# 02 User Guide

**Issue** 01

**Date** 2025-02-28

HUAWEI

**HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.**

### Trademarks and Permissions


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

### Notice

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

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

## 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



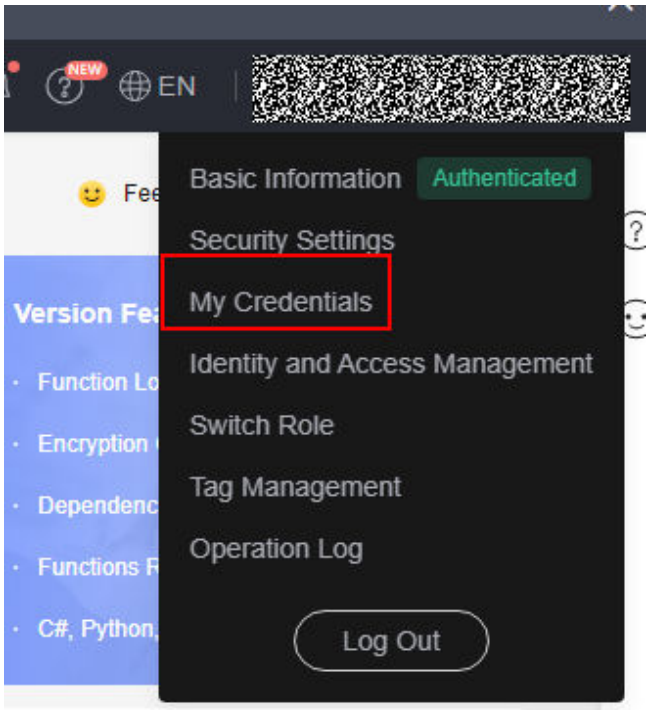
  

# 1 Replacing the Temporary AK/SK

## 1.1 Using Permanent AK/SK

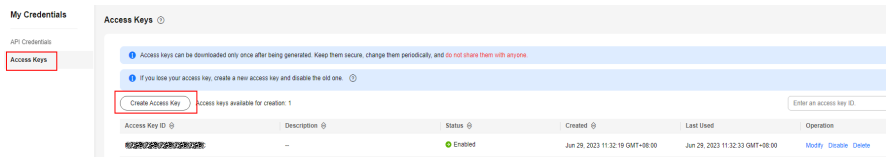
1. Log in to the **FunctionGraph console** and choose **My Credentials**.

   **Figure 1-1** My credentials




2. Choose **Access Keys** and click **Create Access Key**. For details, see **Managing Access Keys for an IAM User**.

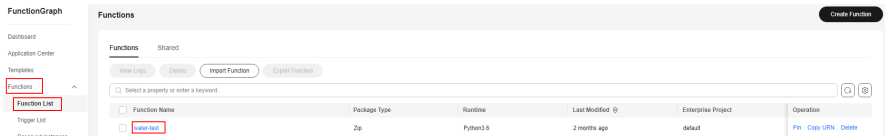
   **Figure 1-2** Creating an access key



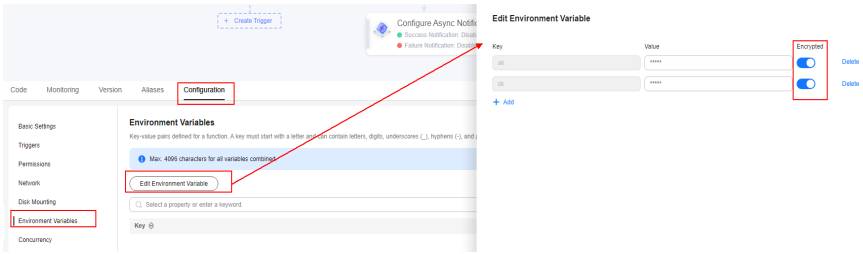
3. Return to the FunctionGraph console, choose **Functions** > **Function List**, and click a function name to go to the function details page.

**Figure 1-3** Viewing function details




4. Choose **Configuration** > **Environment Variables**, click **Edit Environment Variable**, add the AK/SK environment variables, enable **Encrypted**, and click **OK**.

**Figure 1-4** Adding AK/SK environment variables




5. Click the **Code** tab and replace **context.getAccessKey()** and **context.getSecretKey()** in the original code with **context.getUserData("ak")** and **context.getUserData("sk")**, respectively. (Python is used as an example.)

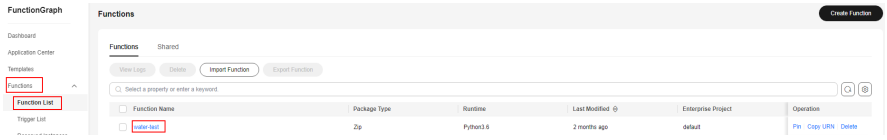
**Figure 1-5** Modifying the code for obtaining AK/SK


6. Click **Deploy** to refresh the code.

# 1.2 Using STS Token

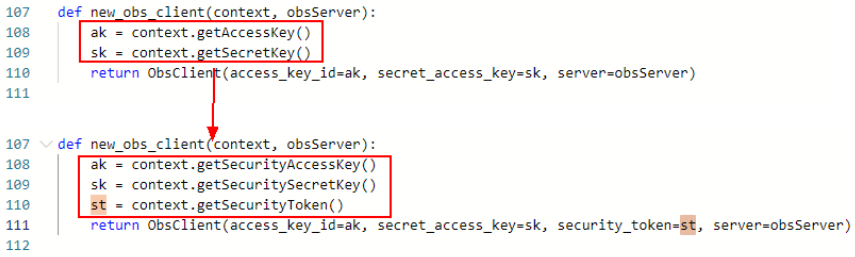
1. Log in to the **FunctionGraph console**, choose **Functions** > **Function List**, and click a function name to go to the function details page.

**Figure 1-6** Viewing function details



2. Click the **Code** tab and replace **context.getAccessKey()** and **context.getSecretKey()** in the original code with **context.getSecurityAccessKey()**, **context.getSecuritySecretKey()**, and **context.getSecurityToken()**. (Python is used as an example. For details about other languages, see **Other Languages**.)

**Figure 1-7** Modifying code




NOTE

The STS token must contain the **securityAccessKey**, **securitySecretKey**, and **securityToken** parameters.

## Other Languages

**Java**, **Python**, **Go**, **Node.js**, **.Net**, and **PHP**.

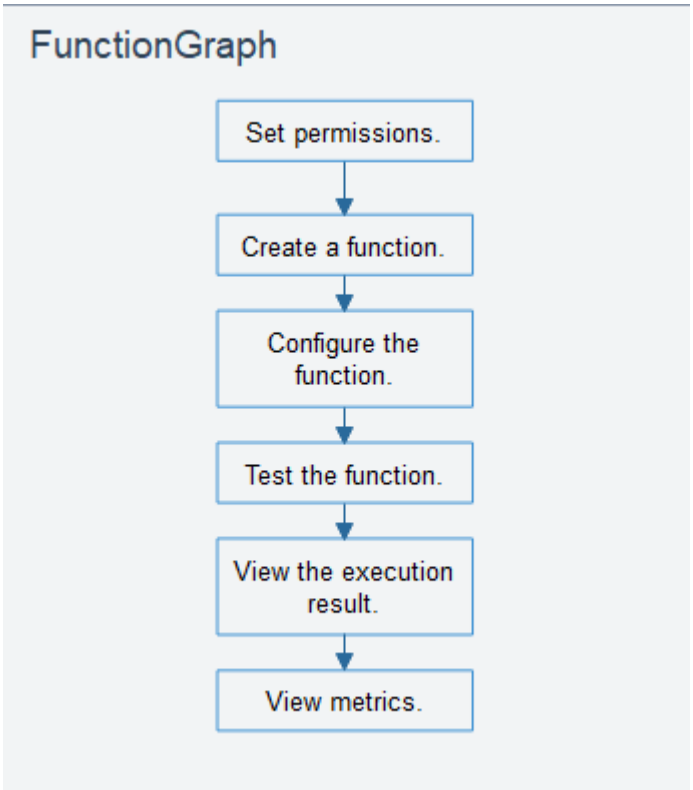
  

# 2 Before You Start

## 2.1 Use of 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:




1. Set permissions: Ensure that you have the **FunctionGraph Administrator** permissions.

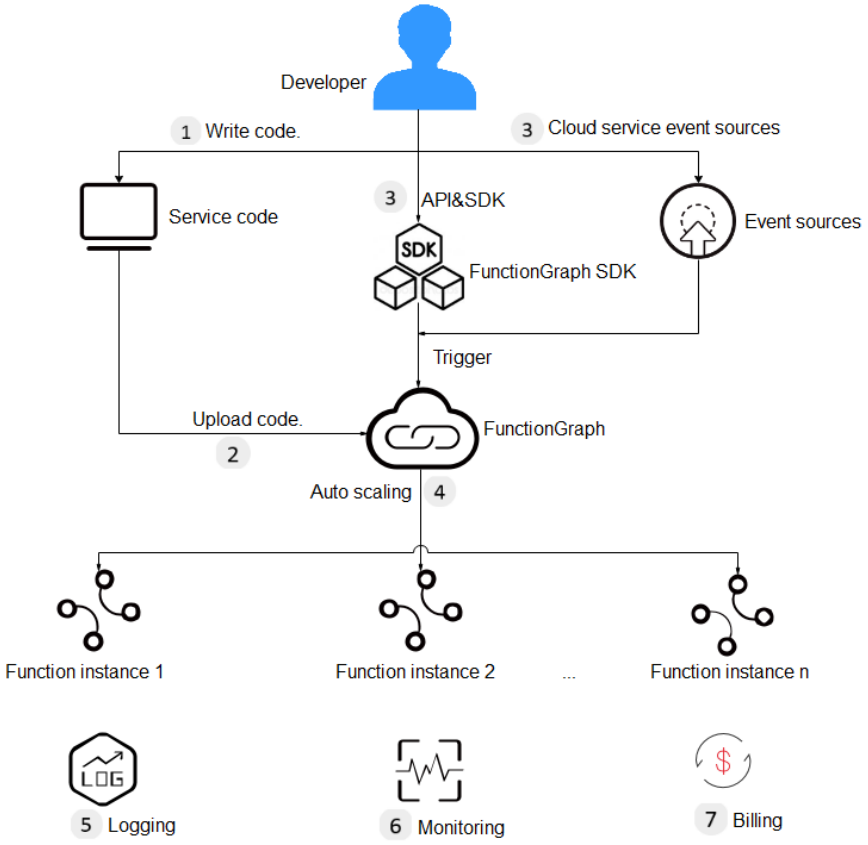
2. Create a function: Create a function from scratch or using the sample code or a container image.
3. Configure the function: Configure the code source or modify other parameters.
4. Test the function: Create a test event to debug the function.
5. View the execution result: On the function details page, view the execution result based on the configured test event.
6. View metrics: On the **Monitoring** tab page of the function details page, view function metrics.

## Process

**Figure 2-1** shows the process of using functions.

1. Write code, package and upload it to FunctionGraph, and add event sources such as Simple Message Notification (SMN) and API Gateway (APIG) event sources to build applications.
2. Functions are triggered by RESTful API calls or event sources to achieve expected service purposes. During this process, FunctionGraph automatically schedules resources.
3. View logs and metrics. Note that you will be billed based on code execution duration.

**Figure 2-1** Flowchart




The following shows the details:

1. Write code.

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

2. Upload code.

   Edit code inline, upload a local ZIP or JAR file, or upload a ZIP file from OBS. For details, see **Creating a Deployment Package**.

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

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

4. Implement auto scaling.

   FunctionGraph implements auto scaling based on the number of requests. For details, see **Notes and Constraints**.

5. View logs.

   View run logs of function. FunctionGraph is interconnected with Log Tank Service (LTS). For details, see **Logs**.

6. View monitoring information.

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

7. Check your bills.

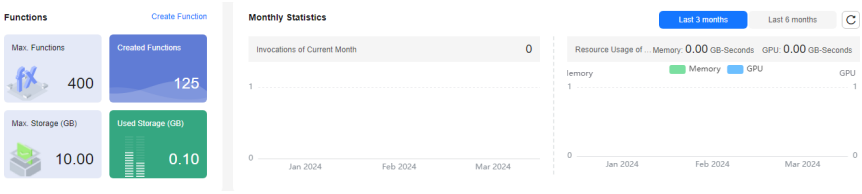
   After function execution is complete, you will be billed based on the number of function execution requests and execution duration. For details, see .**Bills**.

## 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 2-2** Monthly statistics

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

  **Table 2-1** describes the function metrics.

**Table 2-1** 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. |
| 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. |

| Metric | Unit | Description |
| --- | --- | --- |
| Running Workflows | Count | Number of ongoing flow executions. |

# 2.2 Permissions Management

## 2.2.1 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 other accounts or cloud services to perform professional and 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 2-3**.

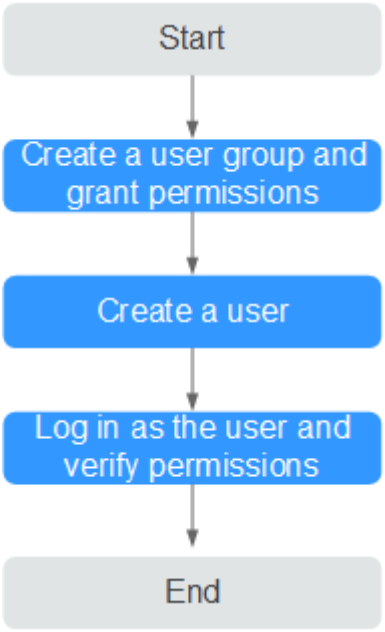
### Prerequisites

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 2-3** Process for granting FunctionGraph permissions




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

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

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

   Log in to the management console as the created user and check whether this user 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.

## 2.2.2 Creating a Custom Policy

Custom policies can be created as a supplement to the system policies of FunctionGraph.

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: Edit JSON policies from scratch or based on an existing policy.

For details, see **Creating a Custom Policy**. This section introduces examples of common FunctionGraph custom policies.

## Example 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",
            "funcitongraph:function:getCode"
         ]
      }
   ]
}
```

- Example 2: Denying function deletion

  A policy with only "Deny" permissions must be used in conjunction with other policies to take effect. 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**.

  NOTE

  Specify function resources:

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

  For function resources, IAM automatically generates the resource path prefix **FUNCTIONGRAPH:*:*:function:**. You can specify a resource path by adding the application or function name next to the path prefix. Wildcards (*) are 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/*"
            ]
        }
    ]
}
```

## 2.2.3 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 2-2** FunctionGraph resources and their paths

| Resource Type | Resource Name | Path |
| --- | --- | --- |
| function | Function | [Format]<br>FunctionGraph::::function:*group/function name*<br>[Notes]<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 | Trigger | [Format]<br>FunctionGraph::::trigger:*trigger ID*<br>[Notes]<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. |

# 2.3 Supported Programming Languages

## 2.3.1 Node.js

√: Supported. ×: Not supported.

| Runtime | Supported | Development Guide |
| --- | --- | --- |
| Node.js 6.10 | √ | For details about function syntax, SDK APIs, and function development, see **Developing Functions in Node.js**. |
| Node.js 8.10 | √ | |
| Node.js 10.16 | √ | |
| Node.js 12.13 | √ | |
| Node.js 14.18 | √ | |
| Node.js 16.17 | √ | |
| Node.js 18.15 | √ | |

## 2.3.2 Python

√: Supported. ×: Not supported.

| Runtime | Supported | Development Guide |
| --- | --- | --- |
| Python 2.7 | √ | For details about function syntax, SDK APIs, and function development, see **Developing Functions in Python**. |
| Python 3.6 | √ | |
| Python 3.9 | √ | |
| Python 3.10 | √ | |

## 2.3.3 Java

√: Supported. ×: Not supported.

| Runtime | Supported | Development Guide |
| --- | --- | --- |
| Java 8 | √ | For details about function syntax, SDK APIs, and function development, see **Developing Functions in Java**. |
| Java 11 | √ | |

## 2.3.4 Go

√: Supported. ×: Not supported.

| Runtime | Supported | Development Guide |
| --- | --- | --- |
| Go 1.x | √ | For details about function syntax, SDK APIs, and function development, see **Developing Functions in Go**. |

## 2.3.5 C#

√: Supported. ×: Not supported.

| Runtime | Supported | Development Guide |
| --- | --- | --- |
| C# (.NET Core 2.1) | √ | For details about function syntax, SDK APIs, and function development, see **Developing Functions in C#**. |
| C# (.NET Core 3.1) | √ | |
| C# (.NET Core 6.0) | √ (only in LA-Mexico City2) | |

## 2.3.6 PHP

√: Supported. ×: Not supported.

| Runtime | Supported | Development Guide |
| --- | --- | --- |
| PHP 7.3 | √ | For details about function syntax, SDK APIs, and function development, see **Developing Functions in PHP**. |

## 2.3.7 Custom Runtime

### Scenarios

A runtime runs the code of a function, reads the handler name from an environment variable, and reads invocation events from the runtime APIs of FunctionGraph. The runtime passes event data to the function handler and returns the response from the handler to FunctionGraph.

FunctionGraph supports custom runtimes. You can use an executable file named **bootstrap** to include a runtime in your function deployment package. The runtime runs the function's handler method when the function is invoked.

Your runtime runs in the FunctionGraph execution environment. It can be a shell script or a binary executable file that is compiled in Linux.

NOTE

After programming, simply package your code into a ZIP file (Java, Node.js, Python, and Go) or JAR file (Java), and upload the file to FunctionGraph for execution. When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.

If you edit code in Go, zip the compiled file, and 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**.

## Compiling Description

If you compile files on Windows and execute them on Linux, you need to configure the following parameters:

```
CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build main.go
```

## Runtime File bootstrap

If there is a file named **bootstrap** in your function deployment package, FunctionGraph executes that file. If the **bootstrap** file is not found or not executable, your function will return an error when invoked.

The runtime code is responsible for completing initialization tasks. It processes invocation events in a loop until it is terminated.

The initialization tasks run once for each instance of the function to prepare the environment for handling invocations.

## Runtime APIs

FunctionGraph provides HTTP runtime APIs to receive function invocation events and returns response data in the execution environment.

- **Obtaining Invocation Event**

  **Method** – Get

  **Path** – http://$RUNTIME_API_ADDR/v1/runtime/invocation/request

  This API is used to retrieve an invocation event. The response body contains the event data. The following table describes additional data about the invocation contained in the response header.

  **Table 2-3** Response header information

| Parameter | Description |
| --- | --- |
| X-Cff-Request-Id | Request ID. |
| X-CFF-Access-Key | AK of the account. An agency must be configured for the function if this variable is used. |
| X-CFF-Auth-Token | Token of the account. An agency must be configured for the function if this variable is used. |
| X-CFF-Invoke-Type | Invocation type of the function. |

| Parameter | Description |
| --- | --- |
| X-CFF-Secret-Key | SK of the account. An agency must be configured for the function if this variable is used. |
| X-CFF-Security-Token | Security token of the account. An agency must be configured for the function if this variable is used. |

- **Invocation Response**

  **Method** – POST

  **Path** – http://$RUNTIME_API_ADDR/v1/runtime/invocation/response/$REQUEST_ID

  This API is used to send a successful invocation response to FunctionGraph. After the runtime invokes the function handler, it publishes the response from the function to the invocation response path.

- **Invocation Error**

  **Method** – POST

  **Path** – http://$RUNTIME_API_ADDR/v1/runtime/invocation/error/$REQUEST_ID

  **$REQUEST_ID** is the value of variable **X-Cff-Request-Id** in the header of an event retrieval response. For more information, see **Table 2-3**.

  **$RUNTIME_API_ADDR** is a system environment variable. For more information, see **Table 2-4**.

  This API is used to send an error invocation response to FunctionGraph. After the runtime invokes the function handler, it publishes the response from the function to the invocation response path.

## Runtime Environment Variables

You can use both custom and runtime environment variables in function code. The following table lists the runtime environment variables that are used in the FunctionGraph execution environment.

**Table 2-4** Environment variables

| Key | Description |
| --- | --- |
| RUNTIME_PROJECT_ID | Project ID |
| RUNTIME_FUNC_NAME | Function name |
| RUNTIME_FUNC_VERSION | Function version |
| RUNTIME_PACKAGE | App to which the function belongs |
| RUNTIME_HANDLER | Function handler |
| RUNTIME_TIMEOUT | Function timeout duration |

| Key | Description |
| --- | --- |
| RUNTIME_USERDATA | Value passed through an environment variable |
| RUNTIME_CPU | Number of allocated CPU cores |
| RUNTIME_MEMORY | Allocated memory |
| RUNTIME_CODE_ROOT | Directory that stores the function code |
| RUNTIME_API_ADDR | Host IP address and port of a custom runtime API |

The value of a custom environment variable can be retrieved in the same way as the value of a FunctionGraph environment variable.

## Example

This example contains one file called **bootstrap**. The file is implemented in Bash.

The runtime loads the function script from the deployment package by using two variables.

The **bootstrap** file is as follows:

```
#!/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 "$REQUEST_ID" ]; then
    continue
  fi
  # Process request data
  RESPONSE="Echoing request: hello world!"
  # Put response
  curl -X POST "http://$RUNTIME_API_ADDR/v1/runtime/invocation/response/$REQUEST_ID" -d
"$RESPONSE"
done
```

After loading the script, the runtime processes invocation events in a loop until it is terminated. It uses the API to retrieve invocation events from FunctionGraph, passes the events to the handler, and then sends responses back to FunctionGraph.

To obtain the request ID, the runtime saves the API response header in a temporary file, and then reads the request ID from the **x-cff-request-id** header field. The runtime processes the retrieved event data and sends a response back to FunctionGraph.

The following is an example of source code in Go. It can be executed only after compilation.

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io"
    "io/ioutil"
    "log"
    "net"
    "net/http"
    "os"
    "strings"
    "time"
)

var (
    getRequestUrl           = os.ExpandEnv("http://${RUNTIME_API_ADDR}/v1/runtime/invocation/
request")
    putResponseUrl          = os.ExpandEnv("http://${RUNTIME_API_ADDR}/v1/runtime/invocation/
response/{REQUEST_ID}")
    putErrorResponseUrl     = os.ExpandEnv("http://${RUNTIME_API_ADDR}/v1/runtime/
invocation/error/{REQUEST_ID}")
    requestIdInvalidError   = fmt.Errorf("request id invalid")
    noRequestAvailableError = fmt.Errorf("no request available")
    putResponseFailedError  = fmt.Errorf("put response failed")
    functionPackage         = os.Getenv("RUNTIME_PACKAGE")
    functionName            = os.Getenv("RUNTIME_FUNC_NAME")
    functionVersion         = os.Getenv("RUNTIME_FUNC_VERSION")

    client = http.Client{
        Transport: &http.Transport{
            DialContext: (&net.Dialer{
                Timeout: 3 * time.Second,
            }).DialContext,
        },
    }
)

func main() {
    // main loop for processing requests.
    for {
        requestId, header, payload, err := getRequest()
        if err != nil {
            time.Sleep(50 * time.Millisecond)
            continue
        }

        result, err := processRequestEvent(requestId, header, payload)
        err = putResponse(requestId, result, err)
        if err != nil {
            log.Printf("put response failed, err: %s.", err.Error())
        }
    }
}

// event processing function
func processRequestEvent(requestId string, header http.Header, evtBytes []byte) ([]byte, error) {
    log.Printf("processing request '%s'.", requestId)
    result := fmt.Sprintf("function: %s:%s:%s, request id: %s, headers: %+v, payload: %s",
functionPackage, functionName,
        functionVersion, requestId, header, string(evtBytes))
```

```
    var event FunctionEvent
    err := json.Unmarshal(evtBytes, &event)
    if err != nil {
        return (&ErrorMessage{ErrorType: "invalid event", ErrorMessage: "invalid json formated
event"}).toJsonBytes(), err
    }

    return (&APIGFormatResult{StatusCode: 200, Body: result}).toJsonBytes(), nil
}

func getRequest() (string, http.Header, []byte, error) {
    resp, err := client.Get(getRequestUrl)
    if err != nil {
        log.Printf("get request error, err: %s.", err.Error())
        return "", nil, nil, err
    }
    defer resp.Body.Close()

    // get request id from response header
    requestId := resp.Header.Get("X-CFF-Request-Id")
    if requestId == "" {
        log.Printf("request id not found.")
        return "", nil, nil, requestIdInvalidError
    }

    payload, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Printf("read request body error, err: %s.", err.Error())
        return "", nil, nil, err
    }

    if resp.StatusCode != 200 {
        log.Printf("get request failed, status: %d, message: %s.", resp.StatusCode, string(payload))
        return "", nil, nil, noRequestAvailableError
    }

    log.Printf("get request ok.")
    return requestId, resp.Header, payload, nil
}

func putResponse(requestId string, payload []byte, err error) error {
    var body io.Reader
    if payload != nil && len(payload) > 0 {
        body = bytes.NewBuffer(payload)
    }

    url := ""
    if err == nil {
        url = strings.Replace(putResponseUrl, "{REQUEST_ID}", requestId, -1)
    } else {
        url = strings.Replace(putErrorResponseUrl, "{REQUEST_ID}", requestId, -1)
    }

    resp, err := client.Post(strings.Replace(url, "{REQUEST_ID}", requestId, -1), "", body)
    if err != nil {
        log.Printf("put response error, err: %s.", err.Error())
        return err
    }
    defer resp.Body.Close()

    responsePayload, err := ioutil.ReadAll(resp.Body)
```

```
    if err != nil {
        log.Printf("read request body error, err: %s.", err.Error())
        return err
    }

    if resp.StatusCode != 200 {
        log.Printf("put response failed, status: %d, message: %s.", resp.StatusCode,
string(responsePayload))
        return putResponseFailedError
    }

    return nil
}

type FunctionEvent struct {
    Type string `json:"type"`
    Name string `json:"name"`
}

type APIGFormatResult struct {
    StatusCode      int               `json:"statusCode"`
    IsBase64Encoded bool              `json:"isBase64Encoded"`
    Headers         map[string]string `json:"headers,omitempty"`
    Body            string            `json:"body,omitempty"`
}

func (result *APIGFormatResult) toJsonBytes() []byte {
    data, err := json.MarshalIndent(result, "", "  ")
    if err != nil {
        return nil
    }

    return data
}

type ErrorMessage struct {
    ErrorType     string `json:"errorType"`
    ErrorMessage  string `json:"errorMessage"`
}

func (errMsg *ErrorMessage) toJsonBytes() []byte {
    data, err := json.MarshalIndent(errMsg, "", "  ")
    if err != nil {
        return nil
    }

