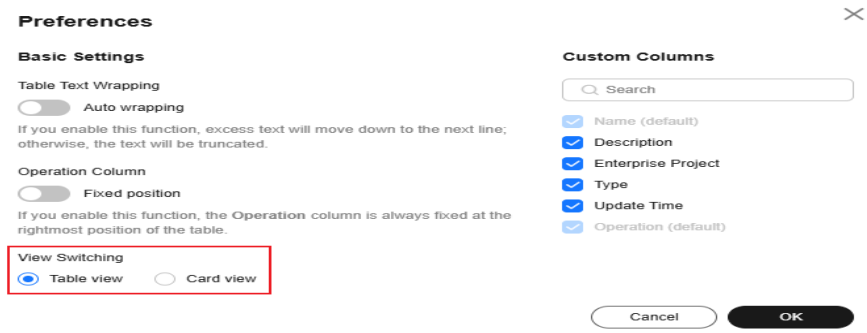
After a function flow is created, you can check its configuration, execution history, and execution logs, retry a failed flow, or terminate an ongoing flow.

## Viewing Flows

**Step 1** Log in to the **FunctionGraph console**. In the navigation pane, choose **Flows**.

**Step 2** View flows in card or list mode.

**Figure 8-13** Selecting a display mode



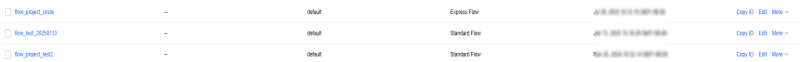

- Card mode

  **Figure 8-14** Card mode

- List mode

  **Figure 8-15** List mode

**Step 3** View all flows that you have created, and perform the operations described in **Table 8-14** as required.

**Table 8-14** Operation description

| Operation | Description |
| --- | --- |
| Edit | Click **Edit** to modify a flow. |
| Start | Choose **More** > **Start** to execute a flow. |
| Delete | • Deleting a single flow: Choose **More** > **Delete** to delete a flow.<br>• Deleting multiple flows: Select multiple flows and click **Delete** in the upper left corner to delete them. |

**Step 4** Click the name of a flow to view details.

- Viewing basic information

  On the **Basic Information** tab page, view the flow name, ID, update time, and creation time.

- Viewing task running records **(only for standard function flows)**

  On the **Metric** tab page, view the execution history, input, output, and node logs.

  To modify the flow information, click **Design** in the upper right corner.

- Viewing task logs **(only for express function flows)**

  On the **Logs** tab page, click the request ID to view the execution result logs. You can download a maximum of 5,000 logs at a time.

- View flow metrics

  On the **Monitoring** tab page, view the counts of invocations, errors, running flows, and throttles, and the running duration. **Table 8-15** describes the monitoring metrics.

  **Table 8-15** Flow metrics

| Metric | Unit | Description |
| --- | --- | --- |
| Invocations | Count | Total number of invocation requests, including successful, failed, and number of executions that are in progress. In case of asynchronous invocation, the count starts only when a flow executes in response to a request. |
| Duration | ms | Average time taken to execute a flow in a specified period. |
| Errors | Count | Number of times that a flow failed to be executed. |
| Running Workflows | Count | Number of invocation requests that are being processed. |

| Metric | Unit | Description |
| --- | --- | --- |
| Throttles | Count | Number of times that a flow failed to be executed due to rate limiting. |

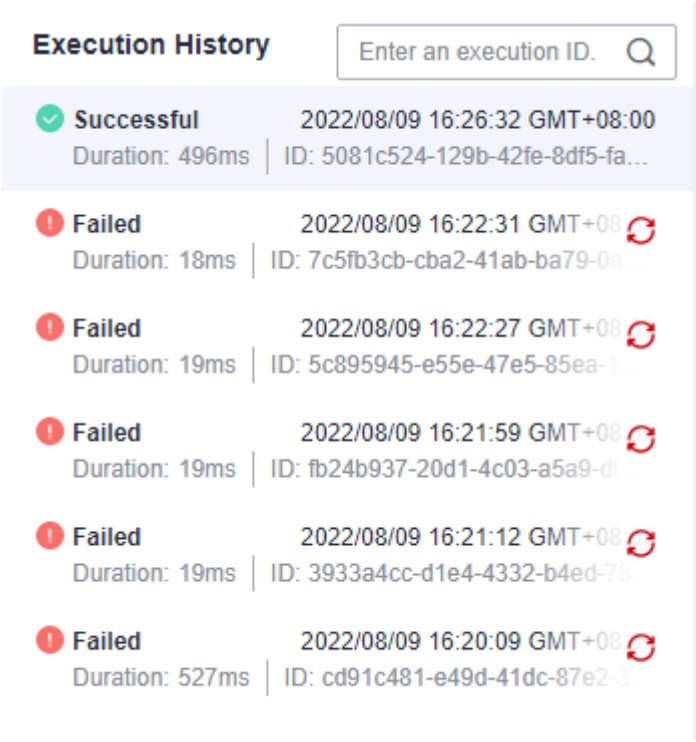
**----End**

## Viewing the Execution History of a Standard Flow

**Step 1** On the function flow list page, click the name of a standard flow and click the **Executions** tab.

**Step 2** The left pane lists the execution history, showing the latest 100 records.

**Figure 8-16** Execution logs



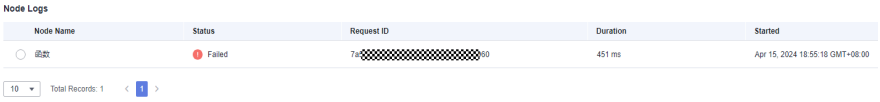

- Click an execution. The execution result is displayed on the canvas in the middle. Nodes in green are successful, and nodes in red have failed.
- Below the canvas is the input and output values of this flow. Click a node on the canvas to check the values. (Output fields with value **null** will not be displayed.)

**Figure 8-17** Input and output display area

- Click on the right of a failed execution. If the retry is successful, a new record is generated.
- Click on the right of an ongoing execution, it goes into the canceled state after the execution is stopped.
- The **Node Logs** area displays the execution records of all nodes from the start to the end of the flow.

**Figure 8-18** Node logs




**----End**

## Viewing Execution Logs of an Express Flow

**Step 1** On the function flow list page, click the name of an express flow and click the **Logs** tab.

**Step 2** Click the request ID to check the execution log details. You can filter logs from the last hour, day, three days, or a custom time range.

**----End**

# 9 Deploying the Function Application Using a Template

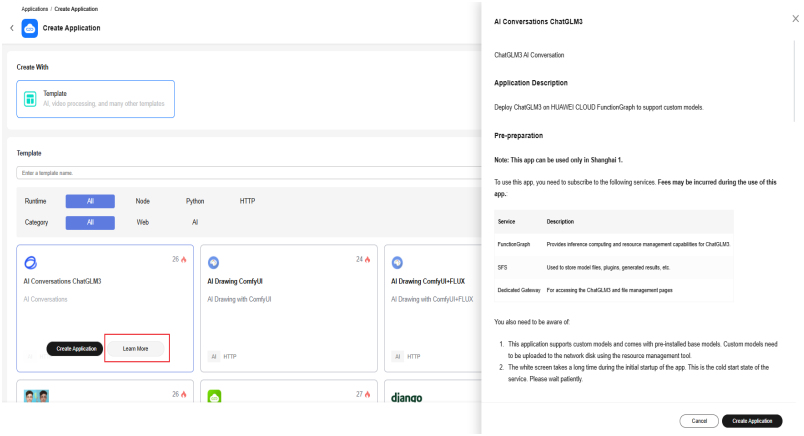
Application Center uses Resource Formation Service (RFS) to deploy peripheral resources (including functions, agencies, and triggers) required by applications so that these resources can cooperate with each other to execute tasks

## Overview

FunctionGraph provides function templates for different scenarios. When you create a function using a template, the template automatically configures the function code, environment variables, and triggers.

You can filter function templates by runtime and scenario. Click **Learn More** of a function template to view the usage description and precautions of the template as shown in **Figure 9-1**.

**Figure 9-1** Template details




## Billing

You are not charged for creating applications. However, other cloud service resources may be created during application creation. For details about the billing of other cloud services, see the billing description of each cloud service.

## Notes and Constraints

Currently, the application center is supported only in the **CN North-Beijing4**, **CN East-Shanghai1** regions.

## Prerequisites

If FunctionGraph needs to access other cloud services such as LTS and VPC, you need to create an agency and assign permissions by referring to **Creating a Function Agency**

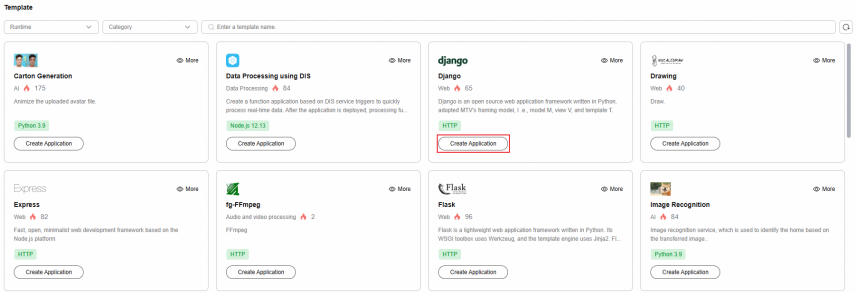
## Creating an Application

**Step 1** Log in to the **FunctionGraph console**. In the navigation pane, choose **Apps**.

**Step 2** Click **Create** in the upper right corner. The **Templates** page is displayed.

**Step 3** The **Django** template is used as an example. After selecting the template, click **Create Application**, as shown in **Figure 9-2**. The application configuration page is displayed.

**Figure 9-2** Selecting a template




**Step 4** On the **Create Application** page, set basic information by referring to **Table 9-1** and click **Create Now**.