    return data
}
```

**Table 2-5** describes the environment variables used in the preceding code.

**Table 2-5** Environment variables

| Environment Variable | Description |
| --- | --- |
| RUNTIME_FUNC_NAME | Function name |
| RUNTIME_FUNC_VERSION | Function version |
| RUNTIME_PACKAGE | App to which the function belongs |

### Streaming Response

Custom runtime supports response of large packets whose size ranges from 6 MB to 200 MB in data streams.

**Step 1** Log in to the FunctionGraph console, and click the name of the custom runtime function to go to the details page.

**Step 2** Choose **Configuration** > **Advanced Settings** and enable **Streaming Response**.

**----End**

## 2.3.8 Cangjie

√: Supported. ×: Not supported.

| Runtime | Supported |
| --- | --- |
| Cangjie 1.0 | √ |

# 3 Building Functions

## 3.1 Creating a Deployment Package

To create a function, you must create a deployment package which includes your code and all dependencies. You can create a deployment package locally or edit code on the FunctionGraph console. If you edit code inline, FunctionGraph automatically creates and uploads a deployment package for your function. FunctionGraph allows you to edit function code in the same way as managing a project. You can create and edit files and folders. After you upload a ZIP code package, you can view and edit the code on the console.

NOTE

- After programming, simply package your code into a ZIP file (Java, Node.js, Python, and Go) or JAR file (Java), and upload the file to FunctionGraph for execution.
- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- If you edit code in Go, zip the compiled file, and 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**.
- Java is a compiled language, which does not support editing code inline. If your function does not use any third-party dependencies, you can upload a function JAR file. If your function uses third-party dependencies, compress the dependencies and the function JAR file into a ZIP file, and then upload the ZIP file.

Table 3-1 lists the code entry modes supported by FunctionGraph for each runtime.

**Table 3-1** Code entry modes

| Runtime | Editing Code Inline | Uploading a ZIP File | Uploading a JAR File | Uploading a ZIP File from OBS |
| --- | --- | --- | --- | --- |
| Node.js | Supported | Supported | Not supported | Supported |

| Runtime | Editing Code Inline | Uploading a ZIP File | Uploading a JAR File | Uploading a ZIP File from OBS |
|---|---|---|---|---|
| Python | Supported | Supported | Not supported | Supported |
| Java | Not supported | Supported | Supported | Supported |
| Go | Not supported | Supported | Not supported | Supported |
| C# | Not supported | Supported | Not supported | Supported |
| PHP | Supported | Supported | Not supported | Supported |
| Custom runtime | Supported | Supported | Not supported | Supported |

**NOTICE**

If the code to be uploaded contains sensitive information (such as account passwords), encrypt the sensitive information to prevent leakage.

**Table 3-2** Code entry modes

| Code Entry Mode | Description |
|---|---|
| Edit code inline | FunctionGraph allows you to edit function code in the same way as managing a project. You can create and edit files and folders. After you upload a ZIP code package, you can edit the code on the **Code** tab of the function details page.<br>• **File**: Create files and folders, save changes, and close all files.<br>• **Edit**: Undo/redo typing; cut, copy, and paste code; find and replace content.<br>• **Settings**: Set the font size, auto formatting, and theme color. |
| Upload ZIP file | 1. On the **Code** tab of the function details page, choose **Upload** > **Local ZIP**.<br>2. Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file cannot exceed 40 MB. For a larger file, upload it through OBS. |

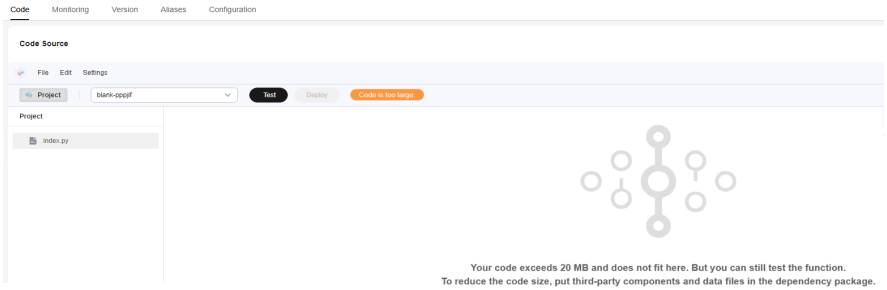
  

| Code Entry Mode | Description |
| --- | --- |
| Upload file from OBS | 1. On the **Code** tab of the function details page, choose **Upload** > **OBS ZIP**.<br>2. Click **Select File** and upload a local code package to FunctionGraph. |

NOTE

If the size of the code you deploy is greater than 20 MB, the inline editor does not display the code, but you can still test your function.

**Figure 3-1** Code not displayed in the editor




## Node.js

### Editing Code Inline

FunctionGraph provides an SDK for editing code in Node.js. If your custom code uses only the SDK library, you can edit code using the inline editor on the FunctionGraph console. After you edit code inline and upload it to FunctionGraph, the console compresses your code and the related configurations into a deployment package that FunctionGraph can run.

### Uploading a Deployment Package

If your code uses other resources, such as a graphic library for image processing, first create a deployment package, and then upload the package to the FunctionGraph console. You can upload a Node.js deployment package in two ways.

NOTICE

- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- The size of the decompressed source code cannot exceed 1.5 GB. If the code is too large, contact the customer service.

- Directly uploading a local deployment package

After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 40 MB, upload the package from OBS.

For details about function resource restrictions, see **Notes and Constraints**.

- Uploading a deployment package using an OBS bucket

  After creating a ZIP deployment package, upload it to an OBS bucket in the same region as your FunctionGraph, and then paste the link URL of the OBS bucket into the function. The maximum size of the ZIP file that can be uploaded to OBS is 300 MB.

  For details about function resource restrictions, see **Notes and Constraints**.

## Python

### Editing Code Inline

FunctionGraph provides an SDK for editing code in Python. If your custom code uses only the SDK library, you can edit code using the inline editor on the FunctionGraph console. After you edit code inline and upload it to FunctionGraph, the console compresses your code and the related configurations into a deployment package that FunctionGraph can run.

### Uploading a Deployment Package

If your code uses other resources, such as a graphic library for image processing, first create a deployment package, and then upload the package to the FunctionGraph console. You can upload a Python deployment package in two ways.

**NOTICE**

- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- The size of the decompressed source code cannot exceed 1.5 GB. If the code is too large, contact the customer service.
- When you write code in Python, do not name your 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.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 40 MB, upload the package from OBS.

  For details about function resource restrictions, see **Notes and Constraints**.

- Uploading a deployment package using an OBS bucket

  After creating a ZIP deployment package, upload it to an OBS bucket in the same region as your FunctionGraph, and then paste the link URL of the OBS bucket into the function. The maximum size of the ZIP file that can be uploaded to OBS is 300 MB.

  For details about function resource restrictions, see **Notes and Constraints**.

## Java

Java is a compiled language, which does not support editing code inline. You can only upload a local deployment package, which can be a ZIP or JAR file.

**Uploading a JAR File**

- If your function does not use any dependencies, directly upload a JAR file.
- If your function uses dependencies, upload them to an OBS bucket, set them during function creation, and upload the JAR file.

**Uploading a ZIP File**

If your function uses third-party dependencies, compress the dependencies and the function JAR file into a ZIP file, and then upload the ZIP file.

You can upload a Java deployment package in two ways.

**NOTICE**

- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- The size of the decompressed source code cannot exceed 1.5 GB. If the code is too large, contact the customer service.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 40 MB, upload the package from OBS.

  For details about function resource restrictions, see **Notes and Constraints**.
- Uploading a deployment package using an OBS bucket

  After creating a ZIP deployment package, upload it to an OBS bucket in the same region as your FunctionGraph, and then paste the link URL of the OBS bucket into the function. The maximum size of the ZIP file that can be uploaded to OBS is 300 MB.

  For details about function resource restrictions, see **Notes and Constraints**.

## Go

**Uploading a Deployment Package**

You can only upload a Go deployment package in ZIP format. There are two ways to upload it.

**NOTICE**

- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- The size of the decompressed source code cannot exceed 1.5 GB. If the code is too large, contact the customer service.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 40 MB, upload the package from OBS.

  For details about function resource restrictions, see **Notes and Constraints**.

- Uploading a deployment package using an OBS bucket

  After creating a ZIP deployment package, upload it to an OBS bucket in the same region as your FunctionGraph, and then paste the link URL of the OBS bucket into the function. The maximum size of the ZIP file that can be uploaded to OBS is 300 MB.

  For details about function resource restrictions, see **Notes and Constraints**.

## C#

### Uploading a Deployment Package

You can only upload a C# deployment package in ZIP format. There are two ways to upload it.

**NOTICE**

- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- The size of the decompressed source code cannot exceed 1.5 GB. If the code is too large, contact the customer service.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 40 MB, upload the package from OBS.

  For details about function resource restrictions, see **Notes and Constraints**.

- Uploading a deployment package using an OBS bucket

  After creating a ZIP deployment package, upload it to an OBS bucket in the same region as your FunctionGraph, and then paste the link URL of the OBS bucket into the function. The maximum size of the ZIP file that can be uploaded to OBS is 300 MB.

  For details about function resource restrictions, see **Notes and Constraints**.

## PHP

### Editing Code Inline

FunctionGraph provides an SDK for editing code in PHP. If your custom code uses only the SDK library, you can edit code using the inline editor on the FunctionGraph console. After you edit code inline and upload it to FunctionGraph, the console compresses your code and the related configurations into a deployment package that FunctionGraph can run.

### Uploading a Deployment Package

If your code uses other resources, such as a graphic library for image processing, first create a deployment package, and then upload the package to the FunctionGraph console. You can upload a PHP deployment package in two ways.

NOTICE

- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- The size of the decompressed source code cannot exceed 1.5 GB. If the code is too large, contact the customer service.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 40 MB, upload the package from OBS.

  For details about function resource restrictions, see **Notes and Constraints**.
- Uploading a deployment package using an OBS bucket

  After creating a ZIP deployment package, upload it to an OBS bucket in the same region as your FunctionGraph, and then paste the link URL of the OBS bucket into the function. The maximum size of the ZIP file that can be uploaded to OBS is 300 MB.

  For details about function resource restrictions, see **Notes and Constraints**.

## Custom Runtime

### Editing Code Inline

After you edit code inline and upload it to FunctionGraph, the console compresses your code and the related configurations into a deployment package that FunctionGraph can run.

### Uploading a Deployment Package

If your code uses other resources, such as a graphic library for image processing, first create a deployment package, and then upload the package to the FunctionGraph console. You can upload a deployment package for a custom runtime in two ways.

NOTICE

- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- The size of the decompressed source code cannot exceed 1.5 GB. If the code is too large, contact the customer service.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 40 MB, upload the package from OBS.

  For details about function resource restrictions, see **Notes and Constraints**.

- Uploading a deployment package using an OBS bucket

  After creating a ZIP deployment package, upload it to an OBS bucket in the same region as your FunctionGraph, and then paste the link URL of the OBS bucket into the function. The maximum size of the ZIP file that can be uploaded to OBS is 300 MB.

  For details about function resource restrictions, see **Notes and Constraints**.

# 3.2 Creating a Function from Scratch

## 3.2.1 Creating an Event Function

### Overview

A function is customized code for processing events. You can create a function from scratch and configure the function based on site requirements.

FunctionGraph manages the compute resources required for function execution. After editing code for your function, configure compute resources on the FunctionGraph console.

You can create a function from scratch or by using **a template** or **container image**.

NOTE

When creating a function from scratch, configure the basic and code information based on **Table 3-3**. The parameters marked with an asterisk (*) are mandatory.

Each FunctionGraph function runs in its own environment and has its own resources and file system.

### Prerequisites

1. You must be familiar with the programming languages supported by FunctionGraph. For details, see **Supported Programming Languages**.
2. You have created a deployment package. For details, see **Creating a Deployment Package**.
3. (Optional) You have created an agency. For details, see **Configuring Agency Permissions**.

### Procedure

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. Click **Create from scratch** and configure the function information by referring to **Table 3-3**. The parameters marked with an asterisk (*) are mandatory.

**Figure 3-2** Creating a function from scratch

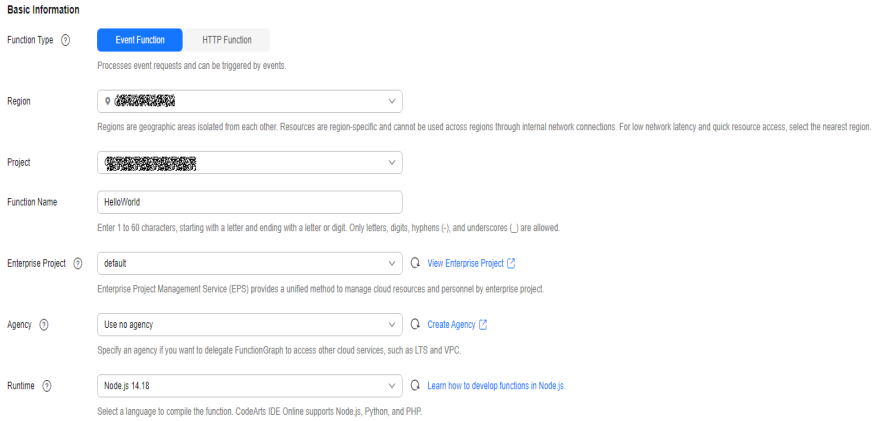



**Table 3-3** Basic information

| Parameter | Description |
| --- | --- |
| * Function Type | • Event functions: triggered by triggers.<br>• HTTP functions: triggered once HTTP requests are sent to specific URLs.<br>**NOTE**<br>• 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.<br>• HTTP functions support APIG and APIC triggers only.<br>• For details about how to use HTTP functions, see **Creating an HTTP Function**. |
| *Region | Select a region where you will deploy your code. |
| *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. |
| Agency | An agency is required if FunctionGraph accesses other cloud services. For details on how to create an agency, see **Configuring Agency Permissions**.<br>No agency is required if FunctionGraph does not access any cloud services. |
| *Enterprise Project | Select a created enterprise project and add the function to it. By default, **default** is selected.<br>**NOTE**<br>If Enterprise Project Management Service (EPS) is not enabled, this parameter is unavailable. For details, see **Enabling the Enterprise Project Function**. |

| Parameter | Description |
| --- | --- |
| Runtime | Select a runtime to compile the function.<br>**NOTICE**<br>CloudIDE supports Node.js, Python, and PHP only. |

4. **Advanced Settings**: Set **Log Tag**. A maximum of 10 tags can be added. You can use these tags to filter function logs in LTS.

   **Table 3-4** Log tag parameters

| | |
| --- | --- |
| **Tag key** | Enter a maximum of 64 characters, including only digits, letters, underscores (_), and hyphens (-). |
| **Tag value** | |

5. Click **Create Function**. On the displayed **Code** tab page, continue to configure the code.

## Configuring Code

1. You can deploy the code based on the runtime you select. For details, see **Creating a Deployment Package**. After the deployment is complete, click **Deploy**.

   As shown in the following example, to deploy code in Node.js 10.16, you can edit code inline, upload a local ZIP file, or upload a ZIP file from OBS.

   **Figure 3-3** Deploying code


   **NOTE**

   If you deploy code by uploading a local ZIP file, ensure that the region selected for creating the function is the same as that selected for creating an OBS bucket.

2. You can modify the code and click **Deploy** to deploy the code again.

## Viewing Code Information

1. View code attributes.

   Code attributes show the code size and the time the code was modified.

   **Figure 3-4** Viewing code attributes


2. View basic information.

**Configuring Basic Settings** shows the default memory and execution timeout in each runtime. You can click **Edit** to switch to the **Basic Settings** page and modify **Handler**, **Memory (MB)**, and **Execution Timeout (s)** as required. For details, see **Figure 3-5**.

**Figure 3-5** Editing basic information


**NOTICE**

Once a function is created, the runtime cannot be changed.

**Table 3-5** Default basic information of each runtime

| Runtime | Default Basic Information |
| --- | --- |
| Java | Memory (MB): 512<br>Handler: com.demo.TriggerTests.apigTest<br>Execution Timeout (s): 15 |
| Node.js | Memory (MB): 128<br>Handler: index.handler<br>Execution Timeout (s): 3 |
| Custom | Memory (MB): 128<br>Handler: bootstrap<br>Execution Timeout (s): 3 |
| PHP | Memory (MB): 128<br>Handler: index.handler<br>Execution Timeout (s): 3 |
| Python | Memory (MB): 128<br>Handler: index.handler<br>Execution Timeout (s): 3 |
| Go 1.x | Memory (MB): 128<br>Handler: handler<br>Execution Timeout (s): 3 |

## 3.2.2 Creating an HTTP Function

### Overview

HTTP functions are designed to optimize web services. You can send HTTP requests to URLs to trigger function execution. HTTP functions support APIG and APIC triggers only.

NOTE

- 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**.
- 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**.
- HTTP functions support multiple programming languages.
- Functions must return a valid HTTP response.
- This section uses Node.js as an example. To use another runtime, simply change the runtime path. The code package path does not need to be changed. For the paths of other runtimes, see **Table 3-6**.
- For details about how to build a FunctionGraph HTTP function using Go, see **Building a FunctionGraph HTTP Function Using Go**.
- When a function initiates an HTTP request, the request IP address is dynamic for private network access and fixed for public network access. For more information, contact technical support.

## Prerequisites

CAUTION

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 **VPC Peering Connection**.
- 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 **Network Communications Among VPCs Across Regions**.
- 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.

1. Prepare a Node.js script. A code example is as follows:

```
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..')
```

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

   **Example**

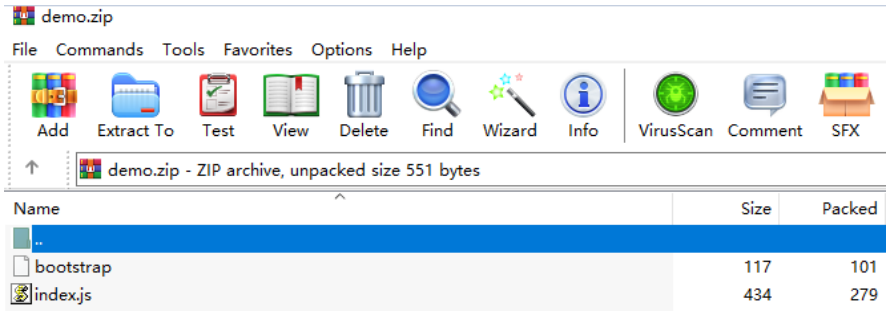
   The content of the **bootstrap** file is as follows:

```
/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 3-6** Compressing files into a ZIP package




**NOTE**

For HTTP functions in Python, add the **-u** parameter in the **bootstrap** file to ensure that logs can be flushed to the disk. 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 3-6**. The code package path does not need to be changed.

**Table 3-6** Paths for different runtimes

| Runtime | Path |
| --- | --- |
| Java 8 | /opt/function/runtime/java8/rtsp/jre/bin/java |
| Java 11 | /opt/function/runtime/java11/rtsp/jre/bin/java |
| Node.js 6 | /opt/function/runtime/nodejs6.10/rtsp/nodejs/bin/node |
| Node.js 8 | /opt/function/runtime/nodejs8.10/rtsp/nodejs/bin/node |
| Node.js 10 | /opt/function/runtime/nodejs10.16/rtsp/nodejs/bin/node |
| Node.js 12 | /opt/function/runtime/nodejs12.13/rtsp/nodejs/bin/node |
| Node.js 14 | /opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node |
| Node.js 16 | /opt/function/runtime/nodejs16.17/rtsp/nodejs/bin/node |
| Node.js 18 | /opt/function/runtime/nodejs18.15/rtsp/nodejs/bin/node |
| Python 2.7 | /opt/function/runtime/python2.7/rtsp/python/bin/python |
| Python 3.6 | /opt/function/runtime/python3.6/rtsp/python/bin/python3 |

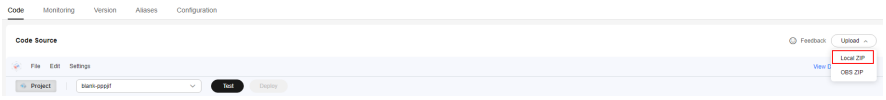
  

| Runtime | Path |
| --- | --- |
| Python 3.9 | /opt/function/runtime/python3.9/rtsp/python/bin/python3 |
| Python3.10 | /opt/function/runtime/python3.10/rtsp/python/bin/python3 |
| PHP 7.3 | /opt/function/runtime/php7.3/rtsp/php/bin/php |
| C# (.NET Core 2.1) | /opt/function/runtime/dotnet2.1/rtsp/dotnet/dotnet |
| C# (.NET Core 3.1) | /opt/function/runtime/dotnet3.1/rtsp/dotnet/dotnet |

## Procedure

1. Create a function.
   a. Create an HTTP function. For details, see **Creating an Event Function**. Pay special attention to the following parameters:
      - **Function Type**: HTTP function
      - **Region**: Select a region where you will deploy your code.
   b. Choose **Upload** > **Local ZIP**, upload the ZIP package, and click **Deploy**.

      **Figure 3-7** Uploading a ZIP file

      NOTE

      If you deploy code by uploading a local ZIP file, ensure that the region selected for creating the function is the same as that selected for creating an OBS bucket.
2. Create a trigger.

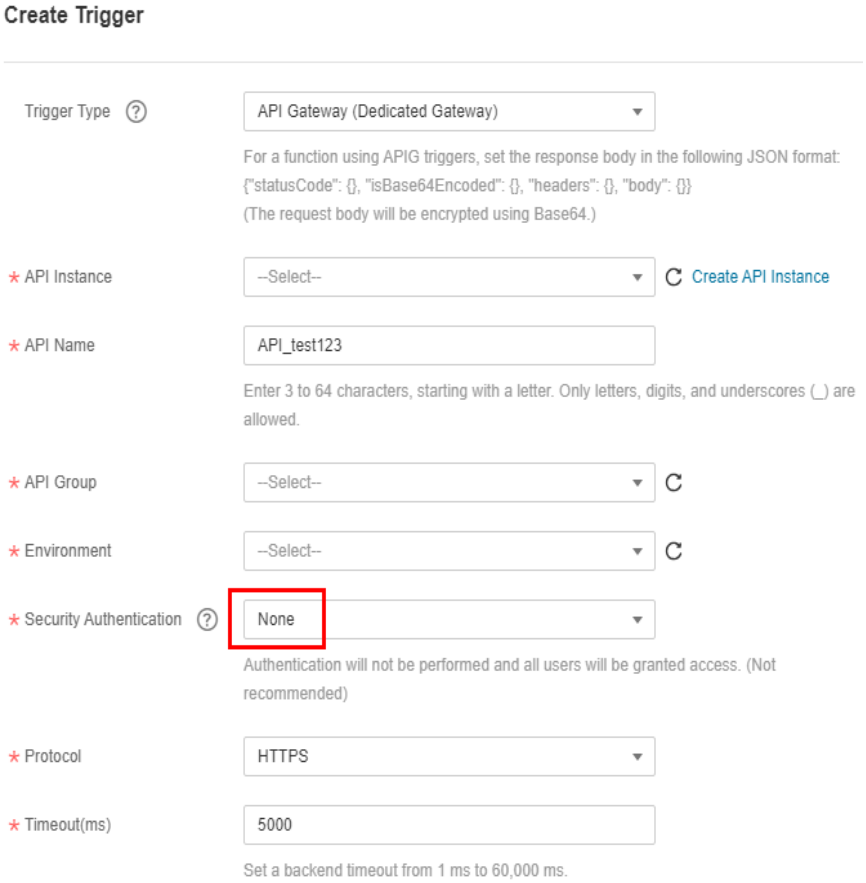
   NOTE

   HTTP functions support APIG and APIC triggers only.

   a. On the function details page, choose **Configuration** > **Triggers** and click **Create Trigger**.
   b. Set the trigger information. This step uses an APIG trigger as an example. For more information, see **Using an APIG Trigger**.

**Figure 3-8** Creating a trigger




**NOTE**

In this example, **Security Authentication** is set to **None**. You need to select an authentication mode based on site requirements.

- **App**: AppKey and AppSecret authentication. This mode is of high security and is recommended.
- **IAM**: IAM authentication. This mode grants access permissions to IAM users only and is of medium security.
- **None**: No authentication. This mode grants access permissions to all users.

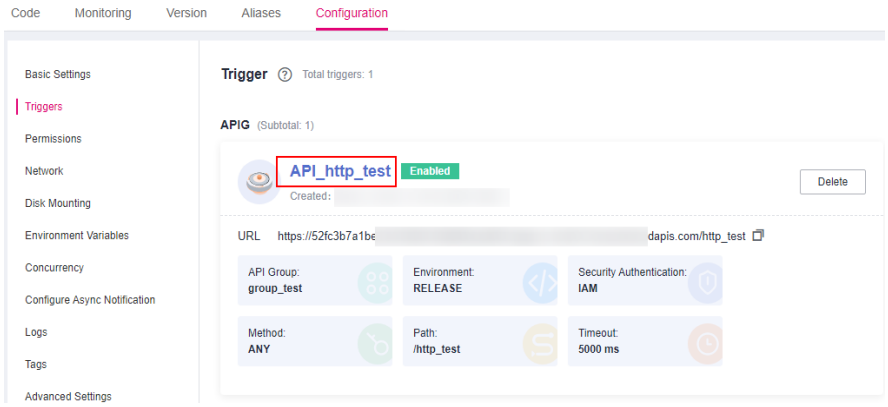
c. When the configuration is complete, click **OK**. After the trigger is created, **API_test_http** will be generated on the APIG console.

3. Publish the API.

a. On the **Triggers** tab page, click an API name to go to the API overview page.

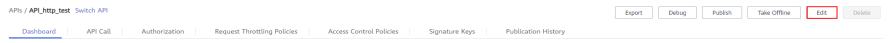
  

**Figure 3-9** API trigger



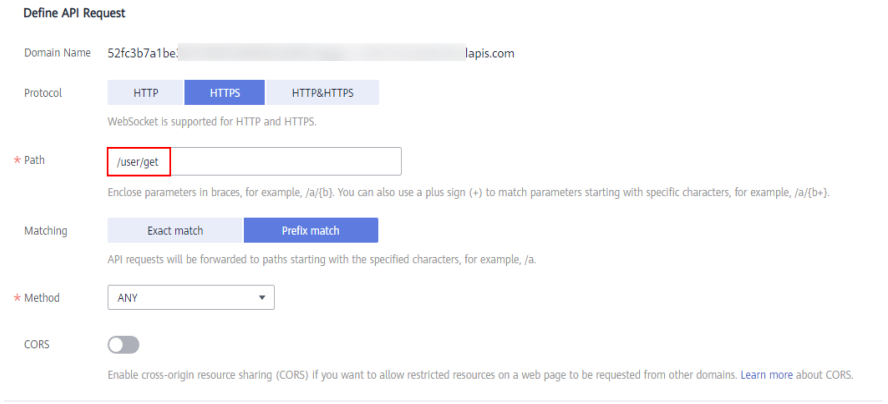

b. Click **Edit** in the upper right corner. The **Basic Information** page is displayed.

**Figure 3-10** Editing an API



c. Click **Next**. On the **Define API Request** page that is displayed, change **Path** to **/user/get** and click **Finish**.

**Figure 3-11** Defining an API request




d. Click **Publish API**. On the displayed page, click **Publish**.

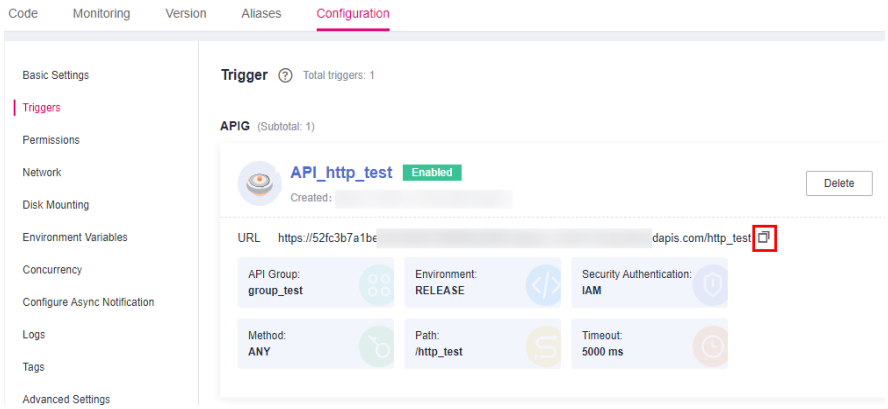
4. Trigger a function.

   a. 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.

   b. Choose **Configuration** > **Triggers**, copy the URL, and access it using a browser.

**Figure 3-12** Copying the URL




c. View the request result.

**Figure 3-13** Viewing the request result

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

## Common Function Request Headers

The following table lists the default request header fields of an HTTP function.

**Table 3-7** Default request header fields

| Field | Description |
| --- | --- |
| X-CFF-Request-Id | ID of the current request |
| X-CFF-Memory | Allocated memory |
| X-CFF-Timeout | Function timeout duration |
| X-CFF-Func-Version | Function version |
| X-CFF-Func-Name | Function name |
| X-CFF-Project-Id | Project ID |
| X-CFF-Package | App to which the function belongs |
| X-CFF-Region | Current region |

# 3.3 Creating a Function Using a Template

## Overview

FunctionGraph provides templates to automatically complete code, and running environment configurations when you create a function, helping you quickly build applications.

## Creating a Function

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Templates**.
2. On the page that is displayed, select the **FunctionGraph** service, select the **context-class-introduction** template for Python 2.7, and click **Configure**.

   NOTE

   The **context-class-introduction** template for Python 2.7 is used as an example. You can also select other templates.
3. After you select a function template, the built-in code and configurations of the template are automatically loaded. The **Create Function** page is displayed.
4. Set **Function Name** to **context**, select a created agency, retain default values for other parameters, and click **Create Function**.

   NOTE

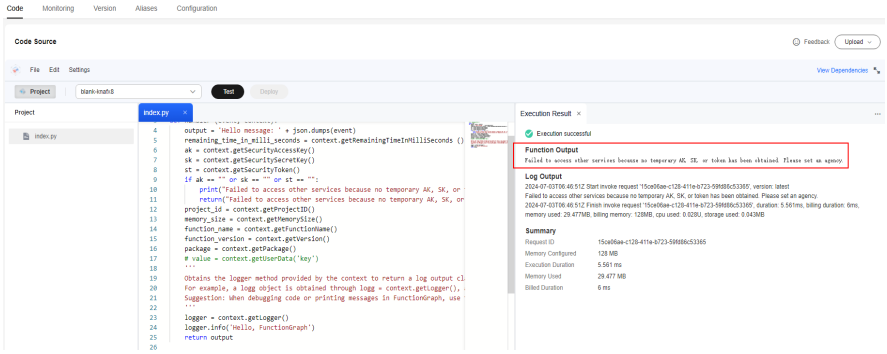
   If no agency is configured, the following message will be displayed when the function is triggered:

   ```
   Failed to access other services because no temporary AK, SK, or token has been obtained. Please set an agency.
   ```
5. Set the parameters based for your service requirements.

## Triggering a Function

1. On the **Code** tab page of the **context** function, click **Test** in the upper right corner.
2. In the **Configure Test Event** dialog box, select **Blank Template** and click **Create**.
3. Click **Test**. After the test is complete, view the test result.

**Figure 3-14** Successful execution result




# 3.4 Deploying a Function Using a Container Image

## Introduction

Package your container images complying with the Open Container Initiative (OCI) standard, and upload them to FunctionGraph. The images will be loaded

and run by FunctionGraph. Unlike the code upload mode, you can use a custom code package, which is flexible and reduces migration costs. You can create event and HTTP functions by using a custom image.

For details about how to develop and deploy an HTTP function using a container image, see **Developing an HTTP Function**.

For details about how to develop and deploy an event function using a container image, see **Developing an Event Function**.

The following features are supported:

- **Downloading images**

  Images are stored in SoftWare Repository for Container (SWR) and can only be downloaded by users with the **SWR Admin** permission. FunctionGraph will call the SWR API to generate and set temporary login commands before creating instances.

- **Setting environment variables**

  Encryption settings and environment variables are supported. For details, see **Configuring Environment Variables**.

- **Attaching external data disks**

  External data disks can be attached. For details, see **Configuring Disk Mounting**.

- **Billing**

  You will not be billed when you download images or wait for the images to be ready.

- **Reserved instances**

  For details, see the description about reserved instances.

- **Streaming response**

  Returns large packets whose size ranges from 6 MB to 200 MB in data streams.

- **Deploying a new image**

  You can redeploy a new image in a function.

NOTE

User containers will be started using UID 1003 and GID 1003, which are the same as other types of functions.

## Prerequisites

You have created an agency with the **SWR Admin** permission by referring to **Configuring Agency Permissions**. Images are stored in SWR, and only users with this permission can invoke and pull images.

## Procedure

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 details, see **Table 3-8**.

**Figure 3-15** Creating a function using a container image

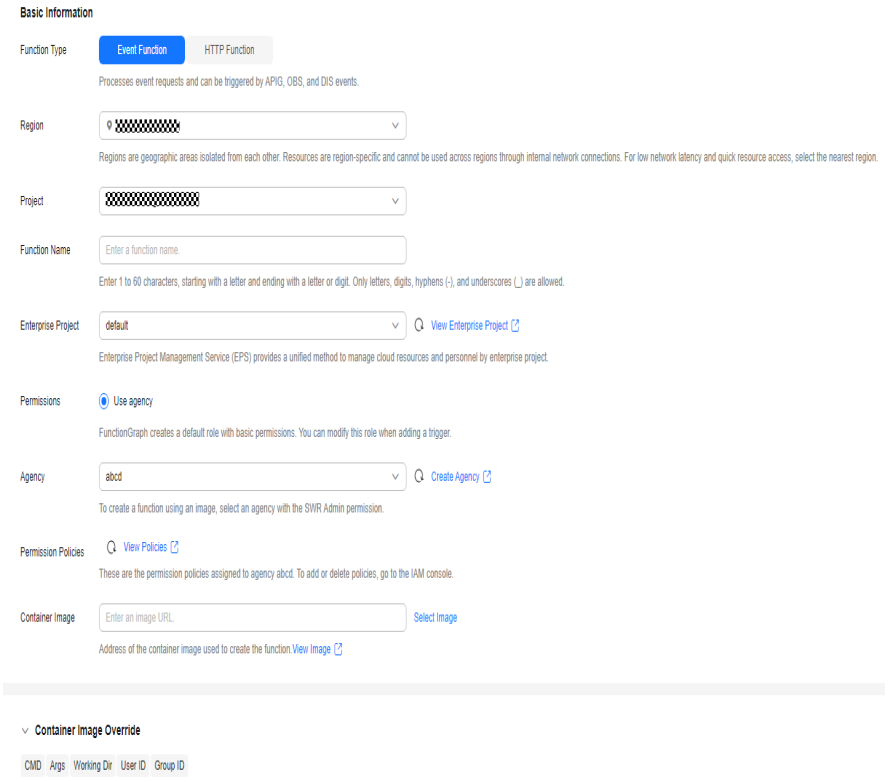



**Table 3-8** Parameter description

| Parameter | Description |
| --- | --- |
| *Function Type | Select a function type.<br>**Event function**: triggered by triggers.<br>**HTTP function**: triggered once HTTP requests are sent to specific URLs.<br>**NOTE**<br>• The custom container image must contain an HTTP server with listening port 8000.<br>• HTTP functions support APIG and APIC triggers only.<br>• 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>• 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.<br>• The function must return characters strings by using the following structure.<br>{<br>"isBase64Encoded": true\|false,<br>"statusCode": httpStatusCode,<br>"headers": {"headerName":"headerValue",...},<br>"body": "..."<br>} |

| Parameter | Description |
| --- | --- |
| *Region | Select a region where you will deploy your code. |
| *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. |
| *Enterprise Project | Select a created enterprise project and add the function to it. By default, **default** is selected.<br>**NOTE**<br>If EPS is not enabled, this parameter is unavailable. For details, see **Enabling the Enterprise Project Function**. |
| 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. 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**. |
| Container Image Override | • **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 of the container. The folder path can only be **/** and cannot be created or modified. The path will be **/** by default if not specified.<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. |
| Agency | Select an agency with the **SWR Admin** permission. To create an agency, see **Creating an Agency**. |

NOTE

- **Command**, **Args**, and **Working dir** can contain up to 5120 characters.
- 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.
- Public and private images are supported. For details, see **Setting Image Attributes**.
- The port of a custom container image must be 8000.
- 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.
- FunctionGraph uses LTS to collect all logs that the container outputs to the console. These logs can be redirected to and printed on the console through standard output or an open-source log framework. The logs should include the system time, component name, code line, and key data, to facilitate fault locating.
- When an out of memory (OOM) error occurs, view the memory usage in the function execution result.
- Functions must return a valid HTTP response.

4. **Advanced Settings**: Set **Log Tag**. A maximum of 10 tags can be added. You can use these tags to filter function logs in LTS.

   **Table 3-9** Log tag parameters

   | | |
   |---|---|
   | **Tag key** | Enter a maximum of 64 characters, including only digits, letters, underscores (_), and hyphens (-). |
   | **Tag value** | |

5. (Optional) Enable **Streaming Response**.

   After the container image function is created, choose **Configuration** > **Advanced Settings** and enable **Streaming Response**.

6. (Optional) Deploy a new image.

   On the **Code** tab, click **Deploy Image** on the right, enter the URL of the new image in the text box, and click **OK**. To obtain the URL, perform the following operations:

   a. Log in to the SWR console. In the navigation pane, choose **My Images**.

   b. Click the **Private Images** or **Images From Others** tab. In the image list, click the image name to go to the details page.

   c. 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.

## Sample Code

The following uses **Node.js Express** as an example. During function initialization, FunctionGraph uses the POST method to access the **/init** path (optional). Each time when a function is called, FunctionGraph uses the POST method to access the **/invoke** path. The function obtains **context** from **req.headers**, obtains **event** from **req.body**, and returns an HTTP response struct.

```
const express = require('express');
const app = express();
```

```
const PORT = 8000;

app.post('/init', (req, res) => {
  res.send('Hello init\n');
});

app.post('/invoke', (req, res) => {
  res.send('Hello invoke\n');
});

app.listen(PORT, () => {
  console.log(`Listening on http://localhost:${PORT}`);
});
```

# 3.5 Deploying a Function Using Terraform

## 3.5.1 Introduction

This section describes how to create a function using Terraform.

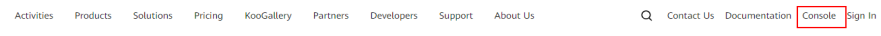
## 3.5.2 Prerequisites

### 3.5.2.1 Obtaining an Access Key

An access key comprises an access key ID (AK) and secret access key (SK), and is your long-term credential for accessing Huawei Cloud APIs.

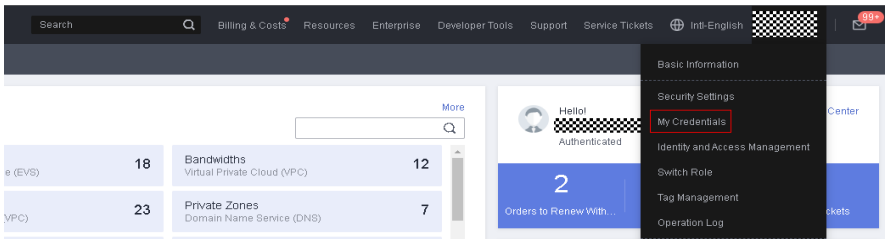
Log in to Huawei Cloud, and click **Console** in the upper right corner.

**Figure 3-16** Console



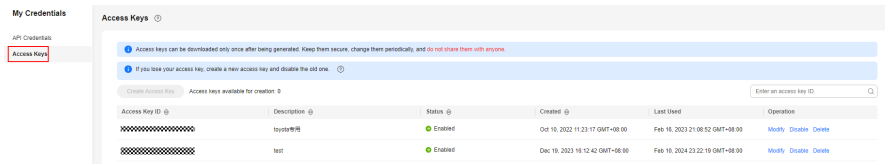

On the management console, hover over the username in the upper right corner and choose **My Credentials** from the drop-down list.

**Figure 3-17** My Credentials




Choose **Access Keys** from the navigation pane.

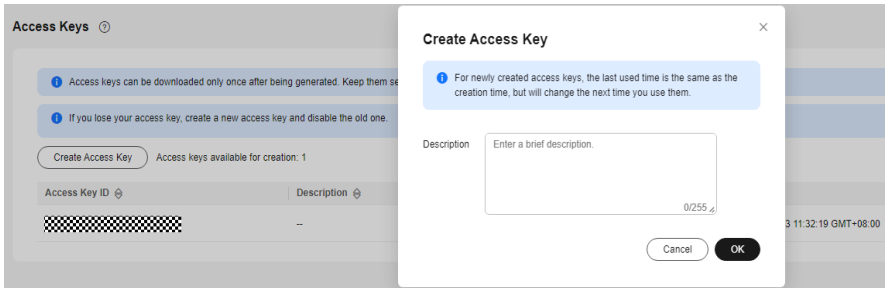
**Figure 3-18** Access Keys page



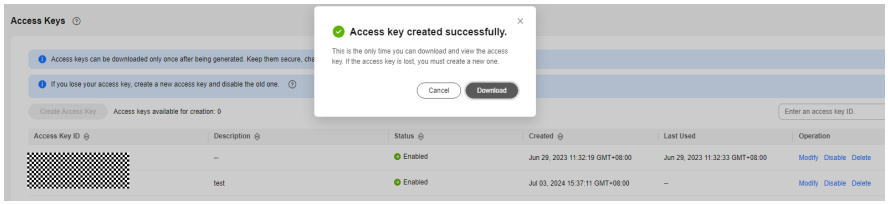
Click **Create Access Key**, and enter a description.

**Figure 3-19** Creating an access key




Click **OK** to generate an access key, and download it.

**Figure 3-20** Downloading the access key




View the AK in the access key list and the SK in the downloaded CSV file.

## 3.5.2.2 Preparing the Terraform Environment

## Installing Terraform

Terraform provides installation packages for different environments. For details, see **https://developer.hashicorp.com/terraform/downloads**.

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
```

## Basic Terraform Commands

Run Terraform to view the command details.

```
Terraform
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
```

```
  plan         Show changes required by the current configuration
  apply        Create or update infrastructure
  destroy      Destroy previously-created infrastructure

All other commands:
  console      Try Terraform expressions at an interactive command prompt
  fmt          Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get          Install or upgrade remote Terraform modules
  graph        Generate a Graphviz graph of the steps in an operation
  import       Associate existing infrastructure with a Terraform resource
  login        Obtain and save credentials for a remote host
  logout       Remove locally-stored credentials for a remote host
  metadata     Metadata related commands
  output       Show output values from your root module
  providers    Show the providers required for this configuration
  refresh      Update the state to match remote systems
  show         Show the current state or a saved plan
  state        Advanced state management
  taint        Mark a resource instance as not fully functional
  untaint      Remove the 'tainted' state from a resource instance
  version      Show the current Terraform version
  workspace    Workspace management

Global options (use these before the subcommand, if any):
  -chdir=DIR   Switch to a different working directory before executing the
               given subcommand.
  -help        Show this help output, or the help for a specified subcommand.
  -version     An alias for the "version" subcommand.
```

For details about the commands, see **https://developer.hashicorp.com/terraform/cli**.

## 3.5.3 Basic Terraform Syntax

Terraform's configuration language is based on the HashiCorp Configuration Language (HCL) syntax. It is easy to configure and read and compatible with the JSON syntax. For details, see **https://developer.hashicorp.com/terraform/language**.

## 3.5.4 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 **https://registry.terraform.io/providers/huaweicloud/huaweicloud/latest/docs/resources/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.

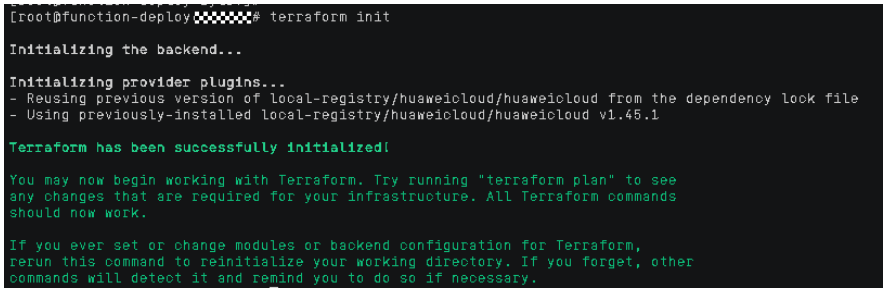
```
terraform {
  required_providers {
    huaweicloud = {
      source  = "huaweicloud/huaweicloud"
      version = ">= 1.40.0"
    }
  }
}

provider "huaweicloud" {
region = "cn-east-3" # Actual region
access_key = "*******" # Obtained key
secret_key = "*******" # Obtained key
}
```