**Table 9-1** Parameters in the Django template

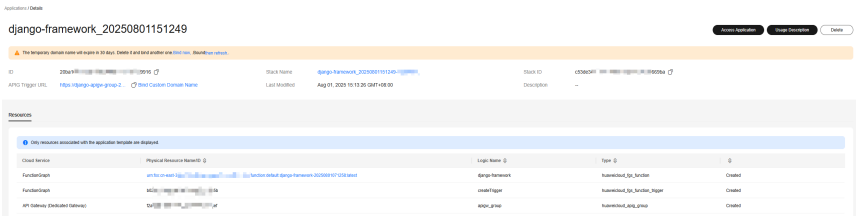
| Parameter | Description | Example Value |
| --- | --- | --- |
| Template | The selected function template is displayed by default.<br>To change it, click **Reselect**. | Django |
| Region | Region where the function application is created.<br>By default, the current region is selected. To change it, return to the FunctionGraph console and switch the region in the upper left corner. | CN East-Shanghai1 |

| Parameter | Description | Example Value |
| --- | --- | --- |
| Application Name | Enter a function application name.<br>The value can contain 2 to 60 characters, starting with a letter and ending with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. | django-framework |
| Runtime | The runtime language of the function application template, which cannot be changed. | http |
| Agency | Select an agency for FunctionGraph.<br>If **Use no agency** is selected, the system will use agency **fgs-app-adminagency**, or create it if not available. | fgs-app-adminagency |
| Create Repository | When enabled, you can create a code repository in CodeArts Repo to update and deploy your application code. | Enabled |
| Project | The CodeArts Req project where a code repository will be created. If no project is specified, a project with the same name will be created in CodeArts Repo, and a code repository will be created in this project. | - |
| Description | Enter a description of the application. The description can contain a maximum of 1,024 characters. | - |
| APIG Instance | Select an available APIG gateway. If no gateway is available, click **Create Instance** on the right to go to the creation page. | apig-fg |

**Step 5** After the application is created, the required function resources and gateway resources are automatically created, as shown in **Figure 9-3**. You can click the link under **Physical Resource Name/ID** to go to the resource details page.

**Figure 9-3** Application created successfully




**----End**

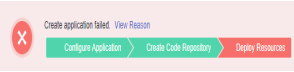
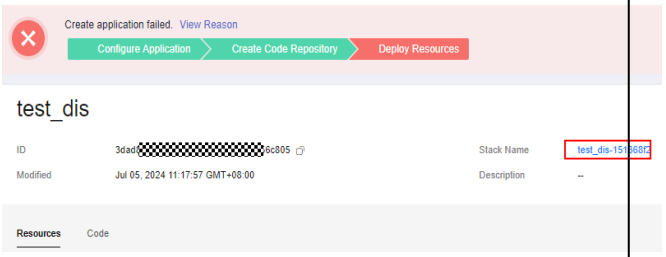
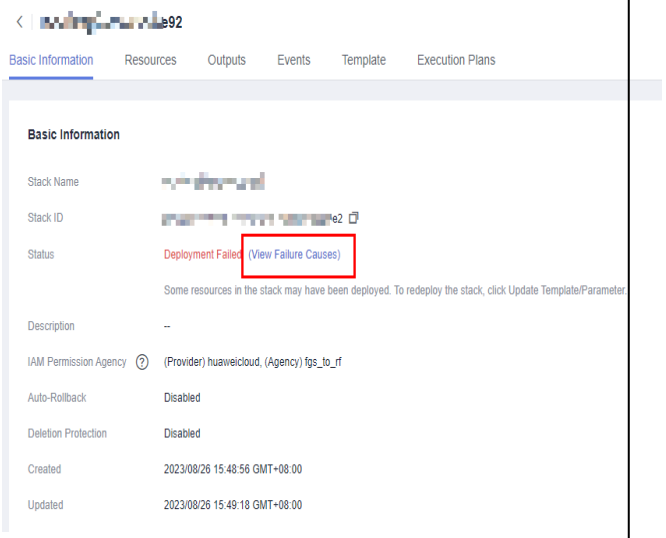
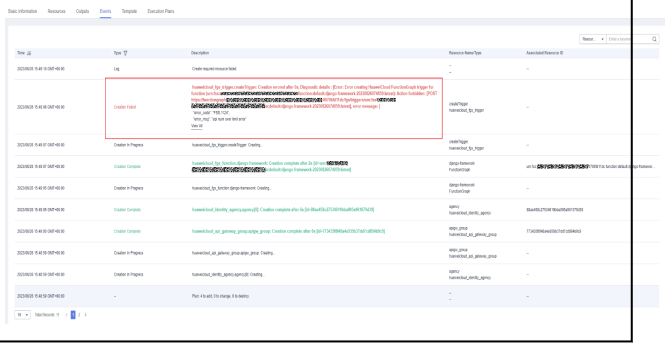
  

## Common Problems and Troubleshooting Methods

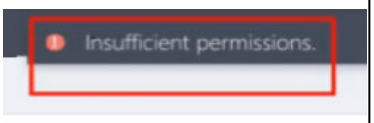
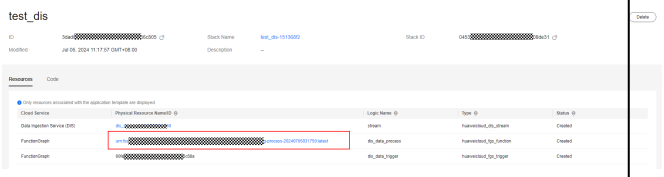
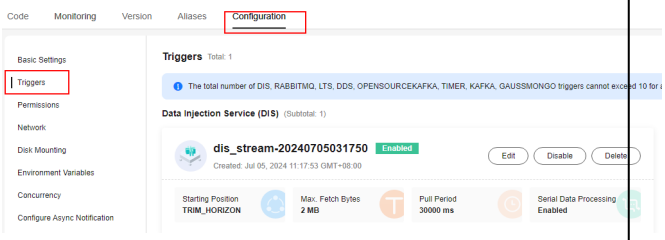
**Table 9-2** Troubleshooting methods for common problems

| Issue | Troubleshooting Method |
| --- | --- |
| The code repository fails to be created, and the message shown in **Figure 9-4** is displayed.<br><br>**Figure 9-4** Code repository creation failed<br>  | Check whether CodeArts is enabled for your account. For details, see **Login Methods** to check and enable CodeArts.<br>If the problem persists, **submit a service ticket** to contact Huawei Cloud FunctionGraph engineers for further help. |

| Issue | Troubleshooting Method |
| --- | --- |
| The stack fails to be deployed, and the message shown in **Figure 9-5** is displayed.<br><br>**Figure 9-5** Stack deployment failed<br>  | 1. Click the application name to go to its details page. Click the resource stack name to go to the RFS console.<br><br>**Figure 9-6** Application details page<br>  <br>2. On the RFS page, click **View Failure Causes** to view the detailed issue.<br><br>**Figure 9-7** RFS page<br>  <br>**Figure 9-8** Viewing failure causes<br>  |

| Issue | Troubleshooting Method |
| --- | --- |
| If the permission is insufficient, the message shown in **Figure 9-9** is displayed.<br><br>**Figure 9-9** Insufficient permissions<br>  | If an error message indicating insufficient permissions is displayed when you create an application for the first time, configure the agency permission for the current account by referring to **Configuring Agency Permissions** and try again. |
| If the deletion fails, the message shown in **Figure 9-10** is displayed.<br><br>**Figure 9-10** Deletion failed<br>  | 1. Click the application name to go to the details page. Click the resource stack name to go to the RFS console.<br>2. Click **View Failure Causes** to view the specific problem. Take **Figure 9-10** as an example, the deletion fails because the API group contains APIs.<br>3. Go to the details page of the application, and click the **Physical Resource Name/ID** link of FunctionGraph to access the function details page.<br><br>**Figure 9-11** Clicking the FunctionGraph link<br>  <br>4. Choose **Configuration** > **Triggers** to view the API name.<br><br>**Figure 9-12** Viewing the API name<br>  <br>5. Log in to the APIG console, take the API offline, delete the API, and try again. |

# 10 Managing Functions Using KooCLI

FunctionGraph provides the command line interface (CLI) for you to manage functions, triggers, and aliases, and invoke functions.

## Notes and Constraints

- When you access Huawei Cloud through an API, you must undergo access key–based identity authentication with the access request encrypted, and ensure that the access request is secret, complete, and correct. Properly keep the **config.yaml** file to prevent unauthorized use of your access key. **Use temporary access keys (Access Key ID, Secret Access Key, and Token) to enhance security. For details about the temporary access keys, see Temporary Access Key (for Federated Users).**
- If your access key is used by an unauthorized person (due to a loss or leakage), you can delete the access key or notify the administrator of resetting it and configure it again.
- Deleted access keys cannot be recovered.

## KooCLI Download Links

KooCLI can run on a 64-bit Linux x86 operating system (OS), 64-bit Windows OS, or macOS. **Table 10-1** provides the download links of KooCLI.

**Table 10-1** Download links of CLI

| OS | Software Package and Verification File | Reference |
| --- | --- | --- |
| Linux | KooCLI and Verification File | KooCLI Overview |
| Windows | | |
| Mac | | |

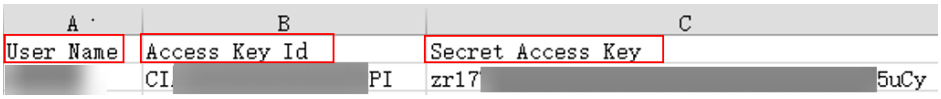
## Installing KooCLI

1. Install KooCLI. For details, see **Installing KooCLI in Linux**.

2. Obtain an access key (access key ID and secret access key, also called "AK/SK").
   - If you have access to the console, log in to it, and create an access key on the **My Credentials** page. For details, see **Creating an Access Key**. An AK/SK file is downloaded. Generally, it is named **credentials.csv**. As shown in the following figure, the file contains a username, AK, and SK.

     **Figure 10-1** Content of the credentials.csv file

   - If you do not have access to the console, request the administrator to create an access key for you on the IAM console in case your access key is lost or needs to be reset. For details, see **Managing Access Keys for an IAM User**.
3. Obtain a region name. For details, see **Regions and Endpoints**.

   **Figure 10-2** Obtaining region information

   | Region Name | Region |
   | --- | --- |
   | AF-Johannesburg | af-south-1 |
   | AP-Bangkok | ap-southeast-2 |
   | AP-Singapore | ap-southeast-3 |

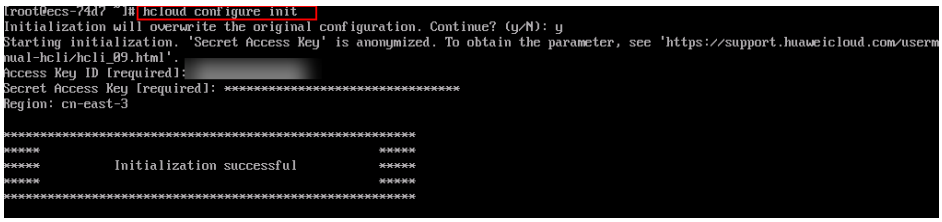
4. Initialize KooCLI.