```
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   =
"aW1wb3J0IGpzb24KZGVmIGhhbmRsZXIgKGV2ZW50LCBjb250ZXh0KToKICAgIG91dHB1dCA9ICdIZWxsbyBtZ
XNzYWdlOiAnICsganNvbi5kdW1wcyhldmVudCkKICAgIHJldHVybiBvdXRwdXQ="
}
```

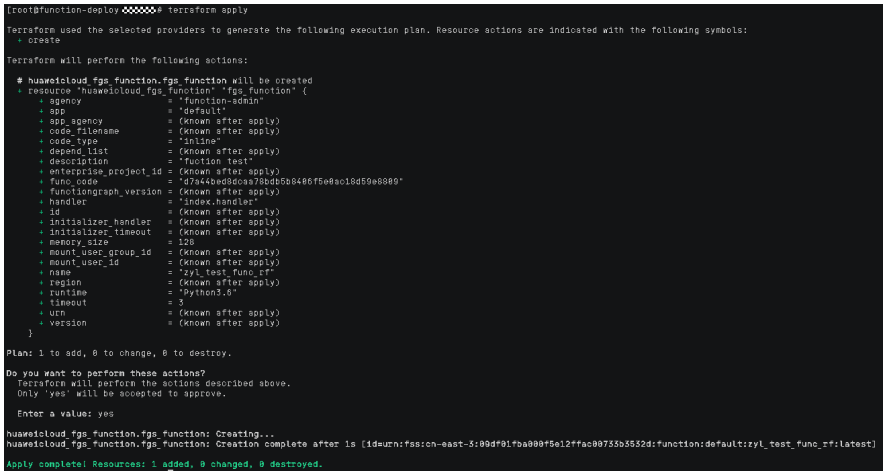
Replace **access_key** and **secret_key** with the AK/SK generated in **Obtaining an Access Key**.

## 3.5.5 Creating a Function by Running Terraform Commands

Go to the file path and run the **terraform init** command to initialize a working directory that contains the Terraform code.




Run the **terraform apply** command and enter **yes** after **Enter a value:**.




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

# 4 Configuring Functions

## 4.1 Configuring Initialization

### Overview

The initializer of a function is executed after an instance is started. The instance starts to process requests only after the initializer is executed. The initializer is executed only once during the lifecycle of a function instance. Initialization will be billed in the same way as function request processing.

### Scenario

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.

### Prerequisites

You have created a function.

### Initializing a Function

**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** > **Lifecycle** and enable **Initialization**.

**Figure 4-1** Enabling initialization

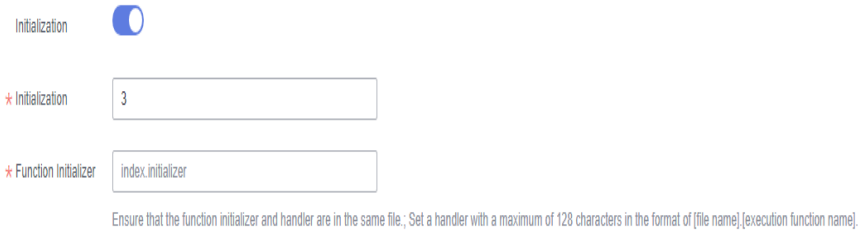


**Table 4-1** Parameter configuration

| Parameter | Description |
| --- | --- |
| Initialization | Enable initialization if needed. |
| Initialization Timeout (s) | Maximum duration the function can be initialized. Set this parameter if you enable function initialization.<br>The value ranges from 1s to 300s. |
| Initializer | You can enable function initialization on the **Configuration** tab page. 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]*.<br>**NOTE**<br>• This parameter is not required if function initialization is disabled.<br>• Ensure that the function initializer and handler are in the same file. |

NOTE

- Set the initializer 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]*.
- For details about the function code configuration, see **Creating a Deployment Package**.

**----End**

# 4.2 Configuring Basic Settings

## Introduction

After a function is created, **Memory (MB)**, **Handler**, and **Execution Timeout (s)** are automatically set based on your runtime. If needed, modify them based on this section.

## Prerequisites

You have created a function.

## Procedure

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 **Configuration** > **Basic Settings** and configure parameters based on **Table 4-2**. Parameters marked with an asterisk (*) are mandatory.

**Table 4-2** Basic settings

| Parameter | Description |
| --- | --- |
| App | After a function is created, it is automatically categorized into the **default** app and cannot be switched to other apps.<br>**NOTICE**<br>An app acts like a folder. In the future, functions will be managed by label for better experience. |
| *Handler | • 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>Example: **myfunction.handler**<br>• For a Java function, the handler must be named in the format of *[package name].[class name].[execution function name]*.<br>Example: **com.xxxxx.exp.Myfunction.myHandler**<br>• For a Go function, the handler name must be the same as the executable file name in the uploaded code package.<br>Example: 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>Example: **HelloCsharp::Example.Hello::Handler** |
| *Enterprise Project | Select a created enterprise project and add the function to it. By default, **default** is selected.<br>**NOTE**<br>If EPS is not enabled, this parameter is unavailable. For details, see **Enabling the Enterprise Project Function**. |
| *Execution Timeout (s) | Maximum duration the function can be executed. You can set this parameter on the **Configuration** tab page. If the execution takes longer than 90s, use asynchronous invocation.<br>The value ranges from 3s to 259,200s.<br>**NOTE**<br>The maximum duration for returning a result is 90s. If both the set timeout and the actual execution duration of a function exceed 90s, a message is displayed indicating that the function has timed out. However, the backend is still running. You can view the return result on the **Logs** tab page. |
| Memory (MB) | Memory of a function instance. Options: 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2560, 3072, 3584, 4096, 8192, 10,240. |

| Parameter | Description |
| --- | --- |
| Description | Description of the function, which cannot exceed 512 characters. |

4. Click **Save**.

# 4.3 Configuring Agency Permissions

## Overview

FunctionGraph works with other cloud services in most scenarios. Create a cloud service agency so that FunctionGraph can perform resource O&M in other cloud services on your behalf.

## Scenario

Before using FunctionGraph in the following scenarios, **create an agency**. Adjust the permissions granted to the agency to meet your service requirements. For example, grant the Admin permission in the development phase, and **change it to the fine-grained minimum permission in the product environment**. This ensures the required permissions while eliminating risks. Select the required action by referring to **Table 4-3**.

**Table 4-3** Common actions

| Scenario | Admin Permission | Fine-Grained Minimum Permission | Description |
| --- | --- | --- | --- |
| Using a custom image | SWR Administrator | Unavailable | SWR Admin: administrator who has all permissions for the Software Repository for Container (SWR) service.<br>For details about how to create a custom image, see **Deploying a Function Using a Container Image**. |

| Scenario | Admin Permission | Fine-Grained Minimum Permission | Description |
| --- | --- | --- | --- |
| Mounting an SFS Turbo file system | SFS Turbo ReadOnlyAccess | sfsturbo:shares:getShare (Query details about a file system)<br>sfsturbo:shares:showFsDir (Check whether a directory exists) | SFS Turbo ReadOnlyAccess: read-only permissions for SFS Turbo.<br>sfsturbo:shares:getShare: permission for querying a file system in SFS.<br>sfsturbo:shares:showFsDir: permission for checking whether a directory exists in SFS.<br>For details about how to mount an SFS Turbo file system, see **Mounting an SFS Turbo File System**. |
| Mounting an ECS shared directory | ECS ReadOnlyAccess | ecs:cloudServers:get (Query details about an ECS) | ECS ReadOnlyAccess: read-only permissions for ECS.<br>ecs:cloudServers:get: permission for querying an ECS.<br>For details about how to mount an ECS shared directory, see **Mounting an ECS Shared Directory**. |
| Configuring a reserved instance policy | AOM ReadOnlyAccess | aom:metric:get (Query a metric)<br>aom:metric:list (Query metric list) | AOM ReadOnlyAccess: read-only permissions for AOM.<br>aom:metric:get: permissions for querying a metric in AOM.<br>aom:metric:list: permissions for querying metric list in AOM. |
| Using a DIS trigger | DIS Administrator | Unavailable | Administrator who has all permissions for the DIS service.<br>For details about how to create a DIS trigger, see **Using a DIS Trigger**. |
| Using a DMS trigger | DMS ReadOnlyAccess | dms:instance:get (Query instance details) | DMS ReadOnlyAccess: read-only permissions for DMS.<br>dms:instance:get: permissions for querying instance details in DMS. |

| Scenario | Admin Permission | Fine-Grained Minimum Permission | Description |
| --- | --- | --- | --- |
| Configuring cross-domain VPC access | VPC Administrator | 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) | Users with the **VPC Administrator** permissions can perform any operations on all cloud resources of the VPC. To configure cross-VPC access, specify an agency with VPC management permissions.<br>Fine-grained minimum permission for VPC: permissions for unbinding a virtual IP address from a VM, querying a port, creating a port, querying a VPC, querying a subnet, and querying security groups or details about a security group.<br>For details about how to configure cross-domain VPC access, see **Configuring the Network**. |
| DNS Resolution | DNS ReadOnlyAccess | 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) | DNS ReadOnlyAccess: user with the permissions only to view DNS resources. To call a DNS API to resolve private domain names, specify an agency with the permissions to read DNS resources.<br>Fine-grained minimum permission for DNS: permission for querying record sets or querying tenant zone list in DNS.<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?** |

| Scenario | Admin Permission | Fine-Grained Minimum Permission | Description |
| --- | --- | --- | --- |
| Configuring asynchronous notification | If the target service is OBS:<br>OBS Administrator | 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) | OBS Administrator: administrator who has all permissions for OBS.<br>Fine-grained minimum permission for OBS: permissions for obtaining bucket metadata, creating a bucket, uploading objects using POST method, copying objects, appending an object, initializing a multipart task, uploading parts, and merging parts.<br>For details about how to configure asynchronous notification, see **Configuring Asynchronous Execution Notification**. |
| | If the target service is SMN:<br>SMN Administrator | smn:topic:publish (Publish a message)<br>smn:topic:list (Query the topic list) | SMN Administrator: administrator who has all permissions for SMN.<br>Fine-grained minimum permission for using SMN: permissions for publishing a message and querying the topic list.<br>For details about how to configure asynchronous notification, see **Configuring Asynchronous Execution Notification**. |
| | If the target service is DIS:<br>DIS Administrator | Unavailable | DIS Administrator: administrator who has all permissions for DIS.<br>For details about how to configure asynchronous notification, see **Configuring Asynchronous Execution Notification**. |

## Creating an Agency

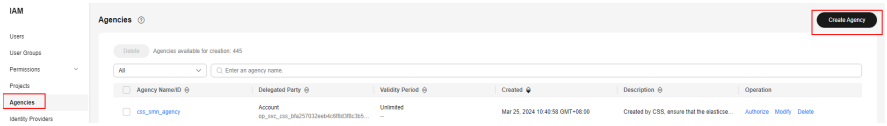
NOTE

In the following example, the **VPC Administrator** permission is assigned to FunctionGraph and this setting takes effect only in the authorized regions.

Create an agency by referring to **Creating an Agency** and set parameters as follows:

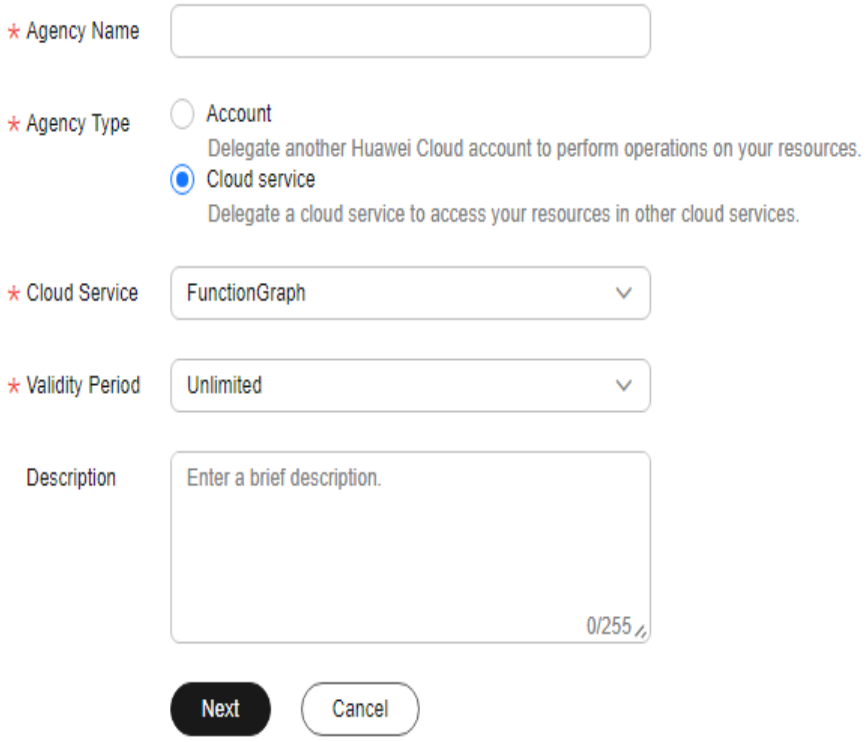
1. Log in to the IAM console.
2. On the IAM console, choose **Agencies** from the navigation pane, and click **Create Agency** in the upper right corner.

   **Figure 4-2** Creating an agency

3. Configure the agency.

   **Figure 4-3** Setting basic information

   - For **Agency Name**, enter **serverless-trust**.
   - For **Agency Type**, select **Cloud service**.
   - For **Cloud Service**, select **FunctionGraph**.
   - For **Validity Period**, select **Unlimited**.
   - **Description**: Enter the description.

4. Click **Next**. On the displayed page, search for the permissions to be added in the search box on the right and select the permissions. The **VPC Administrator** permission is used as an example.

**Figure 4-4** Selecting policies

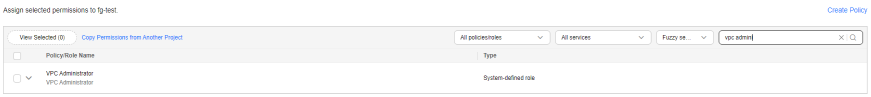
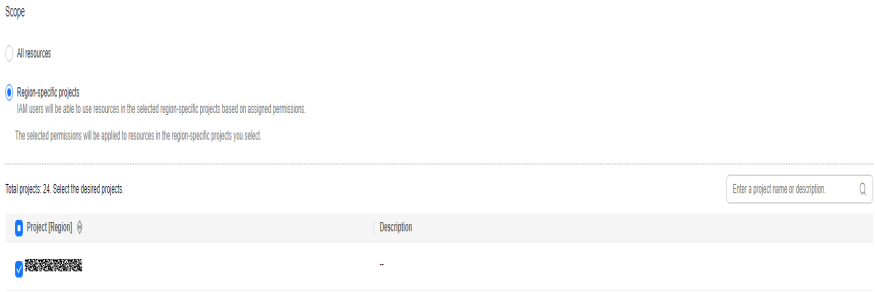



**Table 4-4** Example of agency permissions

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

5. Click **Next** and select the scope, for example, **Region-specific project**.

**Figure 4-5** Selecting the required permissions




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

1. In the left navigation pane of the management console, choose **Compute** > **FunctionGraph**. On the FunctionGraph console, choose **Functions** > **Function List** from the navigation pane.
2. Click the function to be configured to go to the function details page.
3. Choose **Configuration** > **Permissions**, click **Create Agency**, and set an agency based on site requirements by referring to **2**–**5**.

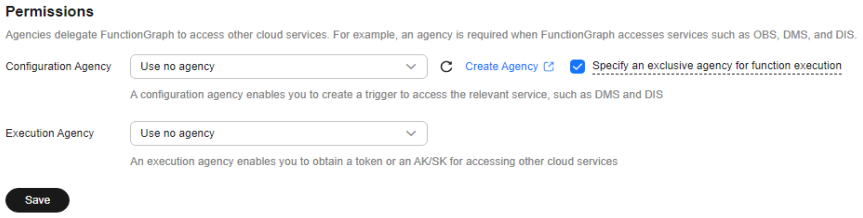
**Table 4-5** Agency configuration parameters

| Parameter | Description |
| --- | --- |
| Configuration Agency | Select a function that you have created. |
| Execution Agency | Mandatory if you select **Specify an exclusive agency for function execution**. |

NOTE

- To ensure optimal performance, select **Specify an exclusive agency for function execution** and set different agencies for function configuration and execution. You can also use no agency or specify the same agency for both purposes. **Figure 4-6** shows the agency options.

  **Figure 4-6** Setting agencies

- **Configuration Agency**: For example, to create Data Ingestion Service (DIS) triggers, first specify an agency with DIS permissions. If such an agency is not specified or the specified agency does not exist, no DIS triggers can be created.
- **Execution Agency**: This type of agency enables you to obtain a token and AK/SK from the context in the function handler for accessing other cloud services.

4. Click **Save**.

### Modifying an Agency

Modifying an agency: You can modify the permissions, validity period, and description of an agency on the IAM console.

CAUTION

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

## 4.4 Configuring the Network

### Public Access

By default, functions can access services on public networks. If the target public network service requires whitelist verification using a fixed IP address, **enable VPC access**, configure a public NAT gateway for the VPC, and bind an Elastic IP (EIP) to the gateway. For details, see **Configuring a Fixed Public IP Address**

### Configuring VPC Access

Functions can access resources in a VPC bound to it. If a function needs both VPC and public access, configure a public NAT gateway for the VPC and bind an EIP to the gateway. For details, see **Configuring a Fixed Public IP Address**.

Added support for enabling IPv6 in the VPC configuration of a function. (available only in **CN East2**.)

**Required Permissions**

Configure an agency by referring to **Configuring Agency Permissions**.

- Permissions for VPC access: an agency with the **VPC Administrator** permission or with the least permissions listed in **Table 4-6**

  **Table 4-6** 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 |

- Permissions for private domain name resolution: an agency with the **DNS ReadOnlyAccess** permission

**Procedure**

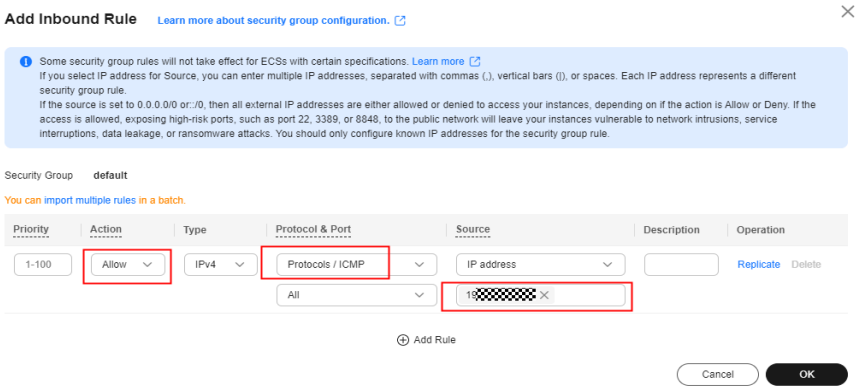
1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the function to be configured to go to the function details page.
3. Choose **Configuration** > **Network**, enable **VPC Access**, and specify a VPC and subnet.

   To enable **VPC Access**, you need to configure the following inbound and outbound rules in the default security group. For details, see .

   – 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**.

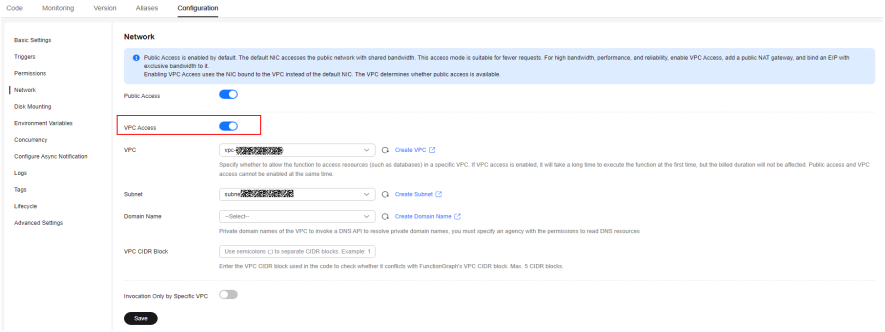
  

**Figure 4-7** Inbound rule




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

**Figure 4-8** Configuring VPC access




**NOTE**

1. For details on how to create a VPC and a subnet, see **Creating a VPC**.
2. Specify an agency with VPC administrator permissions for the function. For details, see **Configuring Agency Permissions**.
3. All functions of a tenant in a project can be bound to a maximum of four subnets. (Each project has a unique 32-digit project ID, which is allocated when your account is created. The project IDs of your account and IAM user are the same.)

4. (Optional) Configure the domain name.

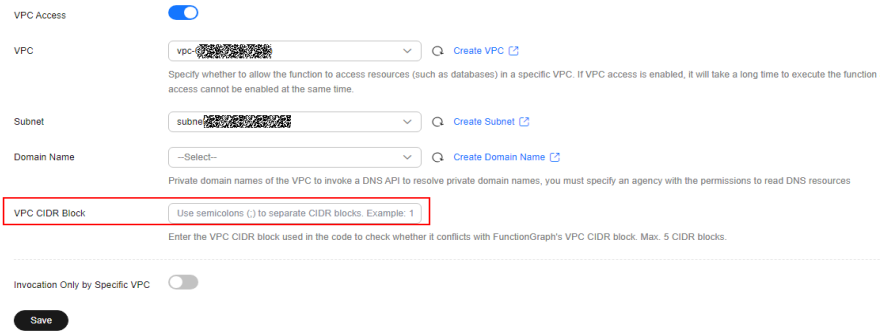
   Enter one or more private domain names of the VPC so that the function can use them to access resources in this VPC. See **Figure 4-8**.

   **NOTE**

   1. For details about how to create a private domain name, see **Creating a Private Zone**.
   2. 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**.

5. (Optional) Configure the VPC CIDR block.

**Figure 4-9** VPC CIDR block

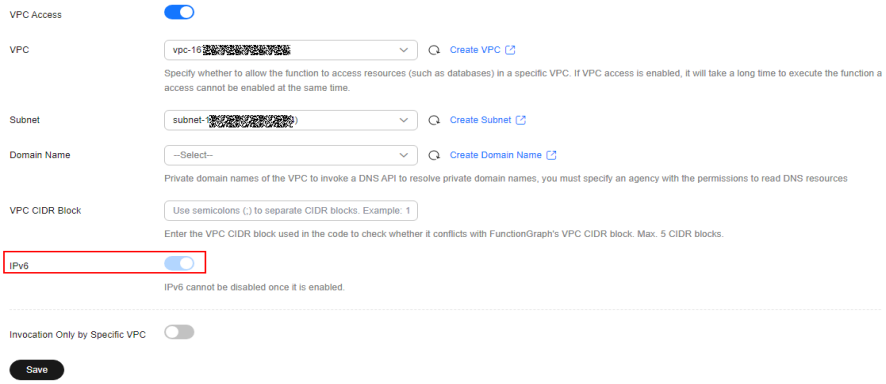



**NOTE**

- You can enter the VPC CIDR block used in the code to check whether it conflicts with FunctionGraph's VPC CIDR block.
- This feature is not available in **CN-Hong Kong**, **AP-Bangkok**, **AP-Singapore**, **AP-Jakarta**, **LA-Mexico City1**, **LA-Mexico City2**, **LA-Sao Paulo1**, and **LA-Santiago**.

6. (Optional) Enable IPv6. Currently, this feature is available only in **CN East2**.
   a. Create a VPC and ensure that IPv6 is enabled in the default subnet configuration. For details, see **Creating a VPC and Subnet**.
   b. In the VPC and Subnet areas in step **3**, select the created VPC and subnet.

**Figure 4-10** Enabling IPv6




**NOTE**

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**.

7. Click **Save**.

## Configuring a Fixed Public IP Address

If a function needs to access public network resources in a VPC or requires a fixed public IP address, configure a public NAT gateway for the VPC and bind an EIP to the gateway.

**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**.

**Procedure**

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**.
2. On the displayed page, enter gateway information, select a VPC (for example, **vpc-01**) and subnet, and confirm and submit the settings. For details, see **Buying a Public NAT Gateway**.
3. Click the public NAT gateway name. On the details page that is displayed, click **Add SNAT Rule**, set the rule, and click **OK**.

## Network Restrictions

FunctionGraph provides the following network access capabilities.

| Parameter | Description |
| --- | --- |
| Public Access | The default public NAT access bandwidth is shared between tenants in testing scenarios that involve a small number of requests. In production scenarios that require high bandwidth, performance, and reliability, enable VPC access for your function, add a public NAT gateway, and bind an EIP with an exclusive bandwidth to it. |
| VPC Access | If this option 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. |
| Invocation Only by Specific VPC | If this option is enabled, the function can be invoked only from the specified VPC instead of the public network. |

# 4.5 Configuring Disk Mounting

## Introduction

FunctionGraph allows you to mount file systems to your functions. Multiple functions can share the same file system. This greatly expands the function execution and storage space compared with the temporary disk space allocated to a function.

## Scenarios

**NOTICE**

Before mounting file systems, enable access over the following ports:

1. 111, 445, 2049, 2051, 2052, and 20048
2. 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?**

FunctionGraph supports the following types of file systems:

- SFS

  Scalable File Service (SFS) is a network-attached storage (NAS) service that provides scalable high-performance file storage. With SFS, shared access can be achieved among multiple ECSs, Bare Metal Servers (BMSs), and Cloud Container Engine (CCE) and Cloud Container Instance (CCI) containers. SFS is expandable to petabytes, and provides fully hosted shared file storage. It features high availability and durability, and seamlessly handles data-intensive and bandwidth-intensive applications. SFS is suitable for high-performance computing (HPC), media processing, file sharing, content management, and web services.

- SFS Turbo

  SFS Turbo supports the following storage classes: Standard (500 GB–32 TB), Standard-Enhanced (10 TB–320 TB), Performance (500 GB–32 TB), and Performance-Enhanced (10 TB–320 TB). SFS Turbo is expandable to 320 TB, and provides fully hosted shared file storage. It features high availability and durability, and supports massive quantities of small files and applications requiring low latency and high input/output operations per second (IOPS). SFS Turbo is suitable for high-performance websites, log storage, compression and decompression, DevOps, enterprise offices, and containerized applications. For details, see **SFS Service Overview**.

- ECS

  A directory on an ECS is specified as a shared file system (see **Mounting an ECS Shared Directory**) by using the network file system (NFS) service. The directory can then be mounted to a function in the same VPC as the ECS so that the function can read and write data in the directory. ECS file systems make it possible for dynamic expansion of compute resources. This type of file system is suitable for low service demand scenarios.

Benefits from using these file systems:

- The function execution space can be greatly expanded comparing with **/tmp**.
- A file system can be shared by multiple functions.
- ECS compute resources can be dynamically expanded and existing ECS storage capability can be used to achieve stronger computing performance.

NOTE

You can write temporary files in the **/tmp** directory. The total size of these files cannot exceed 10,240 MB.

## Creating an Agency

Before adding file systems to a function, specify an agency with permissions for accessing the file system services for the function.

There is a limit on the maximum number of agencies you can create, and cloud service agencies cannot be modified. Therefore, you are advised to create an agency with high-level permissions, for example, **Tenant Administrator**, to allow a function to access all resources in the selected region. For more information, see **Configuring Agency Permissions**.

## Creating an SFS File System

Log in to the SFS console, and create an SFS Capacity-Oriented or SFS Turbo file system. For details, see **Create a File System**.

## Mounting an SFS Capacity-Oriented File System

### Setting an Agency

On the **Configuration** tab page, select an agency that has been granted **SFS Administrator** permission in the selected region.

If no agencies are available, create one in IAM.

### Mounting a File System

On the function details page, choose **Configuration** > **Disk Mounting**, and click **Mount File System**. When you mount a file system to the target function for the first time, you need to set the user ID and group ID.

User ID and group ID (just like **uid** and **gid** in Linux) let the function access a specific file system.

NOTE

**User ID**: Enter –1 or an integer from 1 to 65,534, except 1000 and 1002. The default value **–1** indicates that the FunctionGraph backend automatically allocates an ID.

**Group ID**: Enter –1 or an integer from 1 to 65,534, except 1000 and 1002. The default value **–1** indicates that the FunctionGraph backend automatically allocates an ID.

For example, an ECS has been mounted with an SFS file system, and the owner of a directory in the file system is **test-user**. Then you can run the **id test-user** command to query the **uid** and **gid**.

Select an SFS file system and set the access path.

NOTICE

Up to two levels of function access paths are supported. **/mnt** and **/home** are existing paths. For FunctionGraph v1, you are advised to set an access path starting with **/mnt** or **/home**. For FunctionGraph V2, you are also advised to set an access path starting with **/mnt** or **/home**. If you set the path to **/mnt** or **/home**, message "failed to mount exist system path" will be displayed.

## Mounting an SFS Turbo File System

### Setting an Agency

Before mounting an SFS Turbo file system to a function, specify an agency that has been granted **SFS Administrator** and **VPC Administrator** permissions for the function. If no agencies are available, create one in IAM.

### Configuring VPC Access

An SFS Turbo file system is accessible only in the VPC where it has been created. Before mounting such a file system to a function, enable VPC access for the function.

1. On the SFS console, obtain the information about the VPC and subnet where a file system is to be mounted to your function. For details, see **File System Management**.
2. Enable VPC access by referring to **Configuring the Network** and enter the VPC and subnet obtained in **1**.

### Mounting an SFS Turbo File System

SFS Turbo file systems can be mounted in the same way as SFS file systems. Select a file system and set the access path.

## Mounting an ECS Shared Directory

### Specifying an Agency

Before mounting an ECS shared directory to a function, specify an agency that has been granted **Tenant Guest** and **VPC Administrator** permissions for the function. If no agencies are available, create one in IAM. For details, see **Creating an Agency**.

### Configuring VPC Access

Before adding an ECS shared directory, specify the VPC where the ECS is deployed. View the VPC information on the details page of the ECS. Click the VPC name to go to the VPC details page, and view the subnet.

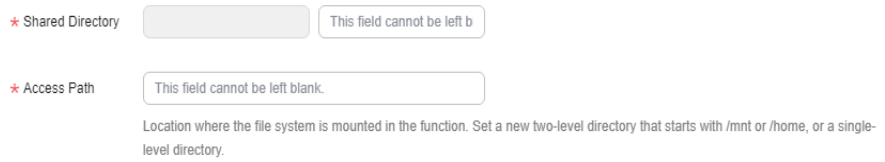
Set the acquired VPC and subnet for the function.

### Mounting an ECS Directory

Enter a shared directory and function access path.

**Figure 4-11** Setting the path




## Follow-up Operations

A function can read and write data in an access path in the same way as in the mounted file system.

Function logs can be persisted by configuring the log path as a subdirectory in the access path.

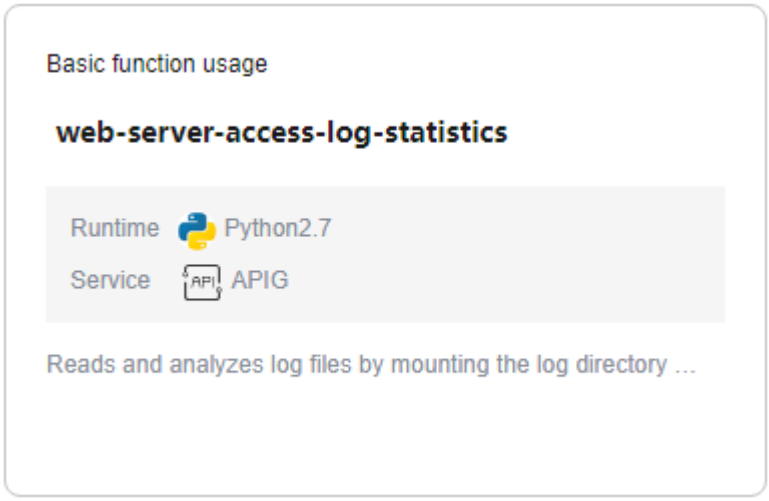
The following uses SFS Turbo and template **Web-Server-Access-Log-Statistics** as an example to describe how to analyze logs of servers running on the cloud.

**Step 1** Log in to the **FunctionGraph console**. In the navigation pane, choose **Templates**.

**Step 2** In the upper right corner of the **Templates** page, enter **Web-Server-Access-Log-Statistics** in the search box and press **Enter**.

**Step 3** In the search result, click **Configure**. The configuration page is displayed, as shown in **Figure 4-12**. Set the parameters as follows:

**Figure 4-12** Function template




- **Region**: Select the same region as the created VPC and file system. For details about how to create a VPC and file system, see **Configuring the Network** and **Creating a File System**.
- **Project**: Use **default**.
- **Function Name**: Enter a custom name.
- **Agency**: Select an agency with the file system, VPC, and APIG permissions. For details about how to create an agency, see **Configuring Agency Permissions**.
- **Enterprise Project**: Select an enterprise project as required.

- **Environment Variables**: **access_log_path** indicates the log file address. Set this parameter to **/home/test/access_log.log**.

  NOTE

  To specify file paths in the file system, use absolute paths starting with a slash (/). However, if no file system is mounted, you can skip adding the slash (/) and simply set the parameter to **code/access_log.log**.
- **Trigger Type**: The default value is **API Gateway (APIG)**. For details about how to configure APIG, see **Using an APIG Trigger**.
- **API Name**: Enter a custom name.
- **API Group**: Select a group based on the actual service.
- **Environment**: Select **RELEASE**.
- **Security Authentication**: Select **None**.
- **Protocol** and **Timeout (ms)**: Retain the default values.

**Step 4** After parameter configuration is complete, click **Create Function**.

**Step 5** On the function details page, click the **Code** tab, add the following code to the **index.py** file, and click **Deploy**.

```
import shutil
shutil.copyfile('/opt/function/code/access_log.log', '/home/test/access_log.log')
```

**Figure 4-13** Adding code


In addition, add the public dependency **Jinja2-2.10**. For details, see **Adding a Dependency for a Function**.

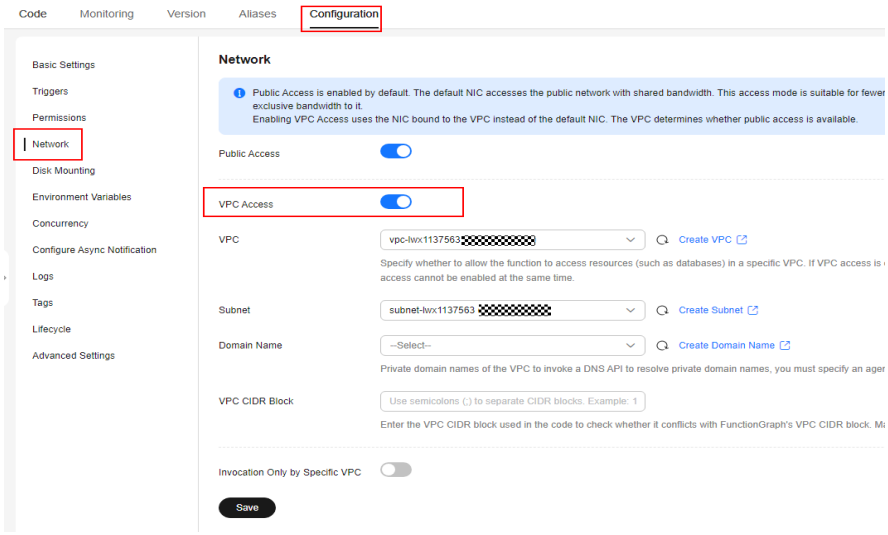
NOTE

If no file system is mounted, you do not need to add the preceding code.

**Step 6** On the function details page, choose **Configuration** > **Network** and enable **VPC Access**. Set **VPC** and **Subnet** to the created VPC and subnet, and click **Save**.

**Figure 4-14** VPC access

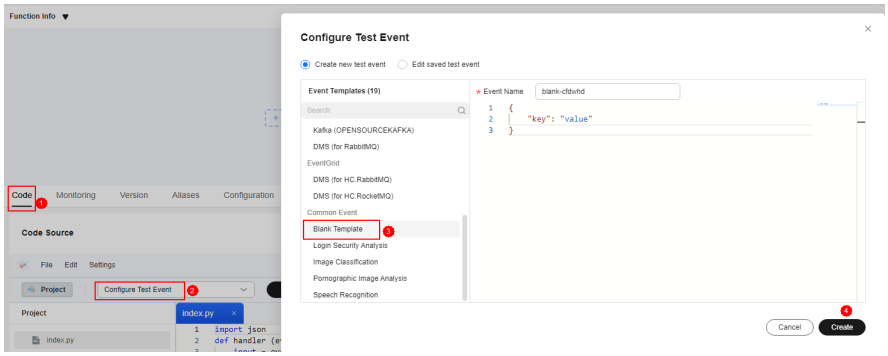



**Step 7** Choose **Disk Mounting**, click **Mount File System**, and select **SFS Turbo**.

- **File System**: Select an existing SFS Turbo file system.
- **Access Path**: Set this parameter to **/home/test**.
- **Shared Directory**: shared directory path of the file system. If this parameter is left blank by default, 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.

**Step 8** Click the **Code** tab, select **Configure Test Event**, create a **Blank Template**, and click **Create**.

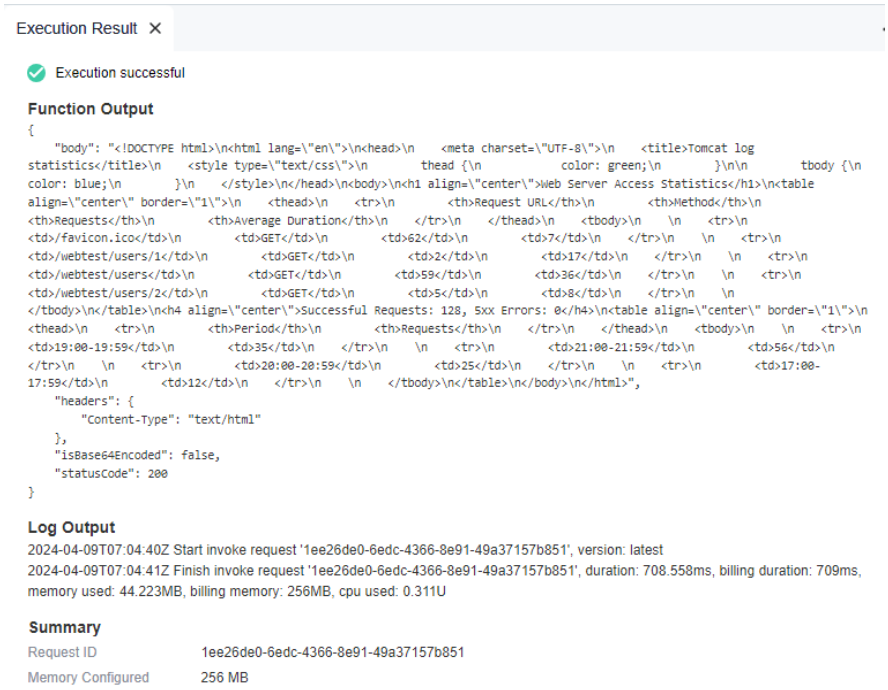
**Figure 4-15** Configuring a test event




**Step 9** Select the created test event and click **Test**.

**Figure 4-16** Test result




**Step 10** Choose **Configuration** > **Triggers**, copy the URL of the APIG trigger, and open the URL using a browser.

**Figure 4-17** Copying the URL

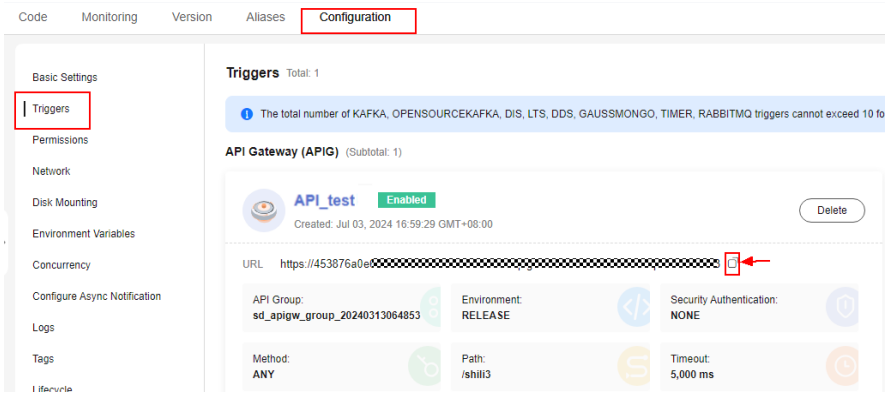
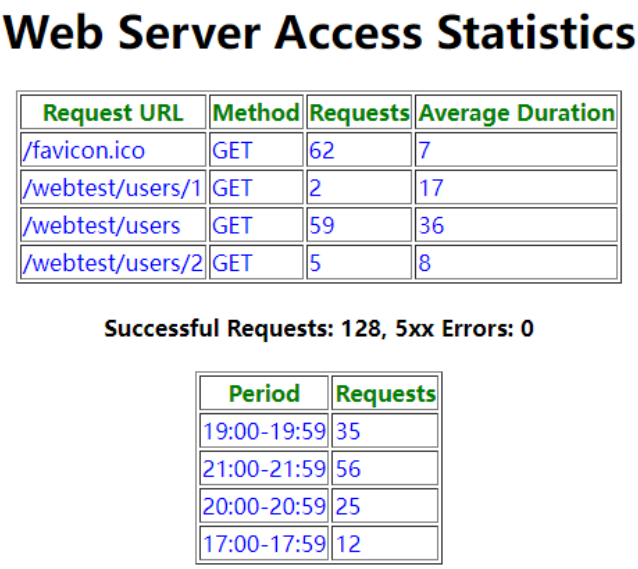


**Figure 4-18** Results display



Web Server Access Statistics

| Request URL | Method | Requests | Average Duration |
|---|---|---|---|
| /favicon.ico | GET | 62 | 7 |
| /webtest/users/1 | GET | 2 | 17 |
| /webtest/users | GET | 59 | 36 |
| /webtest/users/2 | GET | 5 | 8 |

**Successful Requests: 128, 5xx Errors: 0**

| Period | Requests |
|---|---|
| 19:00-19:59 | 35 |
| 21:00-21:59 | 56 |
| 20:00-20:59 | 25 |
| 17:00-17:59 | 12 |

**----End**

## Creating an NFS Shared Directory on ECS

1. **Linux**
   - CentOS, SUSE, EulerOS, Fedora, or openSUSE
     i. Configure a YUM repository.

     1. Create a file named **euleros.repo** in the **/etc/yum.repos.d** directory. Ensure that the file name must end with **.repo**.

     2. Run the following command to enter **euleros.repo** and edit the configuration:

     ```
     vi /etc/yum.repos.d/euleros.repo
     ```

     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
     ```

NOTE

Parameter description:

**name**: repository name

**baseurl**: URL of the repository

- HTTP-based network address: **http://path/to/repo**
- 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. **0**: The GPG check is disabled. **1**: The GPG check is enabled. If this option is not specified, the GPG check is enabled by default.

3. Save the configurations.

4. Run the following command to clear the cache:

```
yum clean all
```

ii. Run the following command to install nfs-utils:

```
yum install nfs-utils
```

iii. Create a shared directory.

When you open **/etc/exports** and need to create shared directory **/sharedata**, add the following configuration:

/sharedata 192.168.0.0/24(rw,sync,no_root_squash)

NOTE

The preceding configuration is used to share the **/sharedata** directory with other servers in the **192.168.0.0/24** subnet.

After the preceding command is run, run the **exportfs -v** command to view the shared directory and check whether the setting is successful.

iv. Run the following commands to start the NFS service:

```
systemctl start rpcbind
service nfs start
```

v. Create another shared directory.

For example, to create the **/home/myself/download** directory, add the following configuration to **/etc/exports**:

/home/myself/download 192.168.0.0/24(rw,sync,no_root_squash)

Restart the NFS service.

```
service nfs restart
```

Alternatively, run the following command without restarting the NFS service:

```
exportfs -rv
```

vi. (Optional) Enable automatic startup of the rpcbind service.

Run the following command:

```
systemctl enable rpcbind
```

– **Ubuntu**

i. Run the following commands to install nfs-kernel-server:

```
sudo apt-get update
sudo apt install nfs-kernel-server
```

ii. Create a shared directory.

```
vim /etc/exports
```

When you open **/etc/exports** and need to create shared directory **/sharedata**, add the following configuration:

/sharedata 192.168.0.0/24(rw,sync,no_root_squash)

NOTE

The preceding configuration is used to share the **/sharedata** directory with other servers in the **192.168.0.0/24** subnet.

iii. Start the NFS service.

```
service nfs-kernel-server restart
```

NOTE

After the preceding command is run, run the **exportfs -v** command to view the shared directory and check whether the setting is successful.

iv. Create another shared directory.

For example, to create the **/home/myself/download** directory, add the following configuration to **/etc/exports**:

/home/myself/download 192.168.0.0/24(rw,sync,no_root_squash)

Restart the NFS service.

```
service nfs restart
```

Alternatively, run the following command without restarting the NFS service:

```
exportfs -rv
```

2. **Windows**

1. Install the NFS server.

Paid software: haneWIN. Download the software at the **haneWIN official website**.

Free software: FreeNFS and WinNFSd. Download the software at the **SourceForge website**.

2. Enable the NFS function.

– In the case of WinNFSd, see **WinNFSd configuration**.

i. Download and decompress WinNFSd, and create the **nfs** folder in the decompressed directory.

ii. Set the sharing and read/write permissions on the **nfs** file.

1) Right-click the **nfs** file and choose **Properties**.

2) Click the **Sharing** tab, and then click **Share...**.

3) Add **Everyone** and click **Share**.

**Figure 4-19** Adding Everyone


4) Click the **Security** tab, select **Everyone** in the **Group or user names** list, and click **Edit**.

5) In the displayed **Security** dialog box, select **Everyone** from the **Group or user name** list, select **Read** and **Write** from the **Allow** check boxes in the **Permissions for Everyone** list, and click **OK**.

iii. Disable all firewalls, including the **Domain network**, **Private network**, and **Public network**. Enable them after the entire configuration is complete.

iv. Log in to the virtual server of the router and enable ports **111**, **2049**, and **1058** of the external network. (Note: An external IP address is required.)

v. Run the following command. For details, see **https://github.com/winnfsd/winnfsd**.