   Run the following command to initialize KooCLI:

   ```
   hcloud configure init
   ```

   Enter an access key ID, secret access key, and region name. If the information shown in **Figure 10-3** is displayed, the initialization is successful.

   **Figure 10-3** Initialization successful

   You can also configure authentication information using the temporary AK, SK, and token.

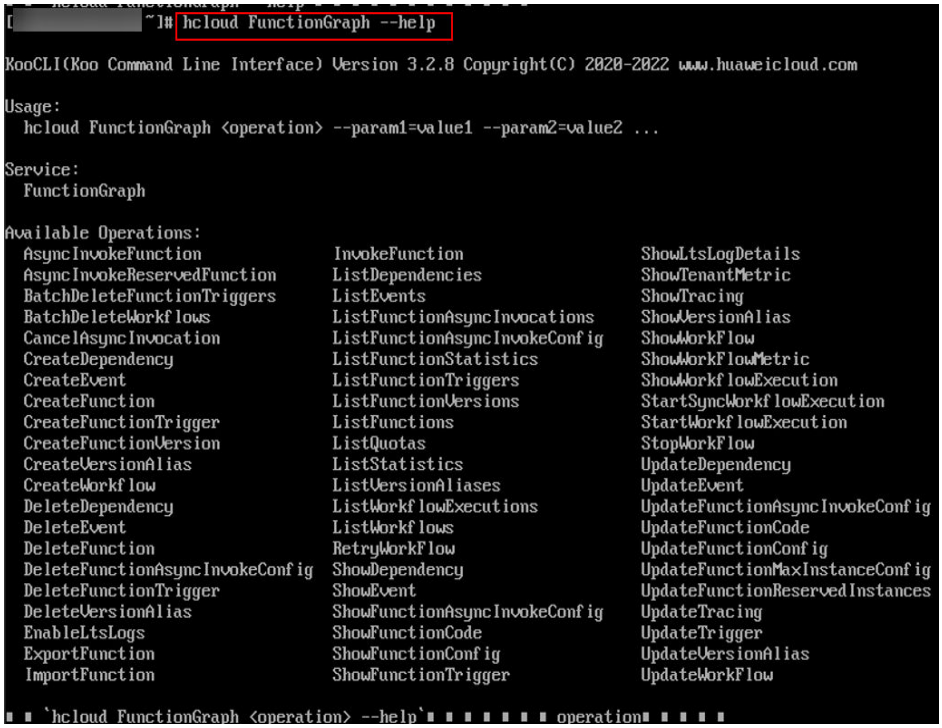
   ```
   hcloud configure set --cli-profile=default --cli-access-key=your-ak --cli-secret-key=your-sk --cli-security-token=your-token
   ```

5. Run the following command to view the commands supported by FunctionGraph. As shown in **Figure 10-4**, **Available Operations** lists the operations supported by FunctionGraph.

   ```
   hcloud FunctionGraph --help
   ```

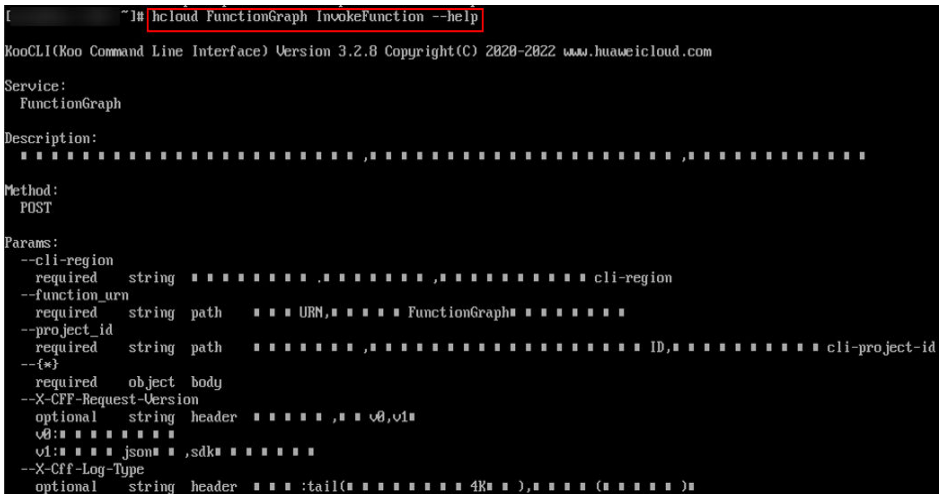
**Figure 10-4** Operations supported by FunctionGraph




Run the following command to obtain help information about operation **InvokeFunction**. If the command is successfully executed, the information shown in **Figure 10-5** is displayed.

```
hcloud FunctionGraph InvokeFunction --help
```

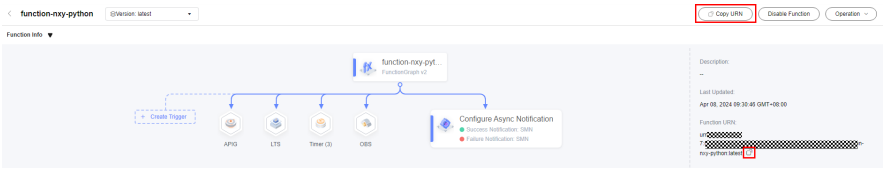
**Figure 10-5** Help information about operation InvokeFunction




## Invoking a Function

Before invoking a function, obtain the URN, as shown in **Figure 10-6**.

**Figure 10-6** Obtaining a function URN



- **Synchronous invocation**

  The following is an example command for synchronous invocation. For details about the parameters, see **Table 10-2**.

  ```
  hcloud FunctionGraph InvokeFunction --cli-region="ap-southeast-1" --X-Cff-Log-Type="tail" --X-CFF-Request-Version="v1" --function_urn="urn:fss:cn-east-3:******:function:default:hcloud-invoke:latest" --project_id="******" --key="value"
  ```

  **Table 10-2** Parameters

| Parameter | Mandatory | Description |
| --- | --- | --- |
| --cli-region | Yes | Region where the target function is located. |
| --function_urn | Yes | Function URN. |
| --project_id | Yes | Project ID. |
| --X-Cff-Log-Type | No | Options: **tail** (4 KB logs will be returned in the header) and **null** (no logs will be returned). |
| X-CFF-Request-Version | No | Response body format. Options:<br>• **v0**: text format.<br>• **v1**: JSON format. Use this format when using an SDK. |
| Body | Yes | Request body in **--key="value"** format. The JSON structure is **{"key":"value"}**. |

**Figure 10-7** shows the output result. For details about the response parameters, see **Table 10-3**.

**Figure 10-7** Output result

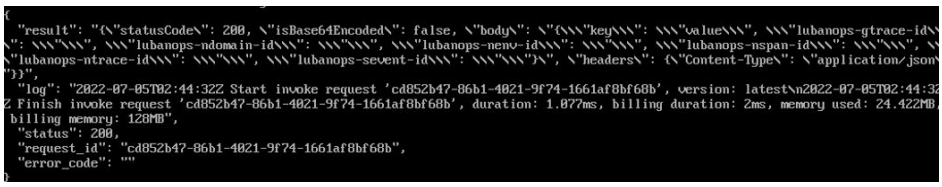


**Table 10-3** Response parameter

| Parameter | Type | Description |
| --- | --- | --- |
| request_id | String | Request ID. |
| result | String | Function execution result. |
| log | String | Function execution log. |
| status | Integer | Execution status. |
| error_code | String | Error code. |

- **Asynchronous invocation**

  The following is an example command for asynchronous invocation. For details about the parameters, see **Table 10-4**.

```
hcloud FunctionGraph AsyncInvokeFunction --cli-region="cn-east-3" --function_urn="urn:fss:cn-east-3:******:function:default:hcloud-invoke:latest" --project_id="******" --key="value"
```

**Table 10-4** Parameters

| Parameter | Mandatory | Description |
| --- | --- | --- |
| --cli-region | Yes | Region where the target function is located. |
| --function_urn | Yes | Function URN. |
| --project_id | Yes | Project ID. |
| Body | Yes | Request body in **--key="value"** format. The JSON structure is **{"key":"value"}**. |

**Figure 10-8** shows the output result. For details about the response parameters, see **Table 10-3**.

**Figure 10-8** Output result

```
{
  "request_id": "3977431d-c254-4e65-979e-30541936651b"
}
```

**Table 10-5** Parameters in the response body

| Parameter | Type | Description |
| --- | --- | --- |
| request_id | String | Request ID. |

## Configuring a Network Proxy Using KooCLI

Run the following command to set a proxy:

```
export HTTP_PROXY="http://user:password@proxyIp:proxyPort"
```

For details, see **Set/Export http_proxy With Special Characters In Password Under Unix / Linux**.

# 11 Applying for More FunctionGraph Quotas

## Overview

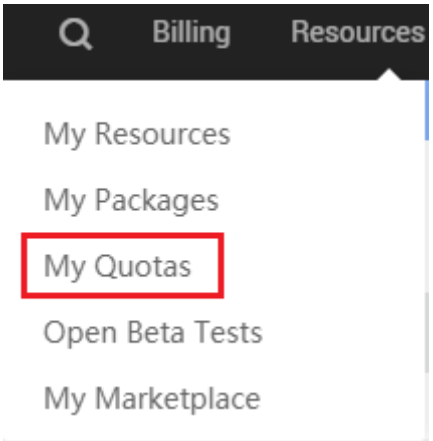
A quota is a limit on the quantity or capacity of a certain type of service resources that you can use. For example, the maximum number of ECSs or EVS disks that can be created.

If a quota cannot meet your needs, apply for a higher quota.

## Viewing Quotas

1. Log in to the management console.
2. In the upper right corner of the page, choose **Resources** > **My Quotas**.

   **Figure 11-1** Going to the Quotas page

3. On the **Quotas** page, view the used and total quotas of resources.

   If a quota cannot meet your needs, apply for a higher quota by performing the following operations.

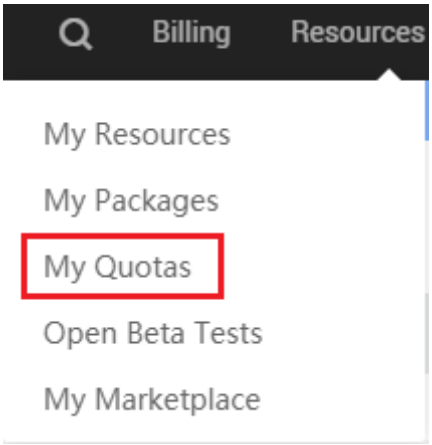
## Increasing Quotas

1. Log in to the management console.

2. In the upper right corner of the page, choose **Resources** > **My Quotas**. The **Quotas** page is displayed.

   **Figure 11-2** Going to the Quotas page

3. Click **Increase Quota**.
4. On the **Create Service Ticket** page, set the parameters.

   In the **Problem Description** area, enter the required quota and reason for the adjustment.
5. Read the agreements and confirm that you agree to them, and then click **Submit**.

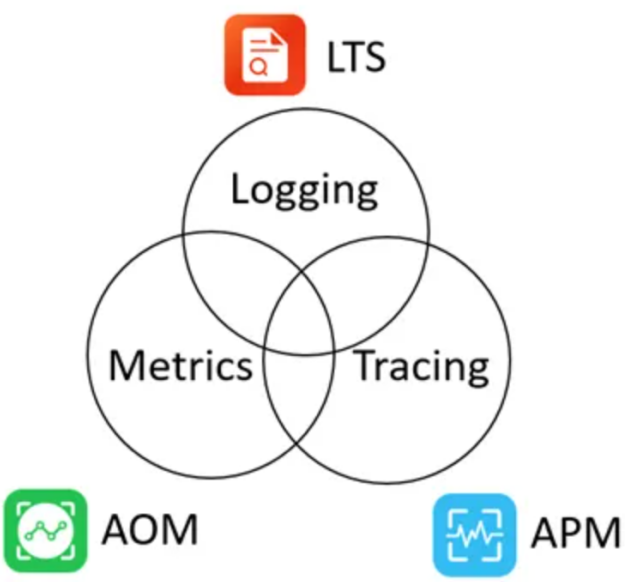
  

# 12 Viewing Metrics and Configuring Alarms

## 12.1 Introduction

FunctionGraph works closely with other Huawei Cloud services to provide you with an overall solution to the observability of serverless applications. It integrates with the monitoring, logging, tracing, and analysis services on Huawei Cloud, enabling you to effectively monitor and optimize function features, performance, and costs.

**Figure 12-1** Function monitoring overview




### One-Stop Log Collection and Storage with LTS

FunctionGraph supports one-click connection to LTS to collect, store, search, and analyze serverless application logs. LTS efficiently processes large number of logs

and provides real-time alarms. Its extensive log processing and statistics features help you quickly optimize your log data. By connecting to LTS, you can easily manage logs and monitor your serverless applications, enhancing log maintenance and application stability.

For details, see **Configuring and Viewing Function Invocation Logs**.

### Extensive Monitoring Metrics Based on AOM

FunctionGraph reports over 20 metrics related to function invocation (such as result, number of invocations, and invocation duration) and instance status (such as number of instances, asynchronous queuing requests, and memory usage) to AOM. These metrics give you insights into the performance and running status of your functions, so that you can precisely tune and optimize the functions, improving their efficiency and stability on Huawei Cloud.

With accurate real-time data, the integration with this monitoring system enables you to better monitor and optimize your functions, enhancing overall management and operational efficiency.

### Function Tracing with APM

FunctionGraph integrates with APM's monitoring technology and OpenTelemetry's distributed tracing to monitor the internal call chain of each function. It tracks all stages and dependencies during the execution, and accurately captures key information on the call chain, such as duration, parameters, and results. This data helps you understand your system performance and diagnose issues efficiently.

APM aggregates and analyzes data for specific requests when an exception occurs in a serverless function, enabling you to track duration changes and results within a specified period and identify unusual requests promptly. By deeply analyzing the abnormal invocation information and triggering time, you can precisely locate the root cause and efficiently rectify the fault.

For details, see **Configuring and Viewing Tracing**.

### Runtime Extension and Telemetry APIs

The **extension API** helps you enhance your functions with custom extensions.

The **telemetry API** provides a standard way for extension programs to subscribe to and receive telemetry data (such as logs, metrics, and trace data) of functions.

Together, these two APIs make it easy to customize monitoring and enhance your functions, while maintaining seamless integration with native function services.

## 12.2 FunctionGraph Metrics

### Introduction

This section describes FunctionGraph metrics reported to Cloud Eye as well as their namespaces and dimensions. You can use the Cloud Eye console or call APIs to query the FunctionGraph metrics and alarms.

## Namespaces

SYS.FunctionGraph

## Notes and Constraints

The metrics displayed on the console vary depending on the region. All metrics supported by FunctionGraph are listed here.

## Function Metrics

**Table 12-1** Function metrics

| Metric ID | Metric Name | Description | Value Range | Unit | Conversion Rule | Dimension | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|---|---|
| count | Invocations | Number of function invocations. | ≥ 0 | Count | None | package-functionname | 1 |
| failcount | Errors | Number of invocation errors.<br>The following errors are included:<br>• Function request error (causing an execution failure and returning status code 200)<br>• Function syntax or execution error | ≥ 0 | Count | None | package-functionname | 1 |
| failRate | Error Rate | Percentage of errors to the total invocations of a function. | 0% ≤X≤ 100% | % | None | package-functionname | 1 |

| Metric ID | Metric Name | Description | Value Range | Unit | Conversion Rule | Dimension | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|---|---|
| rejectcount | Throttles | Number of function throttles.<br>It indicates the number of times that FunctionGraph throttles your functions due to the resource limit. | ≥ 0 | Count | None | package-functionname | 1 |
| concurrency | Concurrency | The number of requests that can be processed concurrently. | ≥ 0 | Count | None | package-functionname | 1 |
| reservedinstancenum | Reserved Instances | Number of instances reserved for running a function. | ≥ 0 | Count | None | package-functionname | 1 |
| duration | Average Duration | Average duration of function invocation. | ≥ 0 ms | ms | None | package-functionname | 1 |
| maxDuration | Maximum Duration | Maximum duration of function invocation. | ≥ 0 ms | ms | None | package-functionname | 1 |
| minDuration | Minimum Duration | Minimum duration of function invocation. | ≥ 0 ms | ms | None | package-functionname | 1 |
| systemErrorCount | System Errors | Number of system errors that occur during function invocation. | ≥ 0 counts | Count | None | package-functionname | 1 |

| Metric ID | Metric Name | Description | Value Range | Unit | Conversion Rule | Dimension | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|---|---|
| functionErrorCount | Function Error Count | Number of function errors that occur during invocation. | ≥ 0 counts | Count | None | package-functionname | 1 |
| payPerUseInstance | Elastic Instances | Number of instances used, excluding reserved ones. | ≥ 0 | Count | None | package-functionname | 1 |
| instances | Instances | Number of instances for function invocation. | ≥ 0 | Count | None | package-functionname | 1 |
| duration_p500 | Duration (P50) | P50 execution duration of the function. | ≥ 0 ms | ms | None | package-functionname | 1 |
| duration_p800 | Duration (P80) | P80 execution duration of the function. | ≥ 0 ms | ms | None | package-functionname | 1 |
| duration_p950 | Duration (P95) | P95 execution duration of the function. | ≥ 0 ms | ms | None | package-functionname | 1 |
| duration_p990 | Duration (P990) | P990 execution duration of the function. | ≥ 0 ms | ms | None | package-functionname | 1 |
| duration_p999 | Duration (P999) | P999 execution duration of the function. | ≥ 0 ms | ms | None | package-functionname | 1 |

| Metric ID | Metric Name | Description | Value Range | Unit | Conversion Rule | Dimension | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|---|---|
| memoryUsed | Memory | Memory used by the function. | ≥ 0 MB | MB | None | package-functionname | 1 |
| functionCost | Resource Usage | Resources used by the function (Memory x Duration). | ≥ 0 MB | MB | None | package-functionname | 1 |
| instanceUsage | Used Instances | Percentage of the used function instances to the total number. | 0% ≤X≤ 100% | % | None | package-functionname | 1 |

**Table 12-2** Flow metrics

| Metric ID | Metric Name | Description | Value Range | Unit | Conversion Rule | Dimension | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|---|---|
| totalCount | Number of calls | Total number of executions of workflow. | ≥ 0 | Count | None | graph_name | 1 |
| errorCount | Number of errors | Total number of execution failures of workflow. | ≥ 0 | Count | None | graph_name | 1 |
| running | Number of running | Number of running flows. | ≥ 0 | Count | None | graph_name | 1 |

| Metric ID | Metric Name | Description | Value Range | Unit | Conversion Rule | Dimension | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|---|---|
| rejectCount | Number of times to be restricted | Number of restricted flows. | ≥ 0 | Count | None | graph_name | 1 |
| averageDuration | Run time | Average execution time of workflow. | ≥ 0 ms | ms | None | graph_name | 1 |
| notifyFailedCount | Number of notify errors | Total number of notified failures. | ≥ 0 | Count | None | graph_name | 1 |
| ExecutionsStarted | Started Workflow Instances | Number of started workflow instances. | ≥ 0 | Count | None | graph_name | 1 |
| ExecutionsSucceeded | Successful Workflow Instances | Number of successfully executed workflow instances. | ≥ 0 | Count | None | graph_name | 1 |
| ExecutionsFailed | Failed Workflow Instances | Number of failed workflow instances. | ≥ 0 | Count | None | graph_name | 1 |
| ExecutionsAborted | Aborted Workflow Instances | Number of aborted workflow instances. | ≥ 0 | Count | None | graph_name | 1 |
| ExecutionsTimedOut | Timed-Out Workflow Instances | Number of workflow instances that timed out. | ≥ 0 | Count | None | graph_name | 1 |

### Dimensions

| Key | Value |
| --- | --- |
| package-functionname | App name-Function name. Example: default-myfunction_Python<br>For details about how to obtain the metric, see **Basic Settings**. |
| graph_name | Flow name. |
| projectId | Tenant's project ID.<br>For details about how to obtain the metric, see **Obtaining a Project ID**. |

## 12.3 Viewing FunctionGraph Metrics

FunctionGraph is interconnected with Application Operations Management (AOM), allowing you to query function metrics without any configurations.

### Viewing Function Metrics

FunctionGraph collects function metrics and displays aggregated results. Switch to your target function version before viewing metrics.

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the function to be configured to go to the function details page.
3. Choose **Monitoring** > **Metrics**, select an interval (last hour, last 3 hours, last 12 hours, last day, last 3 days, or custom), and check the running status of the function.