```
WinNFSd.exe -addr {Your own local IP address 192.168.xxx.xxx} F:\nfs /nfs
```

– In the case of haneWIN, perform the following steps:

i. Run the downloaded **.exe** file as the Windows system administrator.

ii. After the installation is complete, open the **NFS Server** file and choose **Edit** > **Preferences**.

iii. Retain the default settings on the **NFS**, **Server**, and **PortMapper** tab pages. Click the **Exports** tab, click **Edit exports file** to configure the shared directory, and click **Save**.

NOTE

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**.

iv. Click **OK**.

v. 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** is the IP address of the Windows operating system.
- **nfs** is the alias created when the shared directory is configured.
- **/mnt** is the local directory where the remote directory is mounted.

# 4.6 Configuring Environment Variables

## Overview

Environment variables allow you to pass dynamic parameters to a function without modifying code.

## Scenario

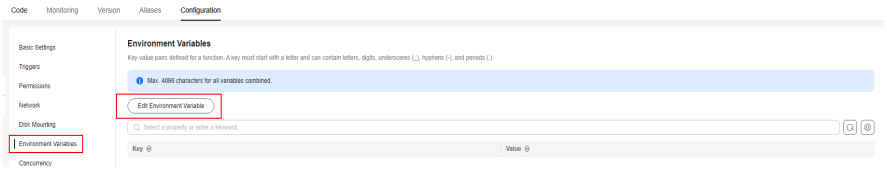
- Environment distinguishing: Configure different environment variables for the same function logic. For example, use environment variables to configure testing and development databases.
- Configuration encryption: Configure encrypted environment variables to dynamically obtain authentication information (account, password, AK/SK) 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.

## Procedure

You can configure encryption settings and environment variables to dynamically pass settings to your function code and libraries without changing your code.

**Figure 4-20** Adding environment variables




For example, for Node.js, encryption settings and environment variable values can be obtained from **getUserData(string key)** in **Context**. For details, see **Developing Functions in Node.js**.

**WARNING**

- Environment variables and encryption settings are user-defined key-value pairs that store function settings. Keys can contain letters, digits, and underscores (_), and must start with a letter.
- The total length of the key and value cannot exceed 4096 characters. (Available in these regions: ME-Riyadh, CN North-Ulanqab-Auto1, AF-Johannesburg, TR-Istanbul, CN North-Ulanqab1, LA-Sao Paulo1, CN-Hong Kong, AP-Singapore, CN East-Shanghai2, LA-Santiago, AP-Jakarta, CN Southwest-Guiyang201)
- When you define environment variables, FunctionGraph displays all your input information in plain text. For security purposes, do not include sensitive information.
- After encryption is enabled, key-value pairs are encrypted on the console and will remain encrypted during transmission.

## Preset Parameters

The following lists preset parameters. Do not configure environment variables with the same names as any of these parameters.

**Table 4-7** Preset parameters and description

| Environment Variable | Description | Obtaining Method and Default Value |
| --- | --- | --- |
| RUNTIME_PROJECT_ID | Project ID | Obtain the value from a Context interface or a system environment variable. |

| Environment Variable | Description | Obtaining Method and Default Value |
| --- | --- | --- |
| 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. |
| RUNTIME_MEMORY | Memory size configured for a function | Obtain the value from a Context interface or a system environment variable.<br>Unit: MB |
| RUNTIME_MAX_RESP_BODY_SIZE | Maximum size of a response body | Obtain the value from a system environment variable.<br>Default value: 6,291,456 bytes |
| 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. |

| Environment Variable | Description | Obtaining Method and Default Value |
| --- | --- | --- |
| RUNTIME_ROOT | Runtime package path | Obtain the value from a system environment variable.<br>Default value: **/home/snuser/runtime** |
| RUNTIME_CODE_ROOT | Path for storing code in a container | Obtain the value from a system environment variable.<br>Default value: **/opt/function/code** |
| RUNTIME_LOG_DIR | Path for storing system logs in a container | Obtain the value from a system environment variable.<br>Default value: **/home/snuser/log** |

## Example

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.

**Python**

In the following code snippet, **obs_output_bucket** is the bucket used for storing processed images.

```
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.getAccessKey()
    sk = context.getSecretKey()

    # download file uploaded by user from obs
    GetObject(obs_address, srcBucket, srcObjName, ak, sk)

    outFile = watermark_image(srcObjName)

    # Upload converted files to a new OBS bucket.
    PostObject (obs_address, outputBucket, outFile, ak, sk)

    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;
}
```

**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**.

Using environment variable **obs_output_bucket**, you can flexibly set the OBS bucket used for storing output images.

**Figure 4-21** Environment variables


# 4.7 Configuring Asynchronous Execution Notification

## Overview

Functions can be invoked synchronously or asynchronously. In asynchronous mode, FunctionGraph sends a response immediately after persisting a request. The request result cannot be known in real time. To retry when an asynchronous request fails or obtain asynchronous processing results, configure asynchronous settings.

## Scenario

- 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 notifications: FunctionGraph automatically notifies downstream services of the asynchronous execution result of a function 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.

## Procedure

**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** > **Configure Async Notification**. On the displayed page, click **Edit** next to **Asynchronous Notification Policy**.

**Figure 4-22** Configuring an asynchronous notification policy


**Step 4** Set parameters by referring to **Table 4-8**. For example, specify **FunctionGraph** for **Target Service**.

**Figure 4-23** Setting parameters

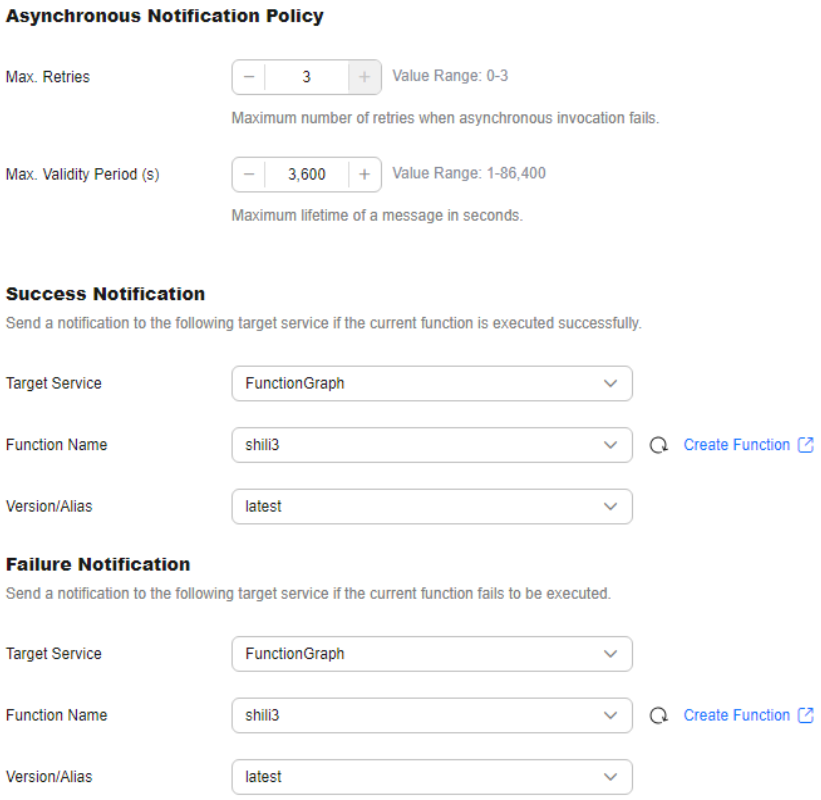


**Table 4-8** Parameter description

| Parameter | Description |
| --- | --- |
| Asynchronous Execution Notification Policy | • **Max. Retries**: maximum number of retries when asynchronous invocation fails. Value range: 0–3. Default value: **1**.<br>• **Max. Validity Period (s)**: maximum lifetime of a message in seconds. Value range: 1–86,400. |
| Success Notification | **Target Service**: to which a notification will be sent if a function is executed successfully.<br>1. FunctionGraph<br>2. OBS<br>3. DIS<br>4. SMN |
| Failure Notification | **Target Service**: to which a notification will be sent if a function fails to be executed.<br>1. FunctionGraph<br>2. OBS<br>3. DIS<br>4. SMN |

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

NOTE

1. If a message indicating insufficient permissions is displayed when you configure asynchronous execution notification, add the **FunctionGraph Administrator** permission. For details, see **Creating a User and Granting Permissions**.
2. Set an agency that allows FunctionGraph to access the target service.
3. To avoid cyclic invocation, do not set two functions as asynchronous execution targets of each other.

**----End**

## Configuration Description

For details about how to set the target for asynchronous invocation, see **Table 4-9**. The following shows an example:

```
{
      "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 4-9** Parameter description

| Parameter | Description |
| --- | --- |
| timestamp | Time when the invocation starts. |
| 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 | Code returned after function invocation. If the code is not 200, a system error occurred. |
| response_context.function_error | Invocation error information. |
| response_payload | Payload returned after function execution. |

## Triggering the Function

**NOTICE**

The return value of a function in asynchronous mode cannot exceed 256 KB; otherwise, a null value is returned.

**Step 1** Create three functions test1, test2, and test3 based on **Procedure**. Configure the asynchronous policy for test1 as follows:

- Success Notification
  - **Target Service**: Select **FunctionGraph**.
  - **Function Name**: Select **test2**.
  - **Version/Alias**: Retain the default value **latest**.
- Failure Notification
  - **Target Service**: Select **FunctionGraph**.

- **Function Name**: Select **test3**.
- **Version/Alias**: Retain the default value **latest**.

**Step 2** Click the **Code** tab and configure a test event.

Select **Blank Template**, copy the example code in **Configuration Description** to the code area, and click **Create**.

**Step 3** In the upper right corner of the FunctionGraph console, choose **Tools** > **API Explorer**. On the displayed API Explorer page, click **FunctionGraph** in the upper left corner.

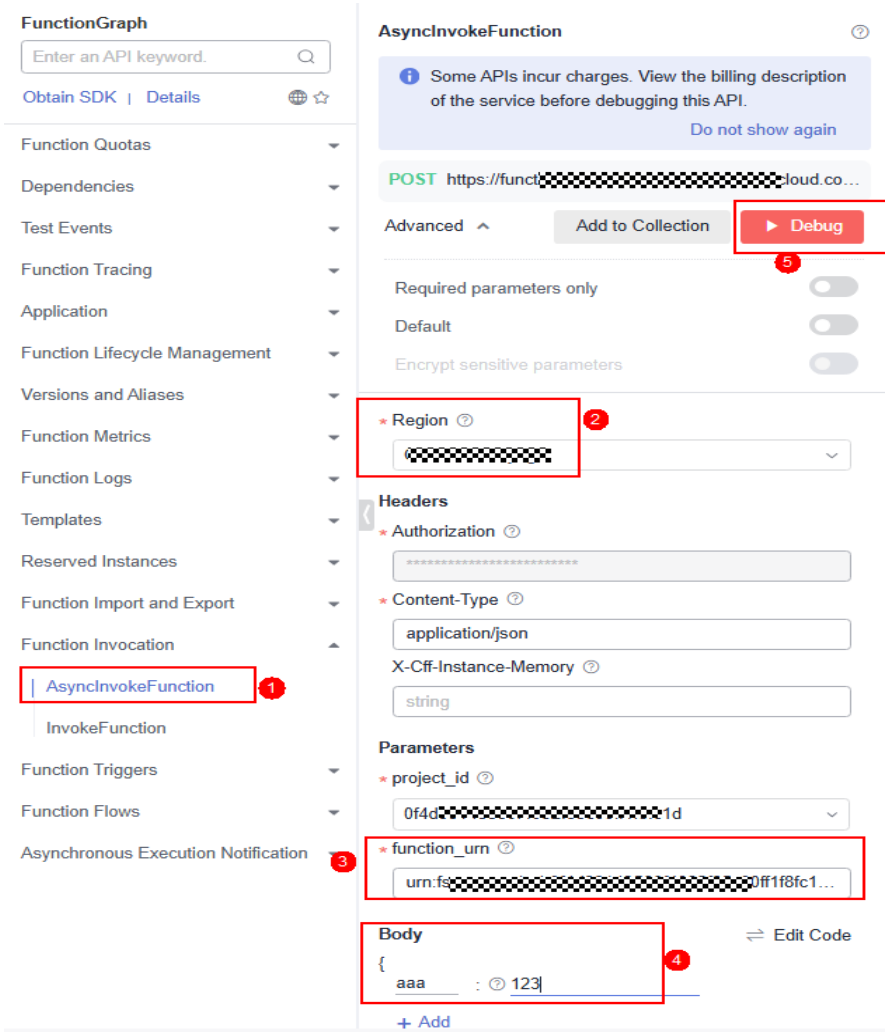
**Step 4** In the API list on the left, choose **Function Invocation** > **AsyncInvokeFunction**. Set the parameters as follows:

- **Region**: Choose the same region as in **step 1**.
- **function_urn**: URN of the function. Enter the URN of **test1**.
- **Body**: custom configuration. The format is **string:JSON string**, for example, **aaa:123**.

Retain the default values for other parameters and click **Debug**.

**Figure 4-24** Parameters for AsyncInvokeFunction




**----End**

## Viewing the Asynchronous Execution Result of the Function

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

**Step 2** If the function **test1** is successfully debugged in **Triggering the Function**, click the name of function **test2** to go to the details page. If the debugging fails, click the name of function **test3** to go to the details page.

**Step 3** Choose **Monitoring** > **Logs** to query running logs of function **test2** or **test3**.

**----End**

# 4.8 Configuring Single-Instance Multi-Concurrency

NOTE

This feature is supported only by FunctionGraph v2.

## Prerequisites

1. You can adjust the **Max. Requests per Instance** for existing functions, but once it is set to 1, it cannot be modified.
2. For HTTP functions created using container images, the default **Max. Requests per Instance** is **1**, but you can adjust it between 1 and 1,000.
3. To use the single-instance multi-concurrency feature in other scenarios, contact Huawei Cloud FunctionGraph engineers to apply for a whitelist.

## Overview

By default, each function instance processes only one request at a specific time. For example, to process three concurrent requests, FunctionGraph triggers three function instances. To address this issue, FunctionGraph has launched the single-instance multi-concurrency feature, allowing multiple requests to be processed concurrently on one instance.

## Scenario

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.

## 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 4-10** Comparison

| Comparison Item | Single-Instance Single-Concurrency | Single-Instance Multi-Concurrency |
| --- | --- | --- |
| Log printing | - | To print logs, Node.js Runtime uses the **console.info()** function, Python Runtime uses the **print()** function, and Java Runtime uses the **System.out.println()** function. In this mode, current request IDs are included in the log content. 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, Python Runtime.<br>log = context.getLogger()<br>log.info("test") |
| Shared variables | Not involved. | 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 | Not involved. | When there are too many 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**. |

## Configuring Single-Instance Multi-Concurrency

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 **Configuration** > **Concurrency**.

   Set parameters by referring to **Table 4-11** and click **Save**.

**Figure 4-25** Concurrency configuration

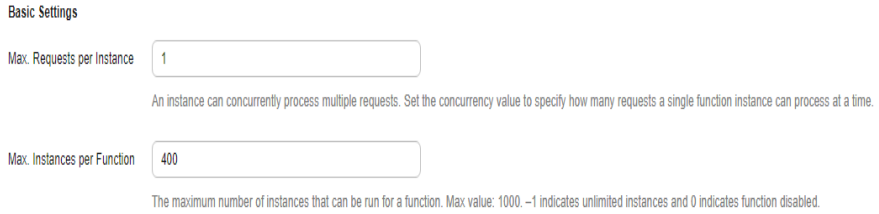



**Table 4-11** Description

| Parameter | Description |
| --- | --- |
| Max. Requests per Instance | Number of concurrent requests supported by a single instance. The default value is 1. The value ranges from 1 to 1,000.<br>**NOTE**<br>This parameter is required only when you create an HTTP function using a container image. |
| Max. Instances per Function | Maximum number of instances in which a function can run. Default: **400**. Maximum: **1000**. **–1**: The function can run in any number of instances.<br>**NOTE**<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 **Configuring Asynchronous Execution Notification**. |

## Configuration Constraints

- 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.

# 4.9 Managing Versions

## Overview

FunctionGraph allows you to publish one or more versions throughout the development, test, and production processes to manage your function code. The code and environment variables of each version are saved as a snapshot. After the function code is published, you can modify settings as required.

After a function is created, the default version is latest. Each function has the latest version. After the function code is published, you can modify the version configuration as required.

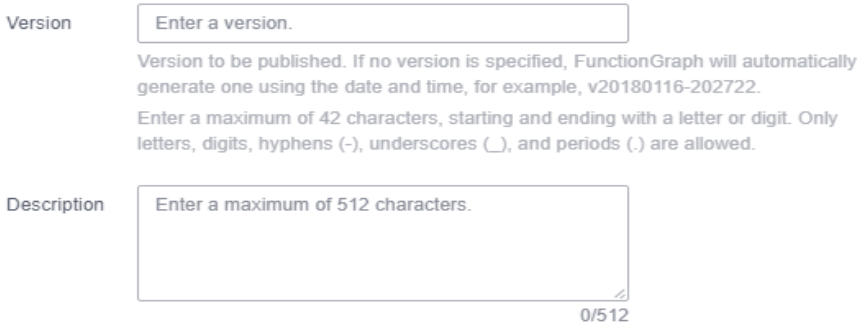
NOTE

A version is a snapshot of a function and corresponds to a tag in code. Each version contains the configuration and code of the function. By default, no trigger is bound to a new version. After a version is published, the configuration (such as environment variables) and code of the version cannot be updated, to ensure stability and traceability.

## Publishing a Version

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. On the **Version** tab page, click **Publish new version**.

   **Figure 4-26** Parameters for publishing a new version

   - **Version**: Enter a version number. If no version number is specified, the system automatically generates a version number based on the current date, for example, **v20220510-190658**.
   - **Description**: Enter a description for the version. This parameter is optional.
4. Click **OK**. The system automatically publishes a version. Then you will be redirected to the new version.

   NOTE

   - You can publish up to 20 versions for a function.
   - For a function whose latest version has been configured with reserved instances, the function configuration can be modified. By default, non-latest versions do not have reserved instances.
   - No disk is attached to a new version created based on latest. Environment variables cannot be set if no trigger has been bound to the version.

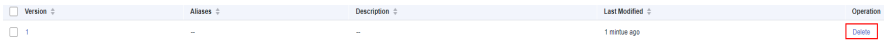
## Deleting a Version

1. Return 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. On the **Version** tab page of the latest version, select the version to delete.

**Figure 4-27** Deleting a version




NOTE

- The latest version of a function cannot be deleted.
- If a function version associated with aliases is deleted, the aliases will also be deleted.

4. Click **OK** to delete the version.

WARNING

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.

# 4.10 Managing Aliases

## Overview

An alias points to a specific function version. Create an alias and expose it to clients, for example, bind a trigger to the alias instead of the corresponding version. Then your modification to the version for update or rollback will be imperceptible to the clients. An alias can point to up to two versions with different weights for dark launch.

## Constraints

You can create up to 10 aliases for a function.

## Creating an Alias

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. On the **Aliases** tab page, click **Create Alias**.

**Figure 4-28** Creating an alias


- **Alias**: Enter an alias.
- **Version**: Select a version to be associated with the alias.
- **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**: Select an additional version to be associated. The latest version cannot be used as an additional version.
  - **Shift By**: You can shift requests to the additional version by **Percentage** or **Rule**.

**Table 4-12** Shifting mode

| Shift By | Description |
| --- | --- |
| 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. The weight value must be an integer from 0 to 100. |

| Shift By | Description |
| --- | --- |
| Rule | Available only for HTTP functions or functions with APIG triggers. The following parameters need to be configured:<br>• **Rule Type**: Select **All rules met** or **Any rule met** to forward requests with specified headers to the additional version.<br>• **Rules**: Set the header rule conditions. For details, see **Table 4-13**. |

**Table 4-13** Rule list

| Parameter Type | Parameter | Condition | Value |
| --- | --- | --- | --- |
| Header, which is unique by default. | Header name, which is case-insensitive. | Options: **=** and **in** | Header value, which is a character string. If the condition is **in**, you can set multiple values and separate them with commas (,), indicating that the traffic can be shifted when one of the values is met. |

For example, if you set **Alias** to **alias1**, **Version** to **version1**, **Additional Version** to **version2**, **Rule Type** to **All rules met**, **Header** to **aaa**, **Condition** to **=**, and **Value** to **123**, the function of **version2** will be executed for the request with function alias **alias1** and header parameter **aaa** with value **123**. If the request header does not meet the rule conditions, the function of **version1** will be executed.

- **Description**: Enter a description for the alias.

4. Click **OK**.

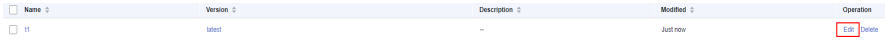
## Modifying an Alias

1. Return 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. On the **Aliases** tab page of the latest version, select the alias to modify.

   **Figure 4-29** Modifying an alias

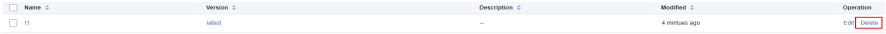
   
   

4. Modify the alias information, and click **OK**.

## Deleting an Alias

1. Return 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. On the **Aliases** tab page of the latest version, select the alias to delete.

   **Figure 4-30** Deleting an alias

4. Click **OK** to delete the version.

# 4.11 Configuring Dynamic Memory

## Overview

By default, a function is bound with only one resource specification. After enabling dynamic memory, you can configure a specification for request processing. If no specification is configured, the default one is used.

## Scenario

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.

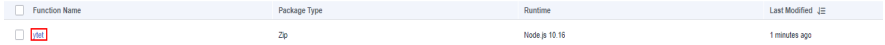
## Prerequisites

You have created a function according to **Creating a Function from Scratch**.

## Procedure

**Step 1** Log in to the FunctionGraph console, choose **Functions** > **Function List** in the navigation pane, and click the name of the created function.

**Figure 4-31** Selecting a created function



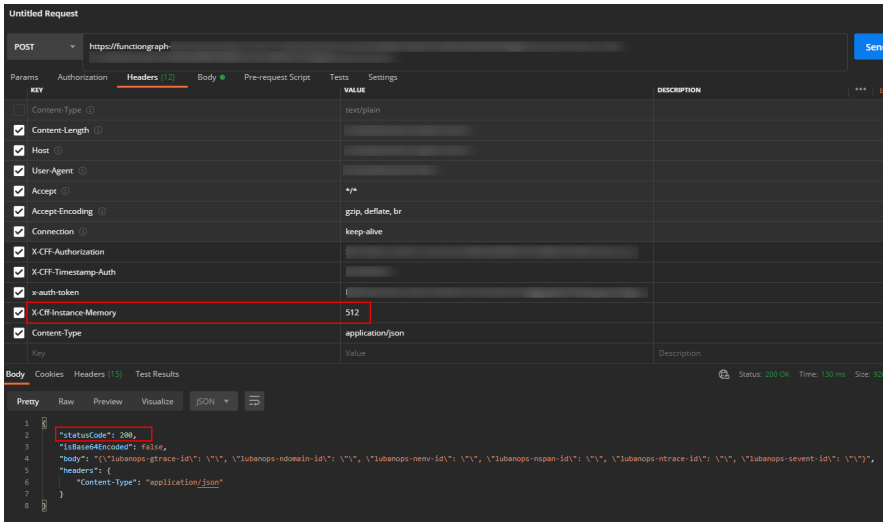

  

**Step 2** On the function details page, choose **Configuration** > **Advanced Settings** and enable **Dynamic Memory**.

**Step 3** Call the synchronous or asynchronous function execution API, add **X-Cff-Instance-Memory** to the request header, and set the value to **128**, **256**, **512**, **768**, **1024**, **1280**, **1536**, **1792**, **2048**, **2560**, **3072**, **3584**, **4096**, **8192**, or **10240**.

The following describes how to call an API using Postman. Add **X-Cff-Instance-Memory** to **Headers** and set the value to **512**. If the API is called successfully, error code 200 will be returned.

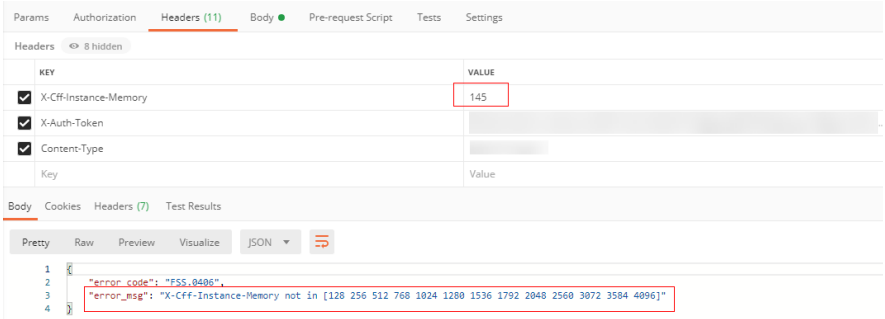
**Figure 4-32** Adding a request header and calling the function




NOTE

- If **Dynamic Memory** is not enabled, the memory size set when the function is created will be used by default.
- If **Dynamic Memory** is enabled but the memory value has not been set, the memory size set when the function is created will be used by default. If the API is called successfully, error code 200 will be returned.
- If **Dynamic Memory** is enabled but the memory value is not **128**, **256**, **512**, **768**, **1024**, **1280**, **1536**, **1792**, **2048**, **2560**, **3072**, **3584**, **4096**, **8192**, or **10240**, error code FSS.0406 will be returned when the API is called. You only need to reset the memory value.

**Figure 4-33** Invocation failure




**----End**

# 4.12 Configuring Heartbeat Function

## Introduction

A heartbeat function detects the following exceptions of your functions:

- Deadlock
- Memory overflow
- Network errors

After you configure a heartbeat function, FunctionGraph sends a heartbeat request to your function instance every 5s. 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.