### Metric Description

**Table 12-3** describes the function metrics.

**Table 12-3** Function metrics

| Metric | Unit | Description |
| --- | --- | --- |
| Invocations | Count | Total number of invocation requests, including invocation errors and throttled invocations. In case of asynchronous invocation, the count starts only when a function is executed in response to a request. |

| Metric | Unit | Description |
| --- | --- | --- |
| Duration | ms | **Maximum duration**: the maximum duration a function is executed within a period.<br>**Minimum duration**: the minimum duration a function is executed within a period.<br>**Average duration**: the average duration a function is executed within a period. Formula: Average duration = Execution duration/ Invocations. |
| Errors/Error rate | Count | **System Errors**: The number of system errors during function invoking, excluding throttling errors and execution errors.<br>**Function Error Count**: Number of times that your functions failed with error code **200** being returned. Errors caused by function syntax or execution are also included.<br>**Error Rate**: The error rate of function invoking. Formula: Error Rate = (Function Error Count + System Errors)/Invocations. |
| Throttles | Count | Number of times that FunctionGraph throttles your functions due to the resource limit. |
| Instance Statistics | Count | **Concurrency**: The number of concurrent requests of the function.<br>**Reserved Instances**: The number of reserved instances of the function.<br>**Elastic Instances**: The number of elastic instances of the function. |
| Memory Usage | MB | Currently, this metric is available only in **CN East-Shanghai1** and **CN North-Beijing4**.<br>**Maximum**: the maximum memory usage during function execution within a period.<br>**Minimum**: the minimum memory usage during function execution within a period.<br>**Average**: the average memory usage during function execution within a period. Formula: Average Memory Usage = Total memory usage/ Invocations. |

## Helpful Links

- For details about FunctionGraph monitoring, see **Introduction**.
- For details about the namespace, metrics, and dimensions reported by FunctionGraph to Cloud Eye, see **FunctionGraph Metrics**.
- For details about how to configure FunctionGraph monitoring alarms, see **Configuring an Alarm Rule**.

# 12.4 Configuring an Alarm Rule

After creating a function and trigger, you can monitor the invocation and running statuses of the function in real time.

## Notes and Constraints

After a function is deleted, the alarm rules created for it will not be updated in real time on the Cloud Eye console and may continue to be displayed there for a maximum of seven days.

## Viewing Function Metrics

FunctionGraph differentiates the metrics of a function by version, allowing you to query the metrics of a specific function version.

## Procedure

Create an alarm rule for a function to report metrics to Cloud Eye so that you can view monitoring graphs and alarm messages on the Cloud Eye console.

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

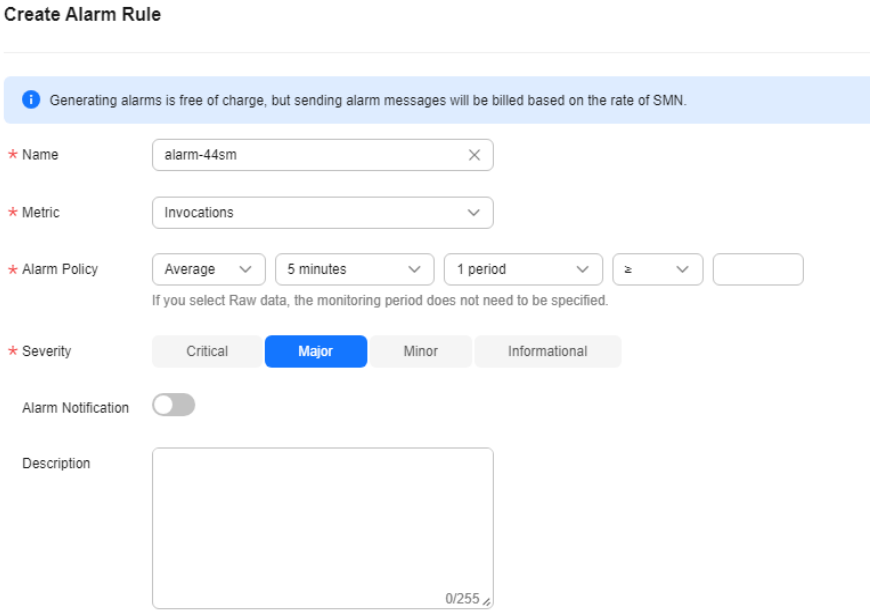
**Step 2** Click the name of the desired function.

**Step 3** On the displayed function details page, select a function version or alias, and choose **Monitoring** > **Metrics**.

**Step 4** Click **Create Alarm Rule**.

**Step 5** Set alarm parameters and click **Next** as shown in **Figure 12-2**.

**Figure 12-2** Creating an alarm rule



**Step 6** Enter a rule name and click **OK**.

**----End**

# 12.5 Configuring and Viewing Function Invocation Logs

FunctionGraph is interconnected with LTS, allowing you to view function logs without the need for any configurations.

On the FunctionGraph console, view function logs in the following ways:

- Viewing logs on the execution result page

  After creating a function, test it and view test logs on the execution result page. For details, see **Debugging Functions Online**.

  The execution result page displays a maximum of 2 KB logs. To view more logs of the function, go to the **Logs** tab page.

- Viewing logs on the **Logs** tab page

  On the function details page, choose **Monitoring** > **Logs** to query log information. For details, see **Using LTS to Manage Function Logs**.

## Configuring Log Groups and Log Streams, and Viewing Function Logs

You can bind a log group and log stream to a function to store its invocation logs. By default, the logs are stored in the log stream automatically created for the function. For details, see **Using LTS to Manage Function Logs**.

**Procedure**

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

**Step 2** Click the name of the desired function.

**Step 3** Choose **Configuration** > **Log**, and configure log collection according to **Table 12-4**.

**Table 12-4** Log configuration parameters

| Parameter | Description |
| --- | --- |
| Collect Logs | Enabled by default in FunctionGraph V2. This feature is unavailable for FunctionGraph V1. |
| Log Group | Select a log group for the function. The default log group **functiongraph.log.group.***xxx* cannot be selected. For details about how to create a log group, see **Managing Log Groups**.<br>By default, the log group (starting with **functiongraph**) automatically generated by FunctionGraph is selected. |

| Parameter | Description |
| --- | --- |
| Log Stream | Select a log stream in the specified log group. For details about how to create a log group, see **Managing Log Streams**.<br>By default, the log stream (starting with the function name) automatically generated for the function is selected. |

**Step 4** Click **Save**.

NOTE

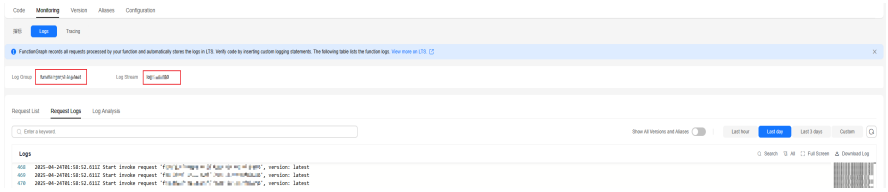
You can change the log stream if needed.

**Step 5** Click the **Code** tab and click **Test** to execute the function.

**Step 6** After the function is executed, choose **Monitoring** > **Logs** to view the function logs.

- As shown in **Figure 12-3**, logs of the function are generated in the specified log group and log stream.
- If no custom log group and log stream are specified, the default log group (starting with **functiongraph**) and log stream are displayed.

**Figure 12-3** Viewing function logs



- The following figure shows two successful requests. The request at the bottom took 13.100 ms, including the cold start time. The request at the top took only 1.671 ms, because no cold start was involved.

**Figure 12-4** Request

**----End**

## Log Tag

You can configure tags to report logs to LTS when a function is executed. You can filter function logs by tag.

**Step 1** When creating a function, you can click **Collect Logs** in the **Advanced Settings** area to add log tags. If the function has been created, go to **Step 2**.

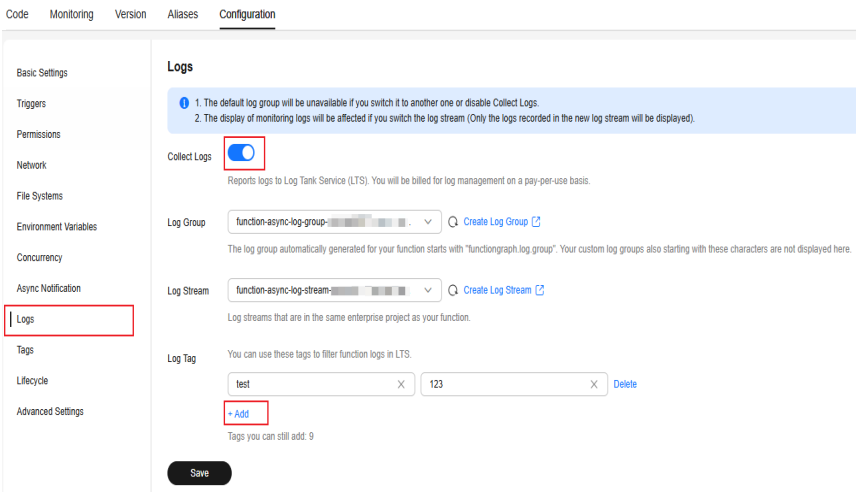
**Step 2** On the function details page, choose **Configuration** > **Logs**, enable **Collect Logs**, and select a log group and log stream.

**Step 3** Click **Add**, enter the tag key and value, and click **Save**. A maximum of 10 tags can be added.

**Table 12-5** Log tag description

| Parameter | Description |
| --- | --- |
| Tag key | Cannot be left blank. The value can contain a maximum of 64 characters, including only digits, letters, underscores (_), and hyphens (-). |
| Tag value | |

**Figure 12-5** Adding a log tag

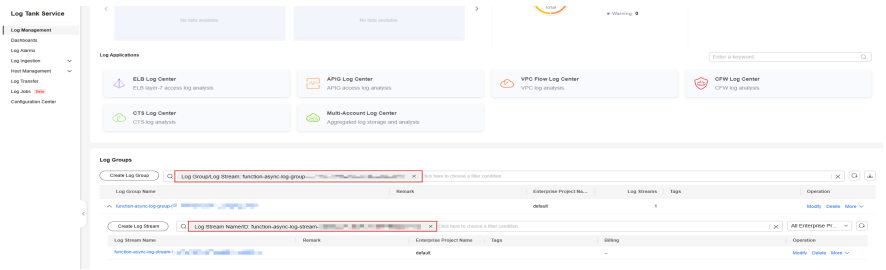



**Step 4** Click the **Code** tab, configure the test event, and click **Test**. Wait until the function execution is complete.

**Step 5** Choose **Monitoring** > **Logs**. On the **Request List** page, view the function request records. (Log reporting may be delayed. Please wait for a few minutes.)

**Step 6** Log in to the LTS console. In the navigation pane, choose **Log Management**. Set the filter criteria to search for the log group and stream. Click the log stream name to go to the details page.

**Figure 12-6** Searching for a log stream



**Step 7** On the log stream details page, click **Log Settings**. On the **Index Settings** panel, click **Add Field** to add the log tag key added to the function. For details about how to set indexes, see **Configuring Log Indexing**. Click **OK**.

**Step 8** Return to the log stream details page, enter the log tag key and value configured for the function in the search box to view the matched logs.

**----End**

## Using LTS to Manage Function Logs

You can enable LTS to better manage function logs. After you enable LTS, FunctionGraph automatically creates a log group starting with **functiongraph**. When you create a function, a log stream starting with the function name is generated.

You can also bind a log group and log stream to a function to store its invocation logs. For details, see **Configuring Log Groups and Log Streams, and Viewing Function Logs**.

**Constraints:**

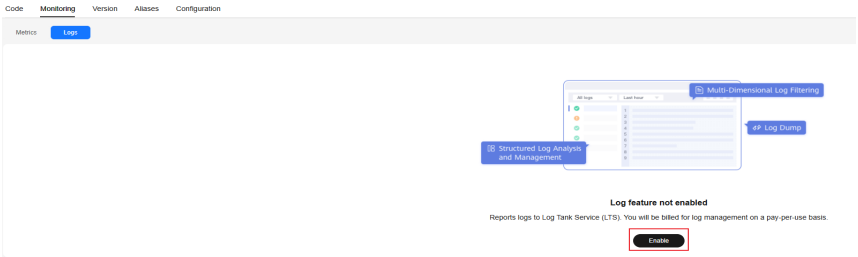
- By default, 20 log streams are created, which cannot be customized. On the **Logs** tab page of the function, press **F12** to find out the log stream ID of the **query** API, and then locate the corresponding log stream ID in LTS.

- Deleting a function log group by mistake on the LTS console will not be detected by FunctionGraph, and the historical log data can no longer be retrieved. To use a log group, modify the function description and save the changes. A new log group will be created.
- Max. 10 MB logs can be retained for common instances during initialization. When this limit is reached, the latest logs replace the old ones.

**Step 1** Enable LTS.

On the **Logs** tab, click **Enable**. The **Logs** page is displayed. Click **Collect Logs**.

**Figure 12-7** Enabling logging




**Step 2** Set filter criteria.

- **Request List**: Filter requests by request ID, **result** (success or failure), duration, actual memory or **cause** (initialization failed, load failed, system error, timed out, out of memory, out of disk space, or code error).

  Before using query fields, configure them in LTS. For details, see **Configuring Log Filtering Fields for a Function**.
- **Request Log**: Filter logs by keyword, request ID, or instance ID.

**Table 12-6** Invocation result

| Result | Description |
| --- | --- |
| Execution successful | Log printed when a function is successfully executed. |
| Execution failed | Log printed when a function fails to be executed due to invocation timeout, memory or disk threshold exceeded, or code errors.<br>To view the logs about invocation timeout, select **Invocation timed out** from the drop-down list. The methods for viewing the other three types of logs are the same. |

**Table 12-7** Cause analysis

| Possible Cause | Description |
| --- | --- |
| Initialization failed | Log printed when the function initialization fails. |
| Load failed | Log generated when the runtime fails to load your function file. |
| System error | Internal error. |
| Invocation timed out | Log printed when the function invocation period is longer than the preset limit. |
| Memory threshold exceeded | Log printed when the function memory size exceeds the preset limit. |
| Disk threshold exceeded | Log printed when the disk size exceeds the preset limit. |
| Code error | Log printed when a code error occurs. |

**NOTE**

- You can view logs of the last hour, last day, last 3 days, or a custom time period.
- To manage function logs, go to the LTS console.

**Step 3** Use log analysis.

On the **Log Analysis** tab page, you can configure log filtering fields for functions by referring to **Configuring Log Filtering Fields for a Function**. For details about how to analyze logs on LTS, see **Searching and Analyzing Logs**.

**----End**

## Configuring Log Filtering Fields for a Function

The log query function of FunctionGraph supports multiple filtering fields. First, perform the following steps:

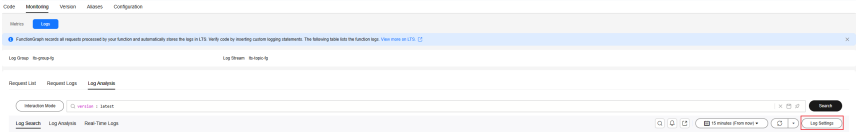
**Step 1** Return to the FunctionGraph console, and choose **Functions** > **Function List** in the navigation pane.

**Step 2** Click the name of a function.

**Step 3** Click **Monitoring**. On the **Logs** tab page, click the **Log Analysis** tab to analyze logs using LTS on the FunctionGraph console.

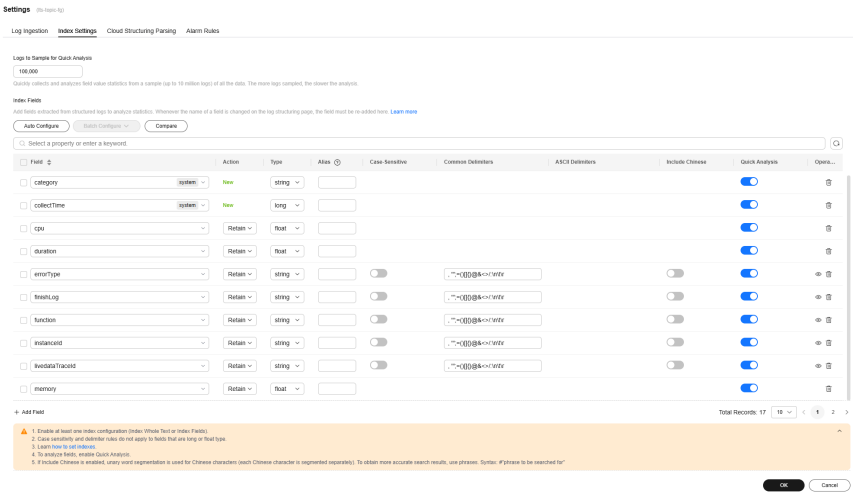
**Step 4** Click **Log Settings** on the right. On the **Index Settings** page, click **Auto Configure**. Wait until the backend automatically configures all fields.

**Figure 12-8** Log settings



**Step 5** After the automatic configuration is complete, change the type of the **memory** and **duration** fields to **float**. Then check whether the **errorType** field exists, if it does not exist, manually add it, as shown in **Figure 12-9**.

**Figure 12-9** Configuring field information



**Step 6** Click **OK**. Return to the **Monitoring** > **Logs** > **Request List** tab page of the function to use related fields to filter information.

**----End**

### Downloading Logs

**Constraints:**

- A maximum of 5,000 logs can be downloaded at a time. When querying logs, select a proper time range to avoid log loss.
- The timestamp of logs is in UTC.

**Step 1** Choose **Monitoring** > **Logs** > **Request Logs**.

**Step 2** Select a time range, and click **Download Log**.

**----End**

## 12.6 Configuring and Viewing Tracing

### Overview

After enabling tracing on the **Monitoring** tab, you can view details on the **Tracing** page, or go to the APM console and choose **Application Monitoring** > **Tracing**. This feature is available only for Java 8 and 11 functions.

### Notes and Constraints

This feature is available only in the **CN East-Shanghai1** and **CN North-Beijing4** regions.

### Prerequisites

- Only functions with 512 MB or larger memory can be traced. For functions whose memory is less than 512 MB, increase the memory on the **Configuration** > **Basic Settings** page.
- You have obtained the permission to use APM. For details, see **APM Permissions Management**. If tracing is not enabled, no trace data can be obtained.

### Enabling Tracing

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

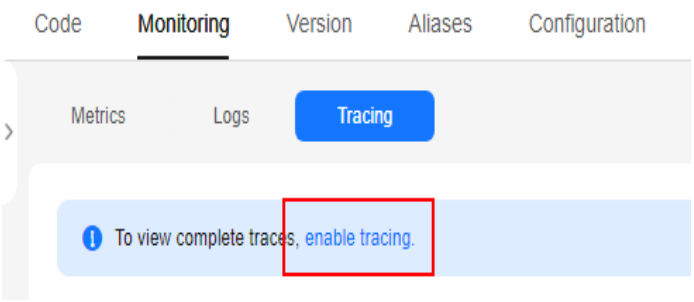
**Step 2** Click the name of the desired function.

**Step 3** Choose **Monitoring** > **Tracing**.

**Step 4** Click **enable tracing**.

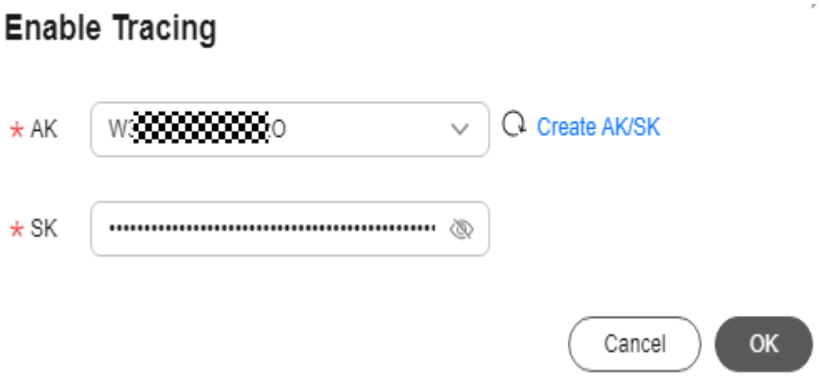
  

**Figure 12-10** Enabling tracing




**Step 5** The system automatically obtains an access key (AK/SK) from APM, as shown in **Figure 12-11**.

**Figure 12-11** Obtaining an access key




- If an access key already exists, select an AK from the AK drop-down list and click **OK**.
- If you do not have an access key, click **Create AK/SK** to create one on the APM console. For details, see **Creating an Access Key**. Then the new AK/SK will be synchronized to the FunctionGraph console.

**Step 6** Manage tracing.

- Click **Disable Tracing** in the upper right corner. In this case, you cannot view function traces.
- If the AK/SK has been changed on the APM console, you need to go to the FunctionGraph console and click **Update AK/SK** in the upper right corner.