## Constraints

1. **Heartbeat detection is available for Java functions.**
2. 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
   ```

3. The heartbeat function has no input parameter and its return value is Boolean.

## Procedure

**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** Click the **Configuration** tab and choose **Advanced Settings**.

**Step 4** Enable **Heartbeat Function** and set the function entry.

**Figure 4-34** Configuring a heartbeat function

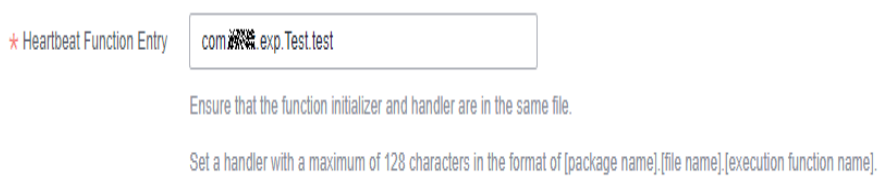


**Table 4-14** Heartbeat function configuration

| Parameter | Description |
| --- | --- |
| Heartbeat Function | If this option is enabled, FunctionGraph detects exceptions when your function is running. |
| Heartbeat Function Entry | The heartbeat function entry must be in the same file as your function handler.<br>The value can contain a maximum of 128 characters in the format of "[package name].[class name].[execution function name]". |

**Step 5** Click **Save**.

**----End**

# 4.13 Configuring Tags

## Introduction

Tags help you identify your cloud resources. When you have many cloud resources of the same type, you can use tags to classify them by dimension (for example, use, owner, or environment).

Add tags for a function on the **Configuration** tab page. Each function can have a maximum of 20 tags.

## Scenario

Tags help you identify and manage your function resources. For example, you can define a set of tags for function resources in your account to track the owner and usage of each function resource.

## Prerequisites

You have enabled Tag Management Service (TMS), or the pre-defined tags cannot be used. For details, see **Permission Management**.

## Adding a Tag

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 **Configuration** > **Tag**, and click **Add Tag**.
4. Add a tag key and value that meet the following rules:
   - Each tag consists of a key-value pair. Each key must be unique and have only one value.
   - Each function can have a maximum of 20 tags.

NOTE

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.

**Table 4-15** 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 these characters: _.:=+-@<br>• Must be unique and cannot exceed 128 characters. |
| Value | • Can be an empty string.<br>• Can contain UTF-8 letters, digits, spaces, and these characters: _.:/=+-@<br>• Max. 255 characters. |

5. Click **Save**.

   Tag values can be modified but tag keys cannot.

## 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. (Optional) Add more filters, such as runtime and package type.
4. View the search result in the function list.

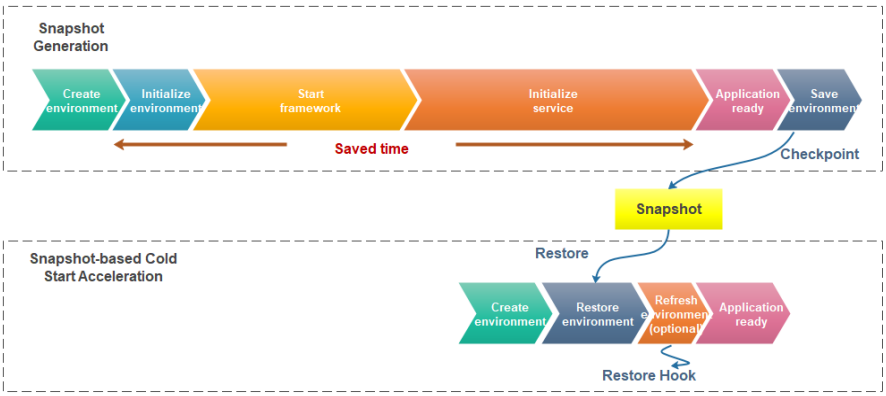
# 4.14 Configuring Snapshot-based Cold Start

## Introduction

Huawei Cloud's snapshot-based cold start solution is a performance optimization service. It is free of charge and requires only simple configurations and a few code changes.

After cold start acceleration 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. When the function scales with cold start, it **restores the execution environment from the snapshot** instead of repeating the initialization process, significantly increasing the startup speed.




## Constraints

- Only Java functions are supported.
- 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 customer service.
- When functions that depend on hard-coded host environment variables (such as **hostname** and **hostip**) are migrated to other hosts, problems may occur. Do not use such variables if possible. Contact customer service.
- Currently, only applications developed using x86 machines are supported.

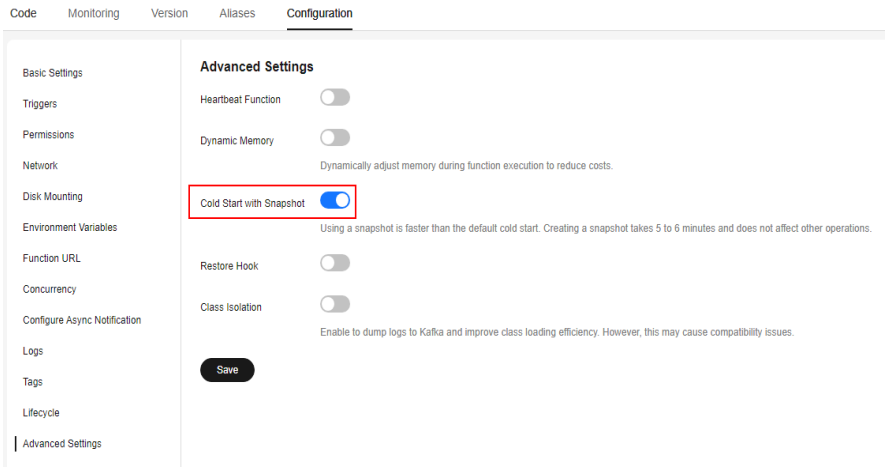
## Prerequisites

- A Java function has been created.

## Procedure

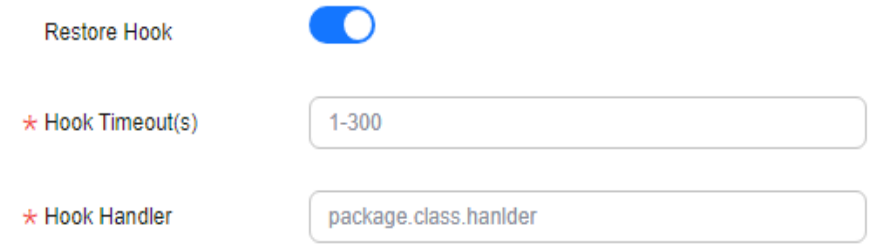
1. Log in to the FunctionGraph console, configure a Java function, and enable **Cold Start with Snapshot**.

   **Figure 4-35** Enabling Snapshot-based Cold Start

2. (Optional) Configure **Restore Hook** and implement the hook logic in the function code.

**Figure 4-36** Enabling Restore Hook

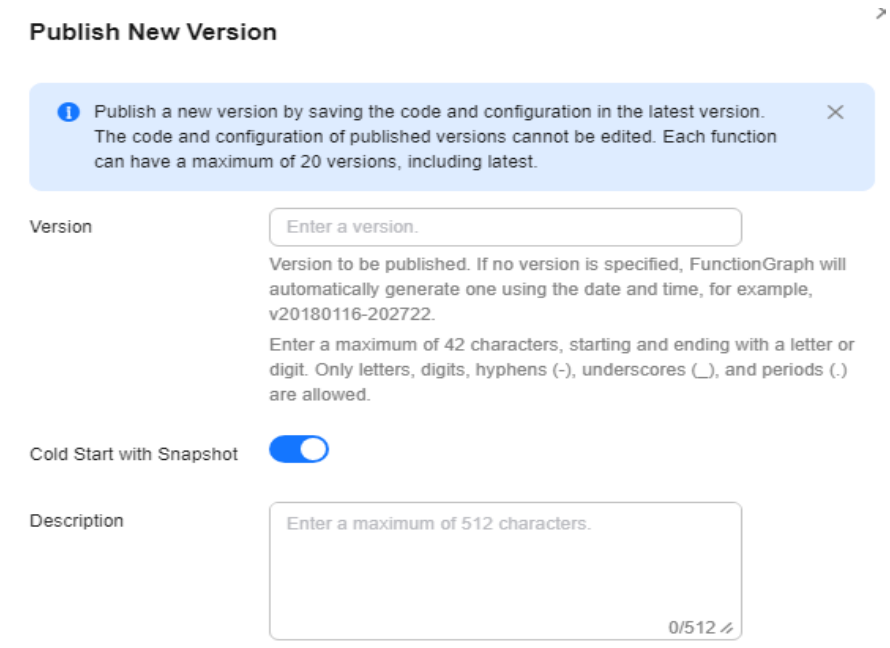



The following is an example of Restore Hook.

```
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"));
    }
```

3. Publish a new version to trigger snapshot creation.

**Figure 4-37** Publishing a new version




4. Wait until the snapshot is created (about 5 minutes).

**Figure 4-38** Creating snapshot...

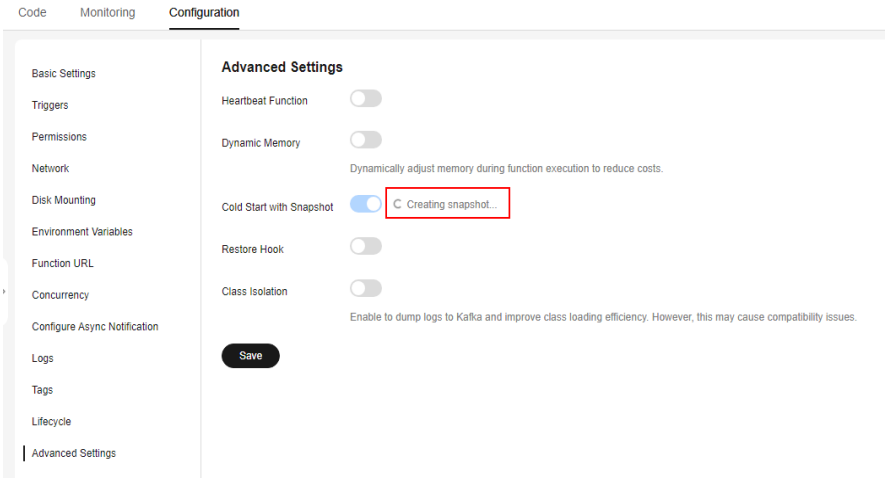
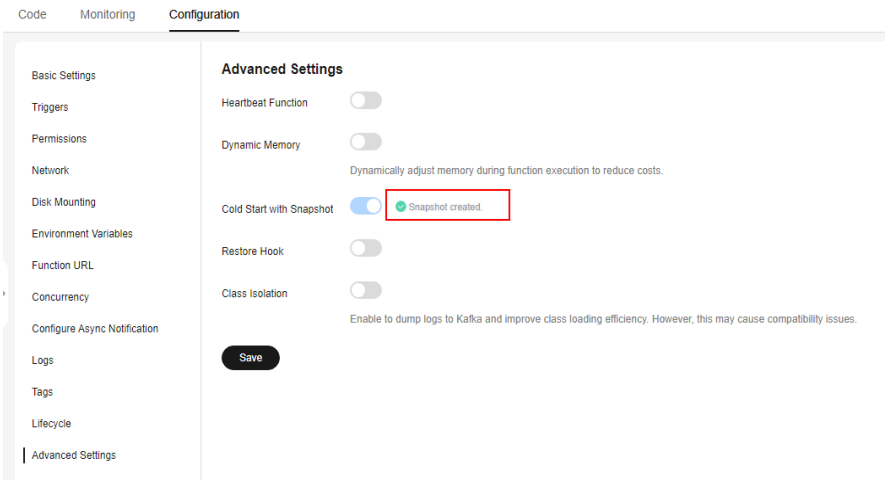



**Figure 4-39** Snapshot created




5. Invoke the Java function to enjoy higher performance.

# 4.15 Configuring a Log Group and Log Stream

NOTE

This feature is supported only by FunctionGraph V2.

## Introduction

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**.

## Prerequisites

You have created a log group and log stream on the LTS console.

## Procedure

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 **Configuration** > **Log**, and configure log collection according to **Table 4-16**.

   **Table 4-16** 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.<br>By default, the log group (starting with **functiongraph**) automatically generated by FunctionGraph is selected. |
| Log Stream | Select a log stream in the specified log group.<br>By default, the log stream (starting with the function name) automatically generated for the function is selected. |

4. Click **Save**.
5. After the function is invoked, **view logs** in the specified log group and log stream.

   NOTE

   You can change the log stream if needed.

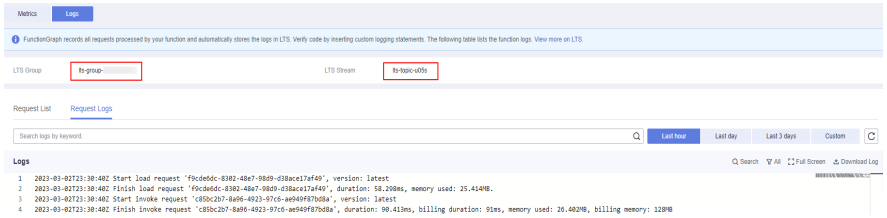
  

### Viewing Function Logs

View function logs in the specified log group and log stream.

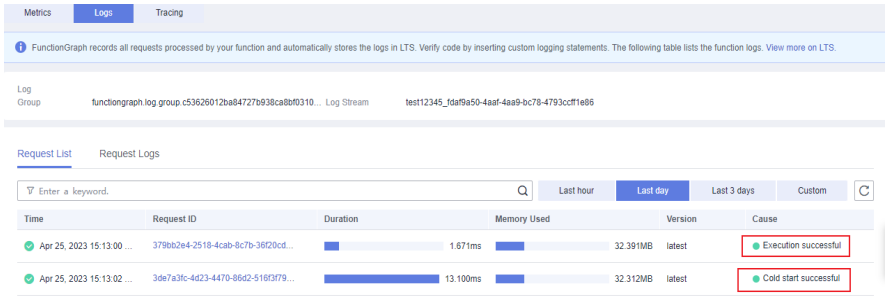
1. Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click the function to be configured to go to the function details page.
3. Choose **Monitoring** > **Logs** and view the function logs.
   - As shown in **Figure 4-40**, logs of the function are 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 4-40** 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 4-41** Logs

# 4.16 Stateful Functions

## 4.16.1 Introduction

### State

In a narrow sense, states are data cyclically used during computing. For example, most model training algorithms in machine learning construct a proper loss function to calculate the gradient using chain rules, and then cyclically update existing model parameters in recursive mode. These model parameters are states cyclically used during training.

In FunctionGraph's kernel, states are process data generated during the invocation of functions in the same app by using the **invoke** API.

### Stateful Function

Functions are classified into stateless and stateful functions. Stateful functions have their states stored in a FaaS service, and define the initial state methods and attributes.

### Function Instance

Stateful function instances are logical. Different from ordinary function instances, stateful function instances can persist state data generated during execution and restore the state data for the next invocation.

Stateful function instances are identified by **InstanceID**, which is a unique 32-bit logical ID allocated by the system.

### objRef

In FunctionGraph's kernel, objRef is an object which obtains the results of function invocation requests that have not yet been completed. objRef separates values from computing (function invocation) for flexibility and concurrency.

## 4.16.2 Development Instructions

Stateful functions support Java, Node.js, and Python.

They are created, edited, and triggered in the same way as stateless functions, with only the following differences:

1. Enable **Stateful Function** on the **Configuration** > **Advanced Settings** page.

   **Figure 4-42** Stateful Function switch

2. Compile initialization code in your function using the template in the *FunctionGraph Developer Guide*.

### State Management

- **Generating Function Instances**

  When a function instance invocation handle is created, the system generates a new state instance and loads it to the function for execution.

  For details about how to create such handles for different languages, see the demo template. The following uses Java as an example:

  a. **f = new Function (context, functionName)**: There is no need to specify a function instance name.

  b. **f = new Function (context, functionName, instanceName)**: Specify a new function instance name to invoke the relevant stateful function.

- **Restoring Function Instance Invocation Handles**

  Specify the function instance name with **instanceName** to restore invocation handles.

  f = new Function (context)

f.getInstance (functionName, instanceName)

- **Accessing Function Instances**

  Use **context.state** to access the state data bound to function instances.

- **State Operations**
  - Before running a function, the system automatically loads the state data to **context.state** based on a function instance ID bound to the invocation handle.
  - Use **context.state** to access the state data bound to a function instance.
  - Use **f.saveState()** to save state data changes.
  - If **f.saveState()** is called after a function is executed, the system automatically persists the function's state data. If **f.saveState()** is not called after a function is executed, the state data will not be saved.

### objRef Operations

**How Do I Create an objRef?**

- Use **f.invoke()** to return objRef.

**How Do I Use objRef?**

- Use **objRef.get** to obtain the value of objRef.

## 4.16.3 Precautions

### Constraints

1. Currently, stateful functions are only available in **AP-Singapore**.
2. A stateful function cannot recursively invoke itself.
3. Return values of Java functions can only be in JsonObject.
4. The size of a single state cannot exceed 256 KB. Each tenant can create a maximum of 100 stateful function instances.
5. State data is not encrypted during execution and persistence. Encrypt your sensitive data if any.
6. Stateful functions do not support reserved instances.
7. **instanceName** can contain 1 to 128 letters and digits.
8. Functions that access or are accessed by resources in a VPC must be in the same subnet as the resources.

### Special init Method of Node.js

Node.js functions have an init() method for initialization and generating instance IDs.

To invoke a stateful function, use the following code:

```
const f = new Function(context, functionName, instanceName);
await f.init();
```

# 4.17 Shared VPC

With Resource Access Manager (RAM), you can share subnets in a VPC with other accounts. In FunctionGraph, you can configure a subnet that other users share with you to access resources in the subnet.

To access the subnet shared by other users, ensure that the owner of the subnet has configured subnet sharing for you. For details, see **Sharing a VPC Subnet with Other Accounts**. Then select the shared subnet in the network configuration of the function. For details, see **Configuring VPC Access**. If the subnet owner stops the sharing, you cannot access the subnet.

For more information about VPC subnet sharing, see section **VPC Sharing** in the *Virtual Private Cloud User Guide*.

# 5 Online Debugging

## Precautions

Event data is passed to the handler of your function as an input. After configuration, event data is persisted for later use. Each function can have a maximum of 10 test events.

## Creating a Test Event

**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**.

**Step 4** In the **Configure Test Event** dialog box, configure the test event information according to **Table 5-1**. The parameter marked with an asterisk (*) is mandatory.

**Table 5-1** 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 Template | If you select **blank-template**, you can create a test event from scratch.<br>If you select a template, the corresponding test event in the template is automatically loaded. For details about event templates, see **Table 5-2**. |
| *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, **even-123test.** |
| Event data | Enter a test event. |

**Table 5-2** Event template description

| Template Name | Description |
| --- | --- |
| API Gateway (APIG) | Simulates an APIG event to trigger your function. |
| 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. |
| 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. |
| Object Storage Service (OBS) | Simulates an OBS 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 (OPENSOURCEKAFKA) | 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-tag** function template. |
| Pornographic Image Analysis | Serves as an input for the **porn-image-analysis** function template. |

| Template Name | Description |
| --- | --- |
| Speech Recognition | Serves as an input for the **voice-analysis** function template. |

**Step 5** Click **Create**.

**----End**

## Testing a Function

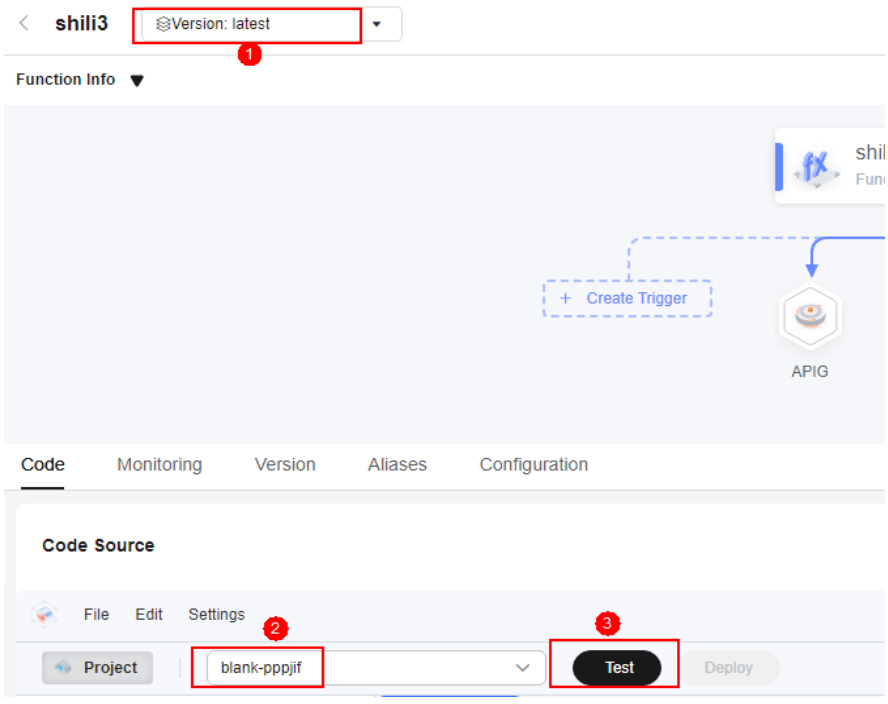
After creating a function, you can test it online to check whether it can run properly as expected.

**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**.

**Figure 5-1** Selecting a test event




**Step 4** Click **Test**. The function test result is displayed.

NOTE

The **Log Output** area displays a maximum of 2 KB logs. To view more logs, see **Managing 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, select a version and 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 5-3**.

**Table 5-3** 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**

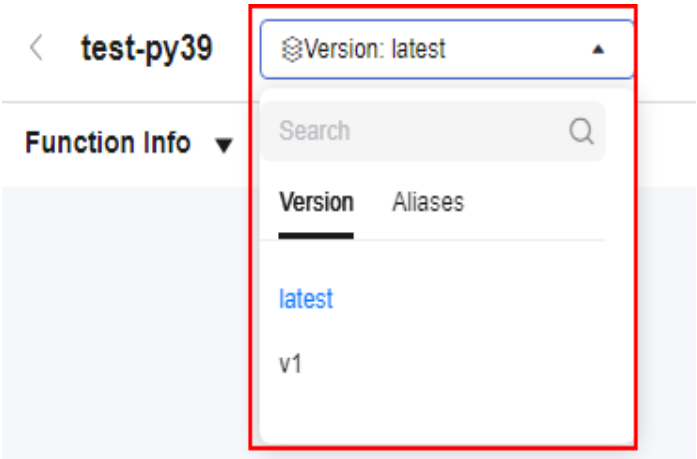
## Deleting a Test Event

**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 function details page that is displayed, select a version, as shown in **Figure 5-2**.

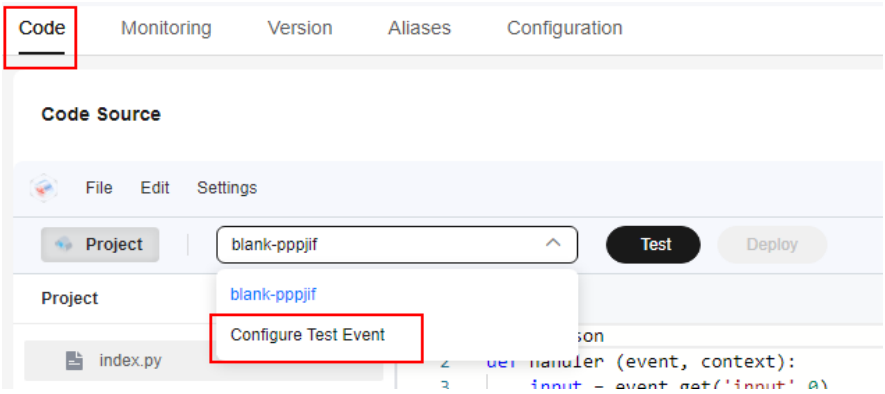
**Figure 5-2** Selecting a FunctionGraph version




**Step 4** On the **Code** tab page, click **Configure Test Event**. The editing page is displayed, as shown in **Figure 5-3**.

**Figure 5-3** Selecting Configure Test Event




**Step 5** 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**.

**Figure 5-4** Deleting a test event

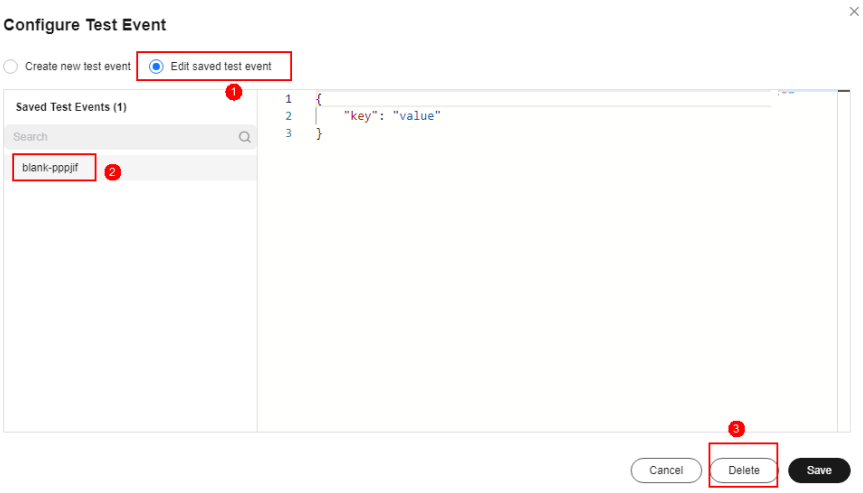



**Table 5-4** Configuring test event information

| Parameter | Description |
| --- | --- |
| Create new test event | Select a test event template. |
| Edit saved test event | Select the test event you want to delete. |

**----End**

# 6 Creating Triggers

## 6.1 Managing Triggers

### Enabling or Disabling a Trigger

You can enable or disable triggers as required. **Note that SMN, APIG 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

You can delete triggers that will no longer be used.

**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**.

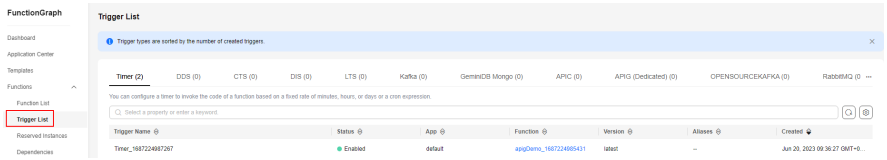
**----End**

**NOTE**

On the trigger list page, the type of trigger in use will be preferentially displayed.

**Figure 6-1** Trigger display




# 6.2 Using a Timer Trigger

This section describes how to create a timer trigger to invoke your function based on a fixed rate or cron expression.

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

## Prerequisites

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

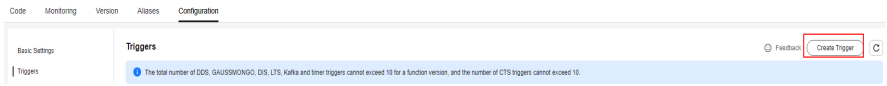
## Creating a Timer Trigger

**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 6-2** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **Timer**.
- **Timer Name**: Enter a timer name, for example, **Timer**.
- **Rule**: Set a fixed rate or a cron expression.
  - **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.
  - **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 Expressions for a Function Timer Trigger**.
- **Enable Trigger**: Choose whether to enable the timer trigger.
- **Additional Information**: The additional information you configure will be put into the **user_event** field of the timer event source. For details, see **Supported Event Sources**.

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

**----End**

### Viewing the Execution Result

After the timer trigger is created, the function is executed every 1 minute. To view the function running logs, perform the following steps:

**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**

## 6.3 Using an APIG (Dedicated) Trigger

This section describes how to create an APIG trigger and call an API to trigger a function.

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

### Prerequisites

You have created an API group, for example, **APIGroup_test**. For details, see **Creating an API Group**.

### Creating an APIG 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** Set the following parameters:

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

**Step 4** Click **Create**.

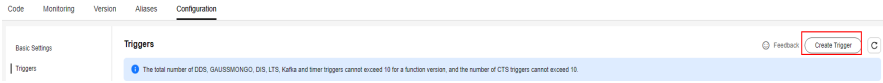
**Step 5** On the **Code** tab page, copy the following code to the code window and click **Deploy**.

```
# -*- 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**.

**Figure 6-3** Creating a trigger




**Step 7** Configure the trigger information.

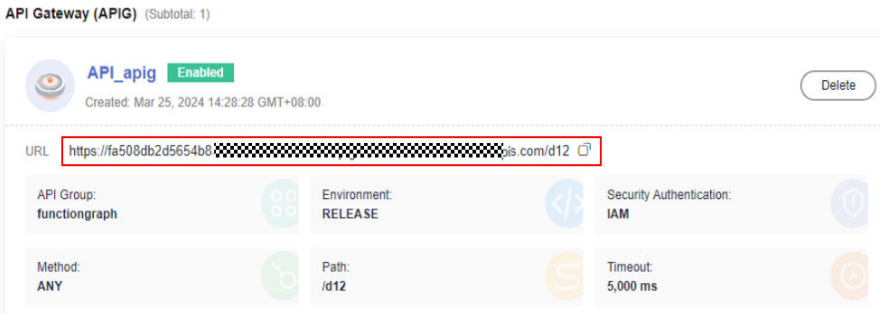
**Table 6-1** Trigger information

| Parameter | Description |
| --- | --- |
| Trigger Type | Select **API Gateway (Dedicated Gateway)**. |
| API Instance | Select an instance. If no instance is available, click **Create Instance**. |
| API Name | Enter an API name, for example, **API_apig**. |
| API Group | An API group is a collection of APIs. You can manage APIs by API group.<br>Select **APIGroup_test**. |
| 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.<br>To ensure that the API can be called, select **RELEASE**. |
| Security Authentication | There are three authentication modes:<br>• **App**: AppKey and AppSecret authentication. This mode is of high security and is recommended. For details, see **App Authentication**.<br>• **App**: AppKey and AppSecret authentication. This mode is of high security and is recommended. For details, see **App Authentication**.<br>• **None**: No authentication. This mode grants access permissions to all users.<br>Select **None**. |
| Protocol | There are two types of protocols:<br>• HTTP<br>• HTTPS<br>Select **HTTPS**. |
| Timeout (ms) | Enter **5000**. |

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

**Figure 6-4** Creating a trigger




NOTE

1. **URL** indicates the calling address of the APIG trigger.
2. After the APIG trigger is created, an API named **API_apig** 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** Enter the URL of the APIG trigger in the address bar of a browser, and press **Enter**.

**Step 2** View the execution result, as shown in **Figure 6-5**.

**Figure 6-5** Returned result


NOTE

1. The input for APIG invocation comes from an event template provided by the function. For details, see **Table 5-2**.
2. The function response for APIG invocation is encapsulated and must contain **body(String)**, **statusCode(int)**, **headers(Map)**, and **isBase64Encoded(boolean)**.

**----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 the function details page.

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

**----End**

# 6.4 Using a Kafka Trigger

This section describes how to create a Kafka trigger and configure a Kafka event to trigger a function.

After a Kafka trigger is used, FunctionGraph periodically polls for new messages in a specific topic in a Kafka instance and passes the messages as input parameters to invoke functions. For details about the DMS for 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**.
- In cases of Kafka data processing failure, the Kafka trigger will discard records that are larger than 6 MB.

## Prerequisites

Before creating a trigger, ensure that you have prepared the following:

- You have created a function. For details, see **Creating a Function from Scratch**.
- You have enabled VPC access for the function. For details, see **Configuring the Network**.
- 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**.

## 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 6-6** Creating a trigger


**Step 4** Set the following parameters:

- **Trigger Type**: Select **DMS (for Kafka)**.
- **Instance**: Select a Kafka premium instance.
- **Topic**: Select a topic of the Kafka premium instance.
- **Batch Size**: Set the number of messages to be retrieved from a topic each time.
- **Username**: Enter the username of the instance if SSL has been enabled for it.