**----End**

## Querying Trace Details

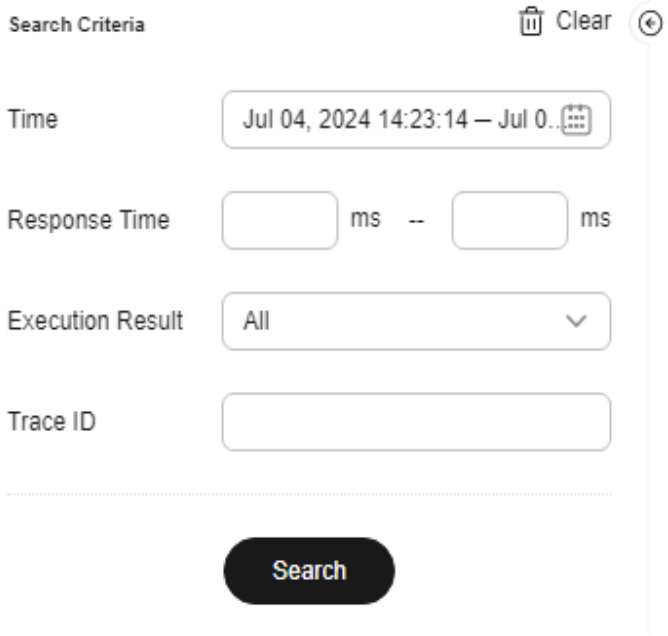
**Step 1** Return to the FunctionGraph console, choose **Functions** > **Function List** in the navigation pane, and click the name of a function with tracing enabled. The function details page is displayed.

**Step 2** On the **Monitoring** tab page, choose **Tracing**.

**Step 3** Set search criteria and click **Search**.

**Figure 12-12** Setting search criteria




- **Time**: Set a time range to query traces. It can be up to 24 hours.
- **Response Time**: Set the response time.
- **Execution Result**: Select **All**, **Successful**, or **Failed**.
- **Trace ID**: If you specify this parameter, all other search criteria become invalid and only the trace with the specified ID will be searched.

**Step 4** View the trace details on the right.

Viewing details about a trace: In the query result, click the trace name and view details on the APM console.

**Figure 12-13** Clicking a trace name

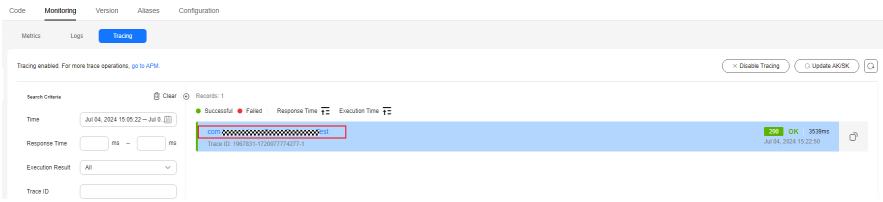
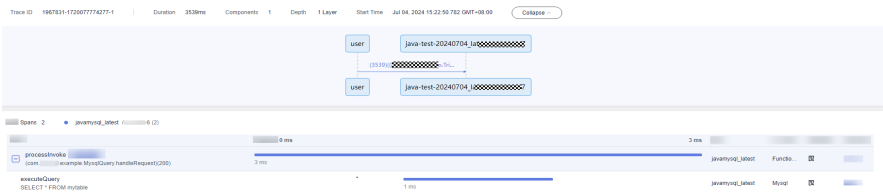


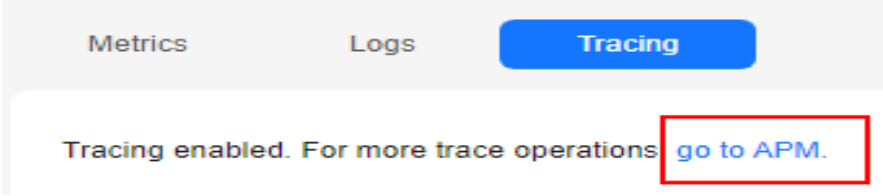
**Figure 12-14** Details about a trace



Viewing information about all traces: Click **go to APM** to perform complete path analysis and other operations.

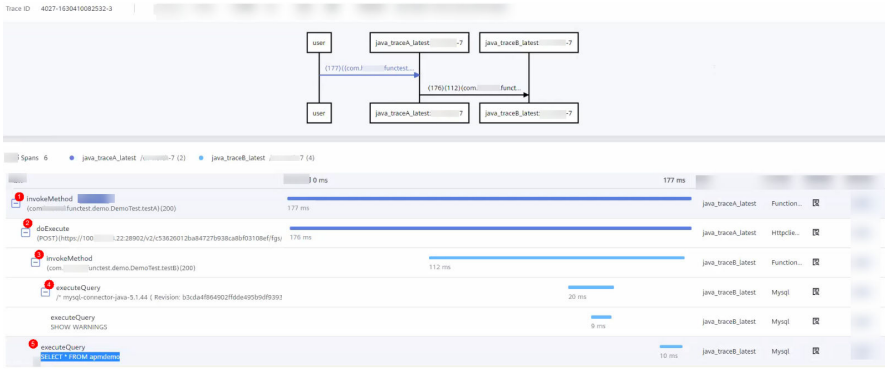
**Figure 12-15** Going to APM




**Example**

1. The following shows how function A (**DemoTestA**) invokes function B (**DemoTestB**) by sending an HTTP request.

   **Figure 12-16** Function invocation details

   1. Total time consumed by a method of function A
   2. Invoking function B by sending an HTTP request
   3. Accessing function B
   4. Executing **executeQuery**
   5. Executing a SELECT statement to query data

2. The following shows a complete function execution process that contains cold start.

   Spans are described as follows:

   - **load**: time for downloading and decompressing the function code package and dependency package
   - **preload**: time for loading function code and initializing the execution environment
   - **init**: execution duration of the initializer. The initializer is executed only during cold start.
   - **processInvoke**: execution duration of the function

   NOTE

   For more information, see section "Tracing".

**----End**

# 13 Querying Audit Logs

## 13.1 FunctionGraph Operations Supported by CTS

Table 13-1 lists the FunctionGraph operations that can be logged by CTS.

**Table 13-1** Operations logged by CTS

| Operation | Resource Type | Event Name |
| --- | --- | --- |
| Creating a function | Function | createFunction |
| Deleting a function | Function | deleteFunction |
| Modifying function information | Function | updateFunctionConfig |
| Publishing a function version | Function version | publishFunctionVersion |
| Deleting a function version alias | Function version alias | deleteVersionAlias |
| Deleting a function trigger | Trigger | deleteTrigger |
| Creating a function trigger | Trigger | createTrigger |
| Disabling a function trigger | Trigger | disableTrigger |
| Enabling a function trigger | Trigger | enableTrigger |

# 13.2 Viewing CTS Traces in the Trace List

## Scenarios

Cloud Trace Service (CTS) records operations performed on cloud service resources. A record contains information such as the user who performed the operation, IP address, operation content, and returned response message. These records facilitate security auditing, issue tracking, and resource locating. They also help you plan and use resources, and identify high-risk or non-compliant operations.

## What Is a Trace?

A trace is an operation log for a cloud service resource, tracked and stored by CTS. Traces record operations such as adding, modifying, or deleting cloud service resources. You can view them to identify who performed operations and when for detailed tracking.

## What Is a Management Tracker and Data Tracker?

A management tracker identifies and associates with all your cloud services, recording all user operations. It records management traces, which are operations performed by users on cloud service resources, such as their creation, modification, and deletion.

A data tracker records details of user operations on data in OBS buckets. It records data traces reported by OBS, detailing user operations on data in OBS buckets, including uploads and downloads.

## Constraints

- Before the organization function is enabled, you can query the traces of a single account on the CTS console. After the organization function is enabled, you can only view multi-account traces on the **Trace List** page of each account, or in the OBS bucket or the **CTS/system** log stream configured for the management tracker with the organization function enabled. For details about organization trackers, see **Organization Trackers**.
- You can only query operation records of the last seven days on the CTS console. They are automatically deleted upon expiration and cannot be manually deleted. To store them for longer than seven days, configure transfer to Object Storage Service (OBS) or Log Tank Service (LTS) so that you can view them in the OBS buckets or LTS log streams.
- After creating, modifying, or deleting a cloud service resource, you can query management traces on the CTS console 1 minute later and query data traces 5 minutes later.
- Data traces are not displayed in the trace list of the new version. To view them, you need to go to the old version.

## Prerequisites

1. **Register with Huawei Cloud and complete real-name authentication.**

If you already have a Huawei Cloud account, skip this step. If you do not have one, do as follows:

a. Log in to the **Huawei Cloud official website**, and click **Sign Up**.

b. Sign up for a HUAWEI ID as prompted. For details, see **Signing Up for a HUAWEI ID and Enabling Huawei Cloud Services**.

Your personal information page is displayed after the registration completes.

c. Complete individual or enterprise real-name authentication by referring to **Real-Name Authentication**.

2. **Grant permissions for users.**

If you log in to the console using a Huawei Cloud account, skip this step.

If you log in to the console as an IAM user, first contact your CTS administrator (account owner or a user in the **admin** user group) to obtain the **CTS FullAccess** permissions. For details, see **Assigning Permissions to an IAM User**.

## Viewing Traces

After you enable CTS and the management tracker is created, CTS starts recording operations on cloud resources. After a data tracker is created, CTS starts recording user operations on data in OBS buckets. CTS retains operation records of the latest seven days.

This section describes how to query and export operation records of the last seven days on the CTS console.

## Viewing Real-Time Traces in the Trace List of the New Edition

**Step 1** Log in to the **CTS console**.

**Step 2** Log in to the management console, click  in the upper left corner, and choose **Management & Deployment** > **Cloud Trace Service**.

**Step 3** In the navigation pane, choose **Trace List**.

**Step 4** In the time range drop-down list above the trace list, select a desired query time range: **Last 1 hour**, **Last 1 day**, or **Last 1 week**. You can also select **Custom** to specify a custom time range within the last seven days.

**Step 5** The search box above the trace list supports advanced queries. Combine one or more filters to refine your search.

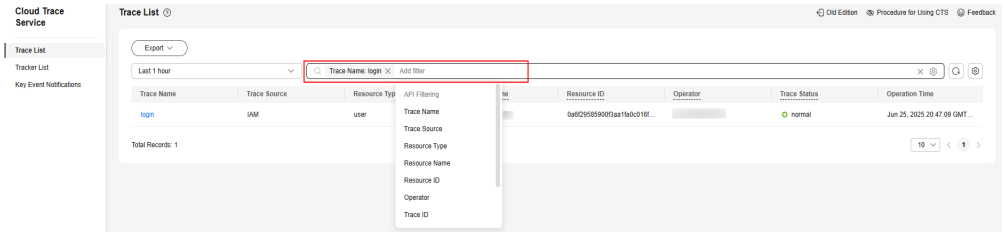
  

**Table 13-2** Trace filtering parameters

| Parameter | Description |
| --- | --- |
| Read-Only | After selecting the **Read-Only** filter, you can select either **Yes** or **No** from the drop-down list.<br>• **Yes**: filters read-only operation traces, for example, resource query operations. This option is available after **Read-Only Trace Reporting** has been enabled in the **Configuration Center** and at least one read-only trace has been triggered.<br>• **No**: filters non-read-only operation traces, such as creating, modifying, and deleting resources. |
| Trace Name | Name of a trace.<br>The entered value is case-sensitive and requires an exact match. Fuzzy matching is not supported.<br>For details about the operations that can be audited for each cloud service, see **Supported Services and Operations**section "Supported Services and Operations" in the *Cloud Trace Service User Guide*.<br>Example: **updateAlarm** |
| Trace Source | Cloud service name abbreviation.<br>The entered value is case-sensitive and requires an exact match. Fuzzy matching is not supported.<br>Example: **IAM** |
| Resource Name | Name of a cloud resource involved in a trace.<br>The entered value is case-sensitive and requires an exact match. Fuzzy matching is not supported.<br>If the cloud resource involved in the trace does not have a resource name or the corresponding API operation does not involve the resource name parameter, leave this field empty.<br>Example: **ecs-name** |
| Resource ID | ID of a cloud resource involved in a trace.<br>The entered value is case-sensitive and requires an exact match. Fuzzy matching is not supported.<br>Leave this field empty if the resource has no resource ID or if resource creation failed.<br>Example: *{VM ID}* |
| Trace ID | Value of the **trace_id** parameter for a trace reported to CTS.<br>The entered value requires an exact match. Fuzzy matching is not supported.<br>Example: **01d18a1b-56ee-11f0-ac81-******1e229** |

| Parameter | Description |
| --- | --- |
| Resource Type | Type of a resource involved in a trace.<br>The entered value is case-sensitive and requires an exact match. Fuzzy matching is not supported.<br>For details about the resource types of each cloud service, see **Supported Services and Operations**section "Supported Services and Operations" in the *Cloud Trace Service User Guide*.<br>Example: **user** |
| Operator | User who triggers a trace.<br>Select one or more operators from the drop-down list.<br>If the value of **trace_type** in a trace is **SystemAction**, the operation is triggered by the service and the trace's operator may be empty.<br>For details about the relationship between IAM identities and operators and the operator username format, see **Relationship Between IAM Identities and Operators**. |
| Trace Status | Select one of the following options from the drop-down list:<br>• **normal**: The operation succeeded.<br>• **warning**: The operation failed.<br>• **incident**: The operation caused a fault that is more serious than a normal failure, for example, causing other faults. |
| Enterprise Project ID | ID of the enterprise project to which a resource belongs.<br>To check enterprise project IDs, go to the Enterprise Project Management Service (EPS) console and choose **Project Management** in the navigation pane.<br>Example: **b305ea24-c930-4922-b4b9-******1eb2** |
| Access Key | Temporary or permanent access key ID.<br>To check access key IDs, hover over your username in the upper right corner of the console and select **My Credentials** from the pop-up list. On the displayed page, choose **Access Keys** in the navigation pane.<br>Example: **HSTAB47V9V*******TLN9** |




**Step 6** On the **Trace List** page, you can also export and refresh the trace list, and customize columns to display.

- Enter any keyword in the search box and press **Enter** to filter desired traces.
- Click **Export** to export all traces in the query result as an .xlsx file. The file can contain up to 5,000 records.
- Click to view the latest information about traces.
- Click to customize the information to be displayed in the trace list. If **Auto wrapping** is enabled ( ), excess text will move down to the next line; otherwise, the text will be truncated. By default, this function is disabled.

**Step 7** (Optional) On the **Trace List** page of the new edition, click **Old Edition** in the upper right corner to switch to the **Trace List** page of the old edition.

**----End**

## Viewing Traces in the Trace List of the Old Edition

**Step 1** Log in to the **CTS console**.

**Step 2** Log in to the management console, click in the upper left corner, and choose **Management & Deployment** > **Cloud Trace Service**.

**Step 3** In the navigation pane, choose **Trace List**.

**Step 4** Each time you log in to the CTS console, the new edition is displayed by default. Click **Old Edition** in the upper right corner to switch to the trace list of the old edition.

**Step 5** In the upper right corner of the page, set a desired query time range: **Last 1 hour**, **Last 1 day**, or **Last 1 week**. You can also click **Customize** to specify a custom time range within the last seven days.

**Step 6** Set filters to search for your desired traces.

**Table 13-3** Trace filtering parameters

| Parameter | Description |
| --- | --- |
| Trace Type | Select **Management** or **Data**.<br>• Management traces record operations performed by users on cloud service resources, including creation, modification, and deletion.<br>• Data traces are reported by OBS and record operations performed on data in OBS buckets, including uploads and downloads. |
| Trace Source | Select the name of the cloud service that triggers a trace from the drop-down list. |
| Resource type | Select the type of the resource involved in a trace from the drop-down list.<br>For details about the resource types of each cloud service, see **Supported Services and Operations**section "Supported Services and Operations" in the *Cloud Trace Service User Guide*. |

| Parameter | Description |
| --- | --- |
| Search By | Select one of the following options:<br>• **Resource ID**: ID of the cloud resource involved in a trace. Leave this field empty if the resource has no resource ID or if resource creation failed.<br>• **Trace name**: name of a trace. For details about the operations that can be audited for each cloud service, see **Supported Services and Operations**section "Supported Services and Operations" in the *Cloud Trace Service User Guide*.<br>• **Resource name**: name of the cloud resource involved in a trace. If the cloud resource involved in the trace does not have a resource name or the corresponding API operation does not involve the resource name parameter, leave this field empty. |
| Operator | User who triggers a trace.<br>Select one or more operators from the drop-down list.<br>If the value of **trace_type** in a trace is **SystemAction**, the operation is triggered by the service and the trace's operator may be empty.<br>For details about the relationship between IAM identities and operators and the operator username format, see **Relationship Between IAM Identities and Operators**. |
| Trace Status | Select one of the following options:<br>• **Normal**: The operation succeeded.<br>• **Warning**: The operation failed.<br>• **Incident**: The operation caused a fault that is more serious than a normal failure, for example, causing other faults. |

**Step 7** Click **Query**.

**Step 8** On the **Trace List** page, you can also export and refresh the trace list.

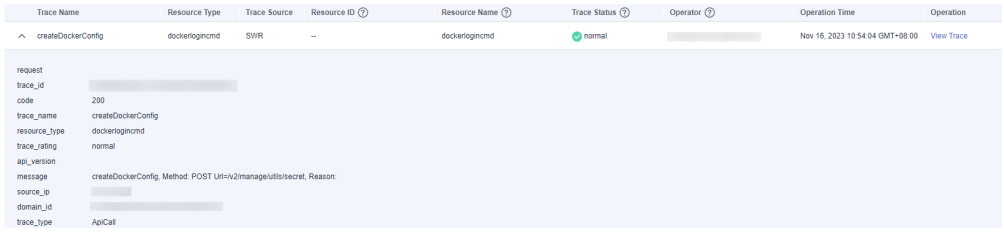
- Click **Export** to export all traces in the query result as a CSV file. The file can contain up to 5,000 records.
- Click to view the latest information about traces.

**Step 9** In the **Tampered or Not** column of a trace, check whether the trace is tampered with.

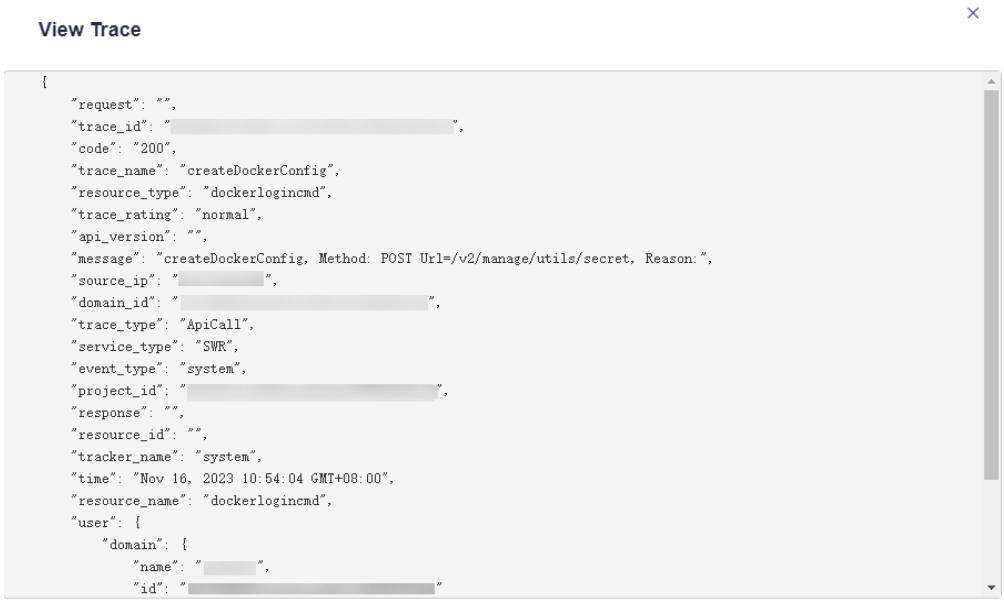
- **No**: The trace is not tampered with.
- **Yes**: The trace is tampered with.

**Step 10** Click on the left of a trace to expand its details.




**Step 11** Click **View Trace** in the **Operation** column. The trace details are displayed.




**Step 12** (Optional) On the **Trace List** page of the old edition, click **New Edition** in the upper right corner to switch to the **Trace List** page of the new edition.

**----End**

## Helpful Links

- For details about the key fields in the trace structure, see **Trace Structure**section "Trace References" > "Trace Structure" in the *Cloud Trace Service User Guide* and **Example Traces**section "Trace References" > "Example Traces" in the *Cloud Trace Service User Guide*.
- You can use the following examples to learn how to query a specific trace:
  - Use CTS to audit Elastic Volume Service (EVS) creation and deletion operations from the last two weeks. For details, see **Security Auditing**.
  - Use CTS to locate a fault or creation failure for an Elastic Cloud Server (ECS). For details, see **Fault Locating**.
  - Use CTS to check all operation records for an ECS. For details, see **Resource Tracking**.