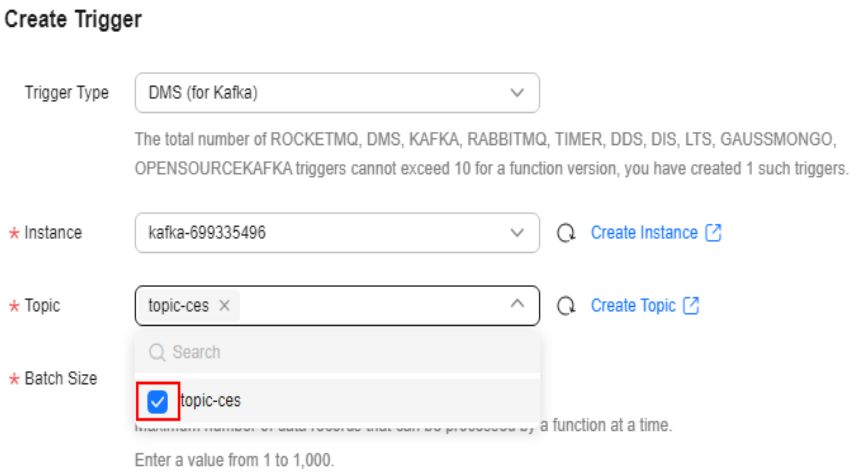
- **Password**: Enter the password of the instance if SSL has been enabled for it.

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

NOTE

- After VPC access is enabled, you need to configure corresponding subnet permissions for the Kafka security group. For details about how to configure VPC access, see **Configuring the Network**.
- 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.

**Figure 6-7** Selecting multiple topics




**----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 6-2** and click **Save**.

**Table 6-2** 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 Template | Select **DMS (for Kafka)** to use the built-in Kafka 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, **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**

# 6.5 Using a DIS Trigger

This section describes how to create a DIS trigger for a function, and configure a DIS event by using the built-in event template to trigger the function.

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

## Prerequisites

Before creating a trigger, ensure that you have prepared the following:

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

## Setting an Agency

Before creating a DIS trigger, set an agency to delegate FunctionGraph to access DIS. For details on how to create an agency, see **Configuring Agency Permissions**.

Since you did not specify an agency while creating the **HelloWorld** function, specify one first.

**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 change the agency to **serverless-trust** created in **Configuring Agency Permissions**.

**Step 4** Click **Save**.

**----End**

## Creating a DIS Trigger

**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 **Create Trigger**.

**Step 4** Set the following parameters:

- **Trigger Type**: Select **Data Ingestion Service (DIS)**.
- **Stream Name**: Select a DIS stream, for example, **dis-function**.
- **Starting Position**: Specify a position in the specified stream from which to start reading data.
  - **TRIM_HORIZON**: Data is read from the earliest valid records that are stored in the partition.
  - **LATEST**: Data is read just after the most recent record in the partition. This setting ensures that you always read the latest data.
- Data processing: **By Byte** and **By Batch**.

**Table 6-3** Data Processing

| Data processing mode | Description |
| --- | --- |
| By Byte | Configure **Max. Fetch Bytes** which indicates the maximum volume of data that can be fetched in each request. Only the records smaller than this value will be fetched. The value ranges from 0 KB to 4 MB. |
| By Batch | Configure **Batch Size** which indicates the maximum number of data records pulled at a time. Only the records smaller than this value will be fetched. The value ranges from 1 to 10,000. The total number of data records pulled at the same time cannot exceed 4 MB. |

NOTE

By default, users do not have permission to use the **By Batch** mode. To use it, **submit a service ticket** to add a whitelist.

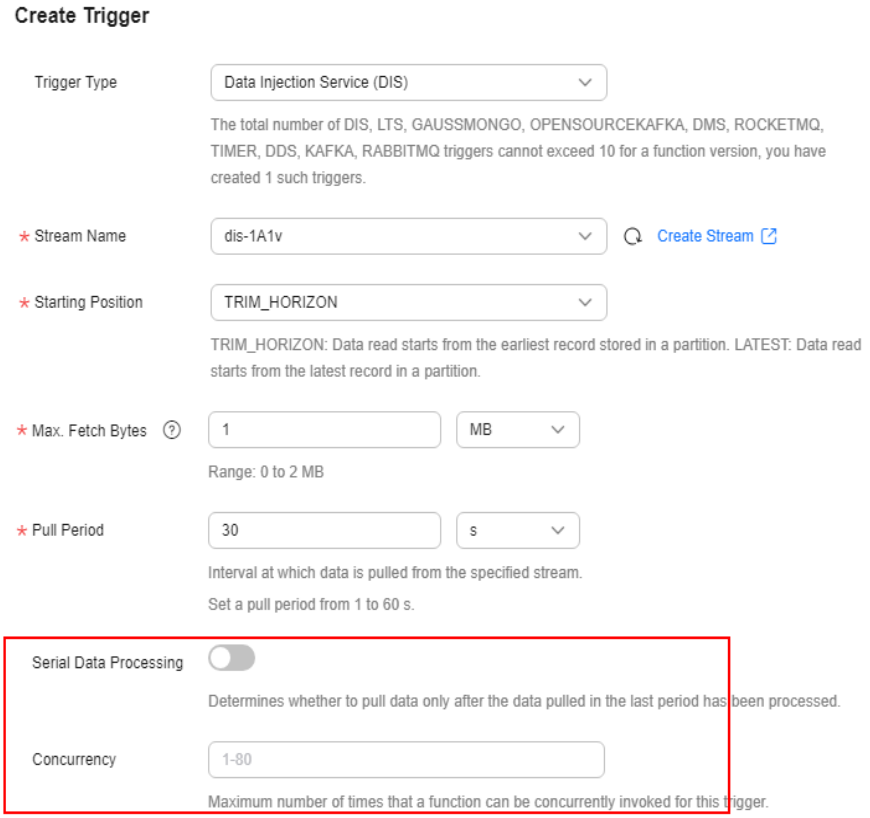
- **Pull Period**: Set a period for pulling data from the stream.
- **Serial Data Processing**: If this option is selected, FunctionGraph pulls data from the stream only after previous data is processed. If this option is not selected, FunctionGraph pulls data from the stream as long as the pull period ends.

NOTE

After **Serial Data Processing** is disabled, 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)

**Figure 6-8** Disabling serial data processing




**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/Batch Size**, **Pull Period**, and **Serial Data Processing** as required, and click **OK**.

**----End**

### Configuring a DIS 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 6-4** and click **Save**.

**Table 6-4** 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 Template | 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**

## 6.6 Using an SMN Trigger

This section describes how to create an SMN trigger and publish a message to trigger a function.

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

### Prerequisites

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

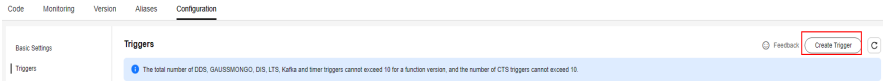
## Creating an SMN Trigger

**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 6-9** 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**.

Table 6-5 describes the parameters required for publishing a message.

**Table 6-5** Parameters required for publishing a message

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

NOTE

After a message is published, the function is triggered automatically. For details about example events, see **Supported Event Sources**.

## 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**

# 6.7 Using an LTS Trigger

This section describes how to create an LTS trigger for a function, and invoke the function when log events occur.

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

## Prerequisites

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

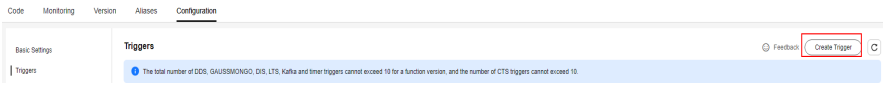
## Creating an LTS Trigger

**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 6-10** 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

NOTE

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.

**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 6-6** and click **Save**.

**Table 6-6** 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 Template | 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**

## 6.8 Using a 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.

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

### Prerequisites

You have created an agency on IAM. For details, see **Configuring Agency Permissions**.

## Creating a CTS 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** Set the following parameters:

- **Function Name**: Enter a function name, for example, **HelloWorld**.
- **Agency**: Select **Use no agency**.
- **Enterprise Project**: Select **default**.
- **Runtime**: Select **Python 2.7**.

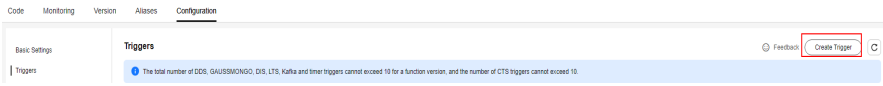
**Step 4** Click **Create Function**.

**Step 5** On the **Code** tab page, copy the following code to the code window and 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)
```

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

**Figure 6-11** Creating a trigger




**Step 7** Configure the trigger information.

**Table 6-7** Trigger information

| Parameter | Description |
| --- | --- |
| Trigger Type | Select **Cloud Trace Service (CTS)**. |
| Event Notification Name | Enter a notification name, for example, **Test**. |
| Service Type | Select **FunctionGraph**.<br>**NOTE**<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 more information about global cloud services, contact technical support. |
| Resource Type | Resource types supported by the selected service, such as triggers, instances, and functions. |
| Trace Name | Operations that can be performed on the selected resource type, such as creating or deleting a trigger. |

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

**----End**

## 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 6-8** and click **Save**.

**Table 6-8** 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 Template | 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**

# 6.9 Using a DDS Trigger

This section describes how to create a DDS trigger for a function, and invoke the function when a database table changes.

A function using a DDS trigger will be triggered every time a database table is updated. For details about the DDS event source, see **Supported Event Sources**.

## Prerequisites

Before creating a trigger, ensure that you have prepared the following:

- You have created a function. For details, see **Creating a Function from Scratch**.
- You have enabled VPC access for the function. For details, see **Configuring the Network**.
- You have created a DDS DB instance.
- You have created a DDS database.

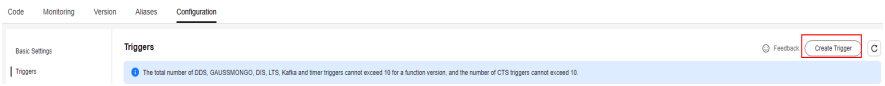
## Creating a DDS Trigger

**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 6-12** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **Document Database Service (DDS)**.
- **DB Instance**: Select a DDS DB instance.
- **Password**: Enter the password of DDS DB instance administrator **rwuser**.
- **Database**: Enter the name of a database. Note that **admin**, **local**, and **config** are reserved database names and cannot be used here.
- **Collection**: Enter the name of a database collection.
- **Batch Size**: Set the number of records to be read from the database at a time.

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

NOTE

After VPC access is enabled, you need to configure corresponding subnet permissions for the DDS security group. For details about how to configure VPC access, see **Configuring the Network**.

**----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 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 6-9** and click **Save**.

**Table 6-9** 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 Template | 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**

## 6.10 Using a GeminiDB Mongo Trigger

This section describes how to create and configure a GeminiDB Mongo trigger for a function.

A function using a GeminiDB Mongo trigger will be triggered every time a database table is updated. For details about the GeminiDB Mongo event source, see **Supported Event Sources**.

## Prerequisites

Before creating a trigger, ensure that you have prepared the following:

- You have created a function. For details, see **Creating a Function from Scratch**.
- You have enabled VPC access for the function. For details, see **Configuring the Network**.
- You have created a GeminiDB Mongo instance. For details, see **Buying a Replica Set Instance**.

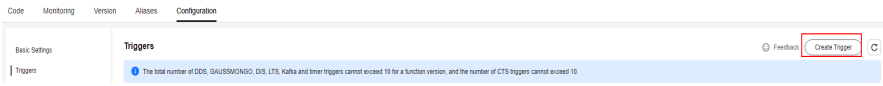
## Creating a GeminiDB Mongo Trigger

**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 6-13** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **GeminiDB Mongo**.
- **GeminiDB Mongo Instance**: Select a GeminiDB Mongo instance.
- **Password**: Enter the password of the GeminiDB Mongo instance administrator **rwuser**.
- **Database**: Enter the name of a GeminiDB Mongo database. Note that **admin**, **local**, and **config** are reserved database names and cannot be used here.
- **Collection**: Enter the name of a database collection.
- **Batch Size**: Set the number of records to be read from the database at a time.

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

NOTE

After VPC access is enabled, you need to configure corresponding subnet permissions for the GeminiDB Mongo security group. For details about how to configure VPC access, see **Configuring the Network**.

**----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 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 6-10** and click **Save**.

**Table 6-10** 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 Template | 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, **geminimongo-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**

# 6.11 Using an APIG Trigger

This section describes how to create an APIG trigger and call an API to trigger a function.

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

NOTE

APIG no longer provides shared gateways. Only customers who had registered before this feature was removed can continue using it.

## Prerequisites

You have created an API group, for example, **APIGroup_test**. For details, see **Creating an API Group**.

## Creating an APIG Trigger

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

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

**Step 3** Set the following parameters:

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

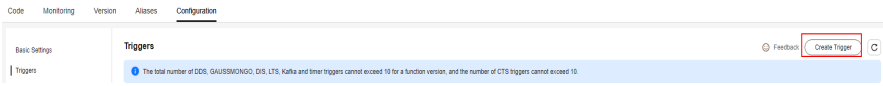
**Step 4** Click **Create Function**.

**Step 5** On the **Code** tab page, copy the following code to the code window 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**.

**Figure 6-14** Creating a trigger




**Step 7** Configure the trigger information.

**Table 6-11** Trigger information

| Parameter | Description |
| --- | --- |
| Trigger Type | Select **API Gateway (APIG)**. |
| API Name | Enter an API name, for example, **API_apig**. |
| API Group | An API group is a collection of APIs. You can manage APIs by API group.<br>Select **APIGroup_test**. |
| 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.<br>To ensure that the API can be called, select **RELEASE**. |

| Parameter | Description |
| --- | --- |
| Security Authentication | There are three authentication modes:<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>Select **None**. |
| Protocol | There are two types of protocols:<br>• HTTP<br>• HTTPS<br>Select **HTTPS**. |
| Timeout (ms) | Enter **5000**. |

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

NOTE

1. The URL of the APIG trigger is **https://0ed9f61512d34982917a4f3cfe8ddd5d.apig.example.example.com/apig**.
2. After the APIG trigger is created, an API named **API_apig** 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** Enter the URL of the APIG trigger in the address bar of a browser, and press **Enter**.

**Step 2** After the function is executed, a result is returned.

**----End**

## 6.12 Using an APIC Trigger

This section describes how to create an APIC trigger and call an API to trigger a function. (This is only available in AP-Singapore.)

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

### Prerequisites

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, and choose **Functions** > **Function List** in the navigation pane.

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

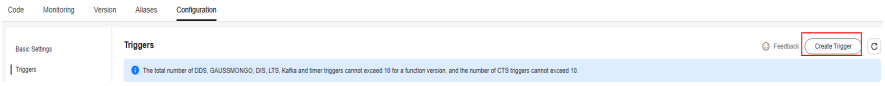
**Step 3** Set the following parameters:

- **Function Name**: Enter a function name, for example, **apig**.
- **Agency**: Select **Use no agency**.
- **Enterprise Project**: Select **default**.
- For **Runtime**, select **Node.js 10.16**.

**Step 4** Click **Create Function**.

**Step 5** Choose **Configuration** > **Triggers** and click **Create Trigger**.

**Figure 6-15** Creating a trigger




**Step 6** Configure the trigger information.

**Table 6-12** Trigger information

| Parameter | Description |
| --- | --- |
| Trigger Type | Select **API Connect (APIC)**. |
| API Instance | Select an instance. If no instance is available, click **Create Instance**. |
| API Name | Enter an API name, for example, **API_apic**. |
| API Group | An API group is a collection of APIs. You can manage APIs by API group.<br>Example: **DEFAULT**. |
| 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.<br>To ensure that the API can be called, select **RELEASE**. |

| Parameter | Description |
| --- | --- |
| Security Authentication | There are three authentication modes:<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>Select **None**. |
| Protocol | There are two types of protocols:<br>• HTTP<br>• HTTPS<br>Select **HTTPS**. |
| Timeout (ms) | Enter **5000**. |

**Step 7** 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 **nodejs-test** function.

**Step 3** On the displayed function details page, click the **Logs** tab to query the function running logs.

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

**----End**

# 6.13 Using a DMS (for RabbitMQ) Trigger

This section describes how to create a DMS (for RabbitMQ) trigger for a function. Currently, only the fan-out exchange mode is supported. After 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. For details about the DMS (for RabbitMQ) event source, see **Supported Event Sources**.

## Prerequisites

- You have created a function. For details, see **Creating a Function from Scratch**.
- You have enabled VPC access. For details, see **Configuring the Network**.
- A RabbitMQ instance has been created. For details, see **Buying an Instance**.
- A virtual host, exchange, and queue have been created.
  a. To create a virtual host, see **Creating a RabbitMQ Virtual Host**.
  b. To create an exchange, see **Creating a RabbitMQ Exchange**.
  c. To create a queue, see **Creating a RabbitMQ Queue**.
  d. 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.
  a. In the **Network** section on the **Basic Information** tab page, click the name of the security group.
  b. Click the **Inbound Rules** tab to view the inbound rules of the security group.
     i. 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.
     ii. 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.

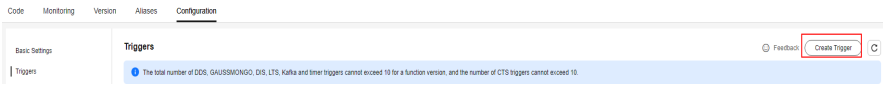
## Creating a RabbitMQ Trigger

**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 6-16** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **DMS (for RabbitMQ)**.
- ***Instance**: Select a RabbitMQ instance.
- ***Password**: Enter the password of the RabbitMQ instance.
- *Exchange: Enter the name of a created exchange. For details, see **Creating a RabbitMQ Exchange**.
- Virtual Host: Enter the name of the virtual host you have created. For details, see **Creating a RabbitMQ Virtual Host**.
- ***Batch Size**: Set the number of messages to be retrieved from a topic each time.

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

**----End**

NOTE

After VPC access is enabled, you need to configure corresponding subnet permissions for the RabbitMQ security group. For details about how to configure VPC access, see **Configuring the Network**.

## 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 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 6-13** and click **Save**.

**Table 6-13** 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 Template | Select **DMS (for RabbitMQ)** to use the built-in RabbitMQ 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, **kafka-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**

# 6.14 Using an Open-Source Kafka Trigger

This section describes how to create an open-source Kafka trigger and configure an event to trigger a function.

If you use an open-source Kafka trigger for a function, FunctionGraph periodically polls messages from a specific topic in Kafka and passes the messages as an input parameter to invoke the function.

NOTE

- For details about the differences between DMS for Kafka and open-source Kafka, see **Comparing DMS for Kafka and Open-Source Kafka**.
- In cases of Kafka data processing failure, the Kafka trigger will discard records that are larger than 6 MB.

## Prerequisites

Before creating a trigger, ensure that you have prepared the following:

- You have created a function.
- You have enabled VPC access for the function. For details, see **Configuring the Network**.

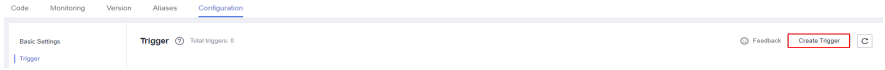
## Creating an Open-Source Kafka Trigger

**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 6-17** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **Kafka (OPENSOURCEKAFKA)**.
- **Connection Address**: Addresses of brokers running Kafka. Separate the addresses with commas (,).
- **Topic**: Enter one or more topics.
- **Batch Size**: Maximum number of data records that can be processed by the function at a time.

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

NOTE

The network configuration must be the same as that of the ECS where Kafka is deployed, including the VPC and subnet.

**----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 a trigger cannot be disabled, 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 6-14** and click **Save**.

**Table 6-14** 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 Template | Select **Kafka (OPENSOURCEKAFKA)** 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**. |

| Parameter | Description |
| --- | --- |
| 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**

# 6.15 Cron Expressions for a Function Timer Trigger

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

- @every format

  The format is "@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 6-15**.

  **Table 6-15** Example expressions

  | Expression | Meaning |
  | --- | --- |
  | @every 30m | Triggers a function every 30 minutes. |
  | @every 1.5h | Triggers a function every 1.5 hours. |
  | @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 6-16** describes the fields in a standard cron expression.

  **Table 6-16** Parameter description

  | Parameter | Mandatory | Value Range | Special Characters Allowed |
  | --- | --- | --- | --- |
  | CRON_TZ | No. If this parameter is not set, the region's time zone is used by default. | - | - |
  | Seconds | Yes | 0–59 | , - * / |
  | Minutes | Yes | 0–59 | , - * / |

| Parameter | Mandatory | Value Range | Special Characters Allowed |
| --- | --- | --- | --- |
| Hours | Yes | 0–23 | , - * / |
| Day-of-month | Yes | 1–31 | , - * ? / |
| Month | Yes | 1–12 or Jan–Dec. The value is case-insensitive, as shown in **Table 6-17**. | , - * / |
| Day-of-week | No | 0–6 or Sun–Sat. The value is case-insensitive, as shown in **Table 6-18**. **0** means Sunday. | , - * ? / |

**Table 6-17** Value description of the month field

| Month | Digit | Abbreviation |
| --- | --- | --- |
| January | 1 | Jan |
| 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 6-18** 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 6-19** describes the special characters that can be used in a cron expression.

**Table 6-19** Special character description

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

| Special Character | Meaning | Description |
| --- | --- | --- |
| / | 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 6-20** describes several example cron expressions.

**Table 6-20** 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 * |

NOTE

If no cron expression is set, the region's time zone is used by default. 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 * ***. The time zone expression varies depending on the region.

# 6.16 Using an EG Trigger

# 6.16.1 Creating an EG Trigger (RocketMQ Custom Event Source)

## Prerequisites

Before creating a trigger, ensure that you have completed the following operations:

- You have created and enabled VPC. For details, see **Creating a Function from Scratch** and **Configuring the Network**.
- You have created an EG event channel. For details, see **Creating an EG Event Channel**.
- 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**.

NOTE

- Currently, EG triggers are available only in **AP-Bangkok**, **AP-Singapore**, **AF-Johannesburg**, **LA-Mexico City2**, and **LA-Santiago**.
- The VPC bound to the function must be the same as the VPC of the RocketMQ instance. After VPC access is enabled, you need to configure corresponding subnet permissions for the RocketMQ security group.

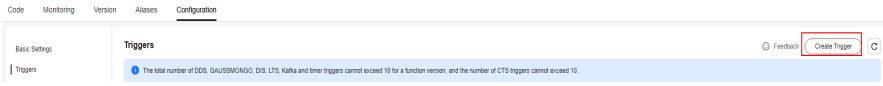
## Creating an EG Trigger

**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 6-18** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **DMS (for HC.RocketMQ)**.
- **Trigger Name**: Enter a trigger name.
- **Event Channel**: Select an existing channel.
- ***Instance**: Select a RocketMQ instance.
- **Topic**: Select a RocketMQ topic.
- **Consumer Group**: Select a RocketMQ consumer group.
- **Username**: This parameter is mandatory when ACL is enabled for the RocketMQ instance. Username used for accessing the RocketMQ instance.
- **Password**: This parameter is mandatory when ACL is enabled for the RocketMQ instance. Password for connecting to the RocketMQ instance.

**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 6-21** and click **Save**.

**Table 6-21** 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 Template | Select **DMS (for HC.RocketMQ)** and use the built-in HC.ROCKETMQ 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, **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**

# 6.16.2 Creating an EG Trigger (OBS Application Service)

## Prerequisites

You have created an OBS bucket, for example, **eventbucket**. For details, see **Creating a Bucket**.

NOTE

Currently, EG triggers are available only in **AP-Bangkok**, **AP-Singapore**, **AF-Johannesburg**, **LA-Mexico City2**, and **LA-Santiago**.

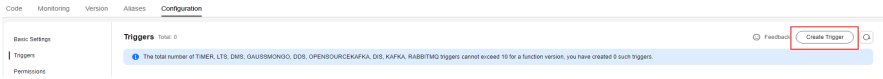
## Creating an EG Trigger

**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 6-19** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **Object Storage Service (OBS)**.
- **Trigger Name**: Enter a trigger name.
- **Bucket Name**: Select an OBS bucket.
- **Events**: Select the required event type.
- **Prefix**: 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.
- **Suffix**: 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**: Indicates whether to encode an object.

  NOTE

  If you select **Delete objects without specifying version** for **Event Type**, enable versioning. For details, see **Deleting Objects from a Bucket with Versioning Enabled**.

**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 6-22** and click **Save**.

**Table 6-22** 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 Template | 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**

## 6.16.3 Creating an EG Trigger (Cloud Service)

NOTE

Currently, EG triggers are available only in **AP-Bangkok**, **AP-Singapore**, **AF-Johannesburg**, **LA-Mexico City2**, and **LA-Santiago**.

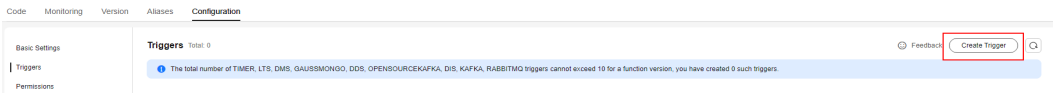
### Creating an EG Trigger

**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**. This section uses OBS as an example.

**Figure 6-20** Creating a trigger




**Step 4** Set the following parameters:

- **Trigger Type**: Select **Object Storage Service (OBS)**.
- **Trigger Name**: Enter a trigger name.
- **Events**: Select the required event type.

**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 6-23** and click **Save**.

**Table 6-23** 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 Template | Select **Blank Template**. For details about the code, see "Cloud Service Event Sources" 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**

# 7 Invoking the Function

## 7.1 Synchronous Invocation

When triggering a function, clients wait for the result before proceeding.

Currently, functions with APIG (shared), APIC, and APIG (dedicated) triggers are executed synchronously. You can also use the **synchronous execution** API to trigger functions.

In this scenario, a function is executed for up to 15 minutes.

## 7.2 Asynchronous Invocation

When a client triggers a function, FunctionGraph persists the request and sends a response immediately to the client. The client proceeds without waiting for the execution result. You cannot know the result in real time. FunctionGraph queues the asynchronous requests and processes them when the server is idle. To obtain asynchronous processing results or to retry when an asynchronous request fails, **configure asynchronous settings**.

- The following triggers are invoked asynchronously by default and the invocation mode cannot be changed.

  **Table 7-1** Invocation mode

| Event Source | Invocation Mode |
| --- | --- |
| SMN | Asynchronous |
| OBS | Asynchronous |
| DIS | Asynchronous |
| Timer | Asynchronous |
| LTS | Asynchronous |
| DDS | Asynchronous |

| Event Source | Invocation Mode |
| --- | --- |
| DMS for Kafka | Asynchronous |
| DMS for RabbitMQ | Asynchronous |
| GeminiDB Mongo | Asynchronous |

- APIG, APIG (dedicated), and APIC triggers can be configured for asynchronous invocation on their console. You can also use the **asynchronous execution** API instead. In this scenario, the maximum execution duration of a function is 12 hours (configured in the whitelist).

  NOTE

  If the E2E function execution latency exceeds 90s, asynchronous invocation is recommended. If synchronous invocation is used, no responses can be received after 90s due to gateway restrictions.

**Example**

The following procedure uses the APIG trigger of a function as an example.

1. Go to the function details page, and choose **Configuration** > **Triggers**.
2. Click the APIG trigger name to go to the APIG console.

   **Figure 7-1** Clicking a trigger name

3. Click **Edit** in the upper right.

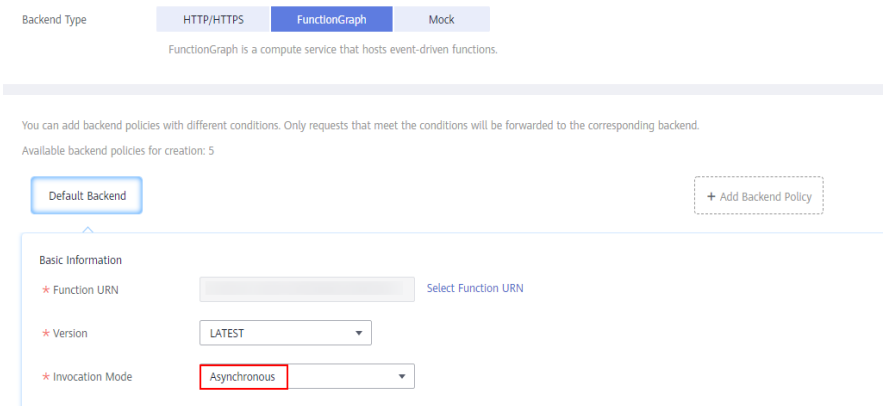
   **Figure 7-2** Clicking Edit

4. Click **Next** until the **Define Backend Request** page is displayed. Then change **Invocation Mode** to **Asynchronous**.

**Figure 7-3** Changing the invocation mode




5. Click **Finish** to save the settings.

# 7.3 Retry Mechanism

If synchronous or asynchronous invocation fails, do as follows:

- Synchronous invocation

  Try again.
- Asynchronous invocation

  You can set the maximum number of retries and the maximum message validity period (up to 24 hours) by referring to **Configuring Asynchronous Execution Notification**. 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.

# 8 Monitoring

## 8.1 Metrics

### 8.1.1 Function Monitoring

FunctionGraph is interconnected with Cloud Eye, allowing you to view function metrics without the need for 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 day, last 3 days, or custom), and check the running status of the function.

   NOTE

   The following metrics are displayed: invocations, errors, duration (maximum, average, and minimum durations), throttles, and instance statistics.

#### Metric Description

**Table 8-1** describes the function metrics.

**Table 8-1** 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. |
| 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. |
| 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. |
| Throttles | Count | Number of times that FunctionGraph throttles your functions due to the resource limit. |
| Instance Statistics | Count | Numbers of concurrent requests and reserved instances. |

## 8.1.2 Function Metrics

### Introduction

This section describes the FunctionGraph namespaces, function metrics, and dimensions reported to Cloud Eye. You can view function metrics and alarms by using the Cloud Eye console or calling APIs.

### Namespaces

SYS.FunctionGraph

## Function Metrics

**Table 8-2** Function metrics

| Metric ID | Metric Name | Description | Value Range | Monitored Object | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|
| count | Invocations | Number of function invocations<br>Unit: Count | ≥ 0 counts | Functions | 5 |
| failcount | Errors | Number of invocation errors<br>The following errors are included:<br>• Function request error (causing an execution failure and returning error code 200)<br>• Function syntax or execution error<br>Unit: Count | ≥ 0 counts | Functions | 5 |
| failRate | Error rate | Percentage of invocation errors to the total invocations<br>Unit: % | 0% ≤X≤ 100% | Functions | 5 |
| rejectcount | Throttles | Number of function throttles<br>That is, the number of times that FunctionGraph throttles your functions due to the resource limit.<br>Unit: Count | ≥ 0 counts | Functions | 5 |

| Metric ID | Metric Name | Description | Value Range | Monitored Object | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|
| concurrency | Number of concurrent requests | Maximum number of concurrent requests during function invocation.<br>Unit: Count | ≥ 0 counts | Functions | 5 |
| reservedinstancenum | Number of reserved instances | Number of reserved instances<br>Unit: Count | ≥ 0 counts | Functions | 5 |
| duration | Average duration | Average duration of function invocation<br>Unit: ms | ≥ 0 ms | Functions | 5 |
| maxDuration | Maximum duration | Maximum duration of function invocation<br>Unit: ms | ≥ 0 ms | Functions | 5 |
| minDuration | Minimum duration | Minimum duration of function invocation<br>Unit: ms | ≥ 0 ms | Functions | 5 |
| systemErrorCount | System errors | Number of errors in function requests that result in failed executions. Unit: Count | ≥ 0 | Functions | 5 |
| functionErrorCount | Function errors | Number of syntax and execution errors. Unit: Count | ≥ 0 | Functions | 5 |
| payPerUseInstance | Number of elastic instances. | Number of elastic instances. Unit: Count | ≥ 0 | Functions | 5 |

**Table 8-3** Flow metrics

| Metric ID | Metric Name | Description | Value Range | Monitored Object | Monitoring Period of Raw Data (Minute) |
|---|---|---|---|---|---|
| toalCount | Invocations | Number of flow invocations<br>Unit: Count | ≥ 0 counts | Flows | 1 |
| errorCount | Errors | Number of invocation errors<br>Unit: Count | ≥ 0 counts | Flows | 1 |
| running | Running Workflows | Number of running flows<br>Unit: Count | ≥ 0 counts | Flows | 1 |
| rejectCount | Throttles | Number of flow throttles<br>Unit: Count | ≥ 0 counts | Flows | 1 |
| averageDuration | Average Duration | Average duration of flow invocation<br>Unit: ms | ≥ 0 ms | Flows | 1 |

### Dimensions

| Key | Value |
|---|---|
| package-functionname | *App name-Function name*<br>Example: default-myfunction_Python |
| graph_name | Flow name |
| projectId | Project ID of the tenant |

## 8.1.3 Creating an Alarm Rule

After creating a function and trigger, you can monitor the invocation and running statuses of the function in real time.

### 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**.

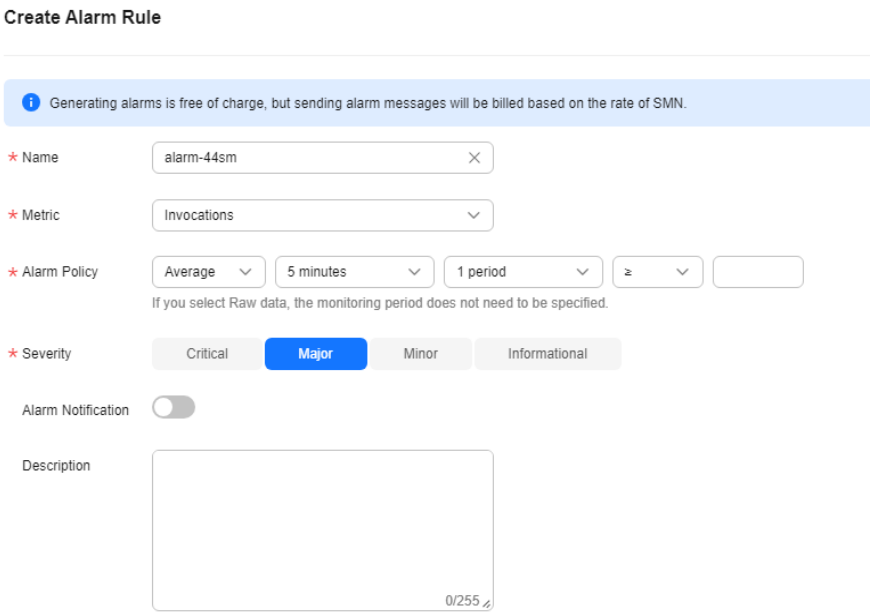
**Step 2** Click the name of the desired function.

**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 8-1**.

**Figure 8-1** Creating an alarm rule




**Step 6** Enter a rule name and click **OK**.

**----End**

**CAUTION**

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.

## Function Metrics

**Table 8-4** lists the function metrics that can be monitored by Cloud Eye.

**Table 8-4** Function metrics

| Metric | Display Name | Description | Unit | Upper Limit | Lower Limit | Recommended Threshold | Value Type | Dimension |
|---|---|---|---|---|---|---|---|---|
| count | Invocations | Number of function invocations | Count | - | 0 | - | int | package-functionname |
| failcount | Errors | Number of invocation errors | Count | - | 0 | - | int | package-functionname |
| rejectcount | Throttles | Number of function throttles | Count | - | 0 | - | int | package-functionname |
| duration | Average Duration | Average duration of function invocation | ms | - | 0 | - | float | package-functionname |
| maxDuration | Maximum Duration | Maximum duration of function invocation | ms | - | 0 | - | float | package-functionname |
| minDuration | Minimum Duration | Minimum duration of function invocation | ms | - | 0 | - | float | package-functionname |

| Metric | Display Name | Description | Unit | Upper Limit | Lower Limit | Recommended Threshold | Value Type | Dimension |
|---|---|---|---|---|---|---|---|---|
| concurrency | Concurrency | The number of requests that can be processed concurrently | Count | - | 0 | - | int | package-functionname |
| payPerUseInstance | Elastic Instances | Number of instances actually used by a function after reserved instances are excluded | Count | - | 0 | - | int | package-functionname |
| failRate | Error Rate | Percentage of errors to the total invocations of a function | % | - | 0 | - | float | package-functionname |
| functionErrorCount | Function Error Count | Number of function errors that occur during invocation | Count | - | 0 | - | float | package-functionname |
| memoryUsed | Memory | Memory used by the function | MB | - | 0 | - | float | package-functionname |

| Metric | Display Name | Description | Unit | Upper Limit | Lower Limit | Recommended Threshold | Value Type | Dimension |
|---|---|---|---|---|---|---|---|---|
| duration_p500 | Duration (P50) | P50 execution duration of the function | ms | - | 0 | - | float | package-functionname |
| duration_p800 | Duration (P80) | P80 execution duration of the function | ms | - | 0 | - | float | package-functionname |
| duration_p950 | Duration (P95) | P95 execution duration of the function | ms | - | 0 | - | float | package-functionname |
| duration_p990 | Duration (P990) | P990 execution duration of the function | ms | - | 0 | - | float | package-functionname |
| duration_p999 | Duration (P999) | P999 execution duration of the function | ms | - | 0 | - | float | package-functionname |
| instances | Instances | Number of instances for function invocation | Count | - | 0 | - | int | package-functionname |

| Metric | Display Name | Description | Unit | Upper Limit | Lower Limit | Recommended Threshold | Value Type | Dimension |
|---|---|---|---|---|---|---|---|---|
| systemErrorCount | System Errors | Number of system errors that occur during function invocation | Count | - | 0 | - | int | package-functionname |
| reservedinstancenum | Reserved Instances | Number of reserved instances | Count | - | 0 | - | int | package-functionname |
| functionCost | Resource Usage | Resources used by the function (Memory x Duration) | MB | - | 0 | - | float | package-functionname |

**Table 8-5** Flow metrics

| Metric | Display Name | Description | Unit | Upper Limit | Lower Limit | Recommended Threshold | Value Type | Dimension |
|---|---|---|---|---|---|---|---|---|
| ExecutionsStarted | Executions Started | Number of instances that start to execute | Count | - | 0 | - | int | ProjectId |

| Metric | Display Name | Description | Unit | Upper Limit | Lower Limit | Recommended Threshold | Value Type | Dimension |
|---|---|---|---|---|---|---|---|---|
| ExecutionsAborted | Executions Aborted | Number of instances that are aborted | Count | - | 0 | - | int | ProjectId |
| ExecutionsTimedOut | Executions Timed Out | Number of instances that run out of time | Count | - | 0 | - | int | ProjectId |
| ExecutionsSucceeded | Executions Succeeded | Number of instances that execute successfully | Count | - | 0 | - | int | ProjectId |
| ExecutionsFailed | Executions Failed | Number of instances that execute unsuccessfully | Count | - | 0 | - | int | ProjectId |

# 8.2 Logs

## 8.2.1 Querying Function Logs

FunctionGraph is interconnected with LTS, allowing you to view function logs without the need for any configurations.

### Viewing Function Logs

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 **Online Debugging**.

  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 **Managing Function Logs**.

### Downloading Logs

After querying the logs of a function version within a specified date range, you can download the logs for further analysis.

**NOTE**

- A maximum of 5000 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.

## 8.2.2 Managing Function Logs

**NOTICE**

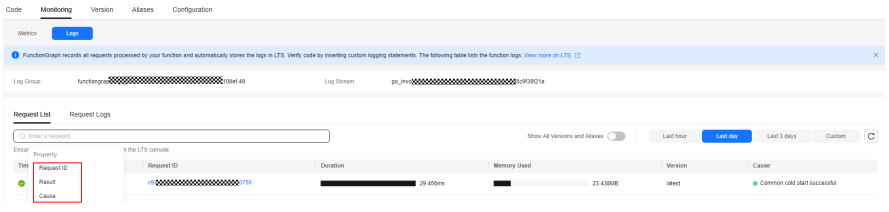
FunctionGraph v1 supports log management by using AOM or LTS.

FunctionGraph v2 supports log management by using LTS only.

- Using AOM to manage function logs


- Using LTS to manage function logs

**Figure 8-2** Logs page




### Using AOM to Manage Function Logs

**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** > **Logs**. On the displayed page, enter search criteria.

NOTE

- You can:

  1. Enter a keyword for exact search. A keyword is a word between two adjacent delimiters.

  2. Enter a keyword with wildcards for fuzzy match. Example: ***ROR***, **ERR***, or **ER*OR**.

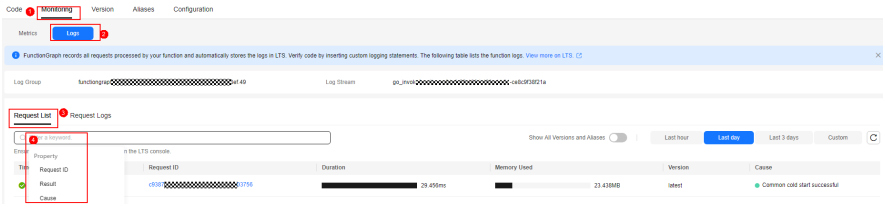
  3. Enter a phrase for exact search. Example: **Start to refresh alm Statistic**.

  4. Use AND (&&) or OR (||) to combine keywords to search. Example: **query&&logs** or **query||logs**.

- You can select **Last 30 minutes**, **Last hour**, **Last day**, and **Custom** (up to one month, for example, 2022/04/01 16:34:48–2022/05/01 16:34:48).
- You can query logs by version.

**Step 4** Click  to search for logs.

**Figure 8-3** Querying logs




NOTE

The log query result contains time, request ID, invocation result, duration, memory, and version.

**Step 5** Perform the following operations if needed:

1. Search for logs by keyword.

2. Filter logs by status: **Error**, **Info**, **Error & Warning**, and **Error & Warning & Info**.

3. View logs in full screen.

4. **Download logs**.

**Figure 8-4** Other operations


**----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 a Log Group and Log Stream**.

NOTE

- 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.

**Step 1** Enable LTS.

Enabling LTS in FunctionGraph v1: On the **Logs** tab page, click **Enable LTS to Manage Function Logs**.

NOTE

In FunctionGraph v1, you can switch back to AOM. In that case, AOM will take over LTS to manage function logs. You will be billed for logs generated on a pay-per-use basis.

**Enabling LTS in FunctionGraph v2**: On the **Logs** tab page, click **Enable LTS**. Click **OK**. LTS is enabled successfully.

NOTE

In FunctionGraph v2, only LTS can be used to manage function logs.

**Step 2** Set filter criteria.

- **Request List**: Filter requests by request ID, result (success or failure), or cause (initialization failed, load failed, system error, timed out, out of memory, out of disk space, or code error).
- **Request Log**: Filter logs by keyword, request ID, or instance ID.

**Table 8-6** 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 8-7** Cause analysis

| 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.
- Max. 10 MB logs can be retained for common instances during initialization. When this limit is reached, the latest logs replace the old ones.

**----End**

## Downloading Logs

NOTICE

- Currently, logs can be downloaded only when you use AOM for log management.
- FunctionGraph v1 allows you to manage function logs using AOM.
- FunctionGraph v2 allows you to manage function logs using LTS, but does not support log download.

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

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

**Step 3** Click **Monitoring**.

**Step 4** On the **Logs** page, select a version and time range, and click **Download Log**.

NOTE

A maximum of 5000 logs can be downloaded at a time. When querying logs, select a proper time range to avoid log loss.

**----End**

# 8.3 Tracing

NOTE

Currently, this feature is available only in CN East-Shanghai1 and CN North-Beijing4 regions.

## 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.

## 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

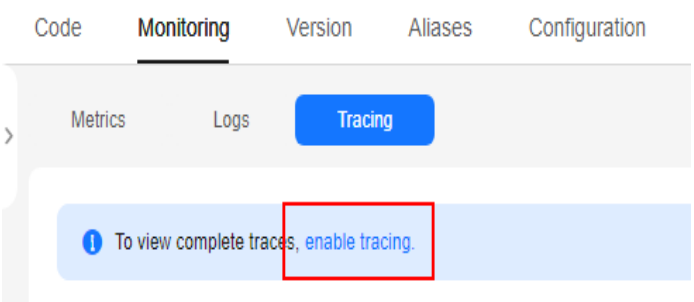
**Step 1** Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.

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

**Step 3** Choose **Monitoring** > **Tracing**.

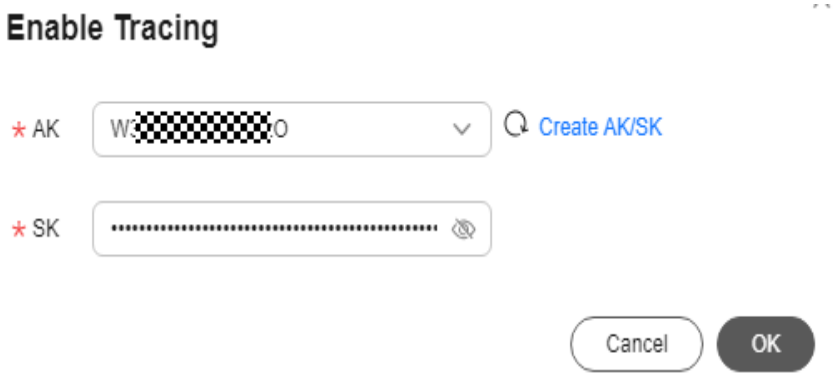
**Step 4** Click **enable tracing**.

**Figure 8-5** Enabling tracing




**Step 5** The system automatically obtains an access key (AK/SK) from APM, as shown in **Figure 8-6**.

**Figure 8-6** 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

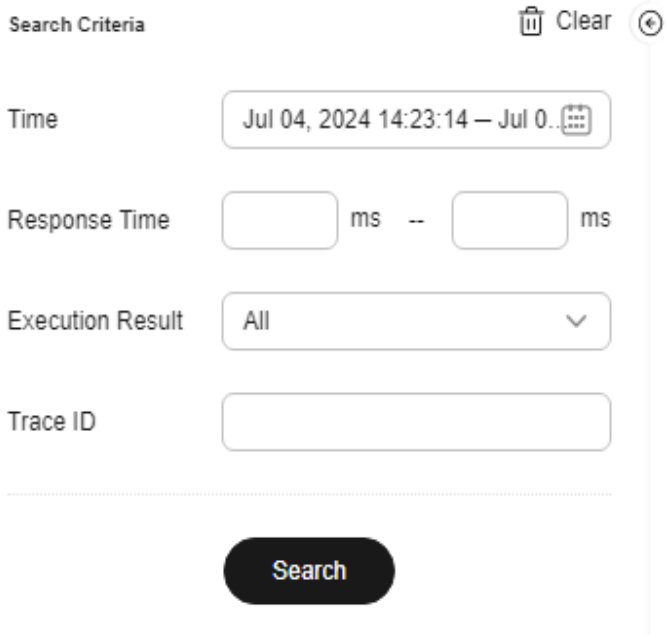
**Step 1** Return to the FunctionGraph console, choose **Functions** > **Function List** in the navigation pane, and click the name of a function with tracing enabled. The function details page is displayed.

**Step 2** On the **Monitoring** tab page, choose **Tracing**.

**Step 3** Set search criteria and click **Search**.

**Figure 8-7** 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 8-8** Clicking a trace name

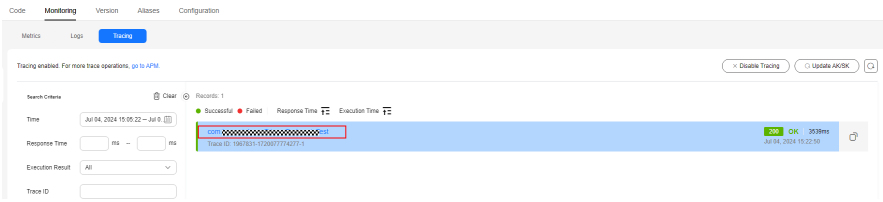
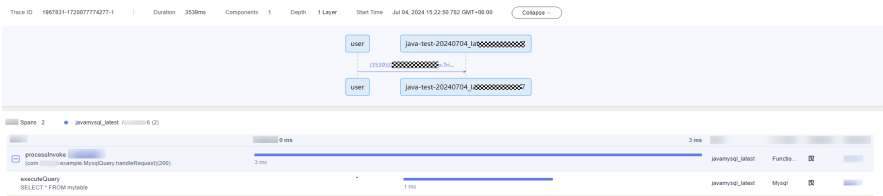


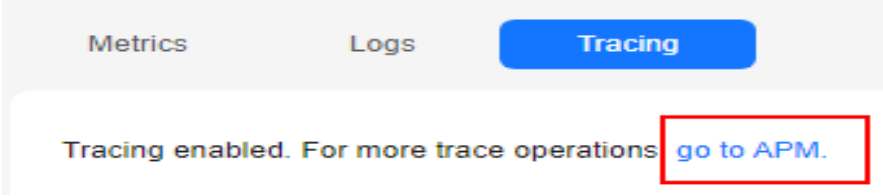
**Figure 8-9** Details about a trace



Viewing information about all traces: Click **go to APM** to perform complete path analysis and other operations.

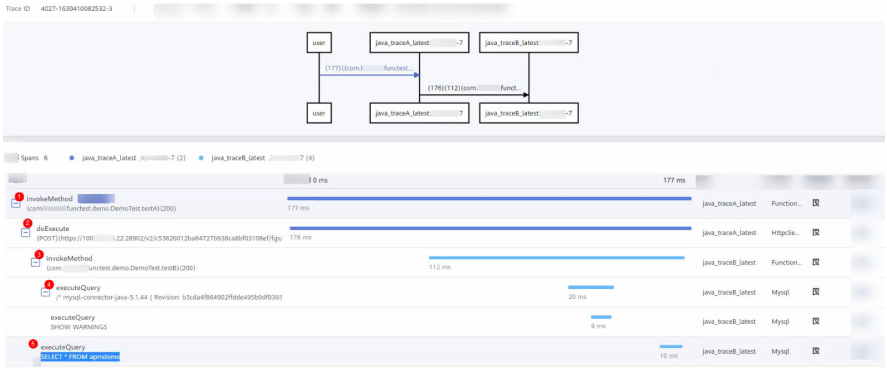
**Figure 8-10** Going to APM




**Example**

1. The following shows how function A (**DemoTestA**) invokes function B (**DemoTestB**) by sending an HTTP request.

   **Figure 8-11** 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**

# 9 Function Management

## Overview

Function is a combination of code, runtime, resources, and settings required to achieve a specific purpose. It is the minimum unit that can run independently. A function can be triggered by triggers and automatically schedule required resources and environments to achieve expected results.

## Exporting a Function

You can export the functions that you created.

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

**Step 2** Click a function name.

**Step 3** On the displayed function details page, choose **Operation** > **Export Function** in the upper right corner.

NOTE

- A user can export only one function at a time.
- The exported function resource package cannot exceed 50 MB.
- The name of the exported function resource package is in the format of *function name*+*MD5 value of function code*.zip.
- The exported function resource package does not include alias information.
- If a function is disabled or enabled, all versions of the function will be disabled or enabled.

**----End**

## Disabling a Function

Disabled functions can no longer be executed.

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

**Step 2** Click the name of the function you want to disable.

**Step 3** On the displayed function details page, click **Disable Function** in the upper right corner.

**Step 4** On the displayed page, click **Yes**. The function is disabled.

NOTE

- 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.

**----End**

## Enabling a Function

Disabled functions can be enabled again as required.

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

**Step 2** Click the name of the function you want to enable.

**Step 3** On the displayed function details page, click **Enable Function** in the upper right corner.

**----End**

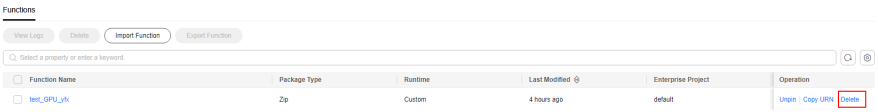
## Deleting a Function

You can delete unused functions to release resources.

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

**Step 2** In the **Function List**, locate the row that contains the target function and click **Delete** in the operation column. In the displayed dialog box, enter **DELETE** and click **OK**.

**Figure 9-1** Deleting a function




**----End**

# 10 Dependency Management

## 10.1 Configuring Dependency Packages

### Overview

Generally, the code of a function consists of public libraries and service logic. The public libraries can be packaged as a dependency and shared among functions, reducing the size of the function code package for easy deployment and update.

FunctionGraph also provides some **public dependencies**, which are cached internally for quick loading. These dependencies are recommended.

FunctionGraph enables you to manage dependencies in a unified manner. You can upload dependencies from a local path, or through OBS if they are too large, and specify names for them. Dependencies can be iterated. Each dependency can have multiple versions.

For details, see **How Do I Create Function Dependencies?**

NOTE

- The name of each file in the dependency package cannot end with a tilde (~).
- A dependency package can contain up to 30,000 files.
- If your function uses a large private dependency, increase the execution timeout by choosing **Configuration** > **Basic Settings** on the function details page.

### Creating a Dependency

**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 following parameters:

**Table 10-1** Dependency configuration parameters

| Parameter | Description |
| --- | --- |
| Name | Dependency name. |
| Code Entry Mode | Upload a ZIP file directly or upload a file from OBS.<br>• **Upload ZIP**: Click **Select File** to upload one.<br>• **Upload from OBS**: Specify an OBS link URL. For details about how to obtain the URL, see **Accessing an Object Using Its URL**. |
| Runtime | Select a runtime. |
| Description | Description of the dependency. This parameter is optional. |

**Step 4** Click **OK**. By default, a new dependency is version **1**.

**Step 5** Click the dependency name, and view all versions and related information on the displayed page. Each dependency can have multiple versions.

- To create a dependency version, click **Create Version** in the upper right corner of the page.
- To view the address of a version, click the version.
- To delete a version, click the delete icon in the same row.

**Figure 10-1** Deleting a dependency version


**----End**

## Configuring Dependencies for a Function

**Step 1** Return 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** On the displayed **Select Dependency** dialog box, select dependencies and click **OK**.

**Table 10-2** Dependency configuration

| Parameter | Description |
| --- | --- |
| Runtime | Runtime of this function. It cannot be changed. |
| Type | Add a **Public** or **Private** dependency. |
| Name | Select a dependency. |

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

NOTE

- You can add a maximum of 20 dependencies for a function.
- Except your private dependencies, FunctionGraph provides some public dependencies, which you can choose when creating a function.

**----End**

## Deleting a Dependency

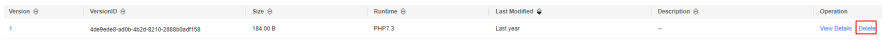
To delete a dependency, just delete all of its versions.

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

**Step 2** Click the name of the target dependency to go to the **Versions** page.

**Step 3** Click the delete icon in the row of a version. Repeat this operation if the dependency has multiple versions.

**Figure 10-2** Deleting a dependency version




NOTE

Dependencies referenced by functions cannot be deleted.

**----End**

# 10.2 Dependent Libraries

### Supported Dependent Libraries

FunctionGraph supports both standard and third-party libraries.

- Standard libraries

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

- Supported non-standard libraries

  FunctionGraph provides built-in third-party components listed in **Table 10-3** and **Table 10-4**. You can import these libraries to your inline code in the same way as you import standard libraries.

**Table 10-3** 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 |

**Table 10-4** Non-standard libraries supported by the Python runtime

| Module | Usage | Version |
| --- | --- | --- |
| dateutil | Date and time processing | 2.6.0 |
| requests | HTTP library | 2.7.0 |
| httplib2 | httpclient | 0.10.3 |

| Module | Usage | Version |
| --- | --- | --- |
| numpy | Mathematical computing | For pip 2.7, numpy==1.16.6.<br>For pip 3.10, numpy==1.24.2.<br>For pip 3.9, numpy==1.18.5.<br>For pip 3.6, numpy==1.18.5. |
| redis | Redis client | 2.10.5 |
| obsclient | OBS client | 3.0.3 |
| smnsdk | SMN access | 1.0.1 |

- Other third-party libraries (FunctionGraph has no built-in non-standard third-party libraries except those listed in the preceding table.)

  Package the dependency third-party libraries and upload them to an OBS bucket or on the function details page. These libraries will then be used in your function code.

**Importing Dependent Libraries**

Importing a dependency for Python:

```
from com.obs.client.obs_client import ObsClient
```

Importing a dependency for Node.js:

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

For standard libraries and supported non-standard libraries, you can directly use them in your function.

For non-standard third-party libraries that are not provided by FunctionGraph, you can use them by performing the following steps:

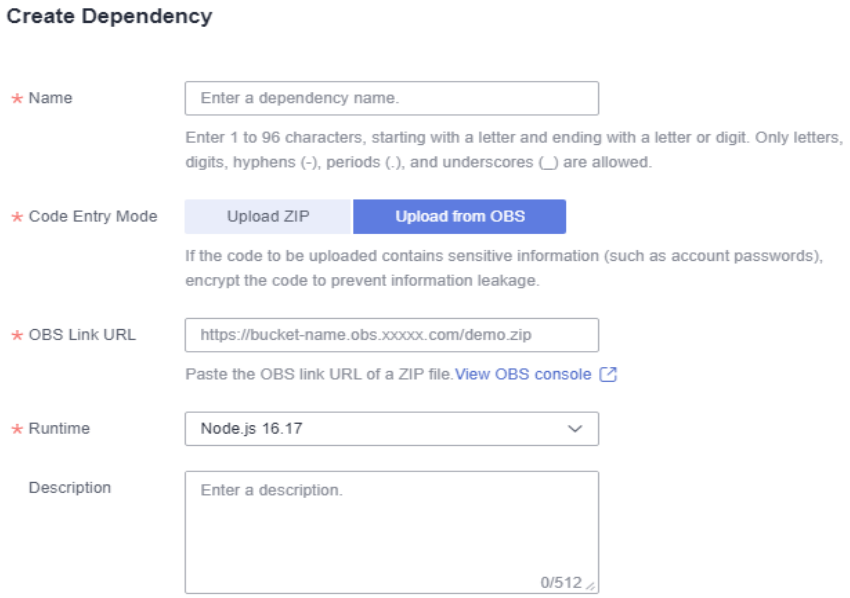
1. Package the dependent libraries into a ZIP file, upload the ZIP file to an OBS bucket, and obtain the OBS link URL.
2. Log in to the FunctionGraph console, and choose **Functions** > **Dependencies** in the navigation pane.
3. Click **Create Dependency**.
4. Set the dependency name and runtime, specify the OBS link URL, and click **OK**.

   NOTE

   For details about how to obtain the OBS link URL, see **Accessing an Object Using Its URL**. (The following figure is for reference only. Please use the actual URL of the uploaded file package.)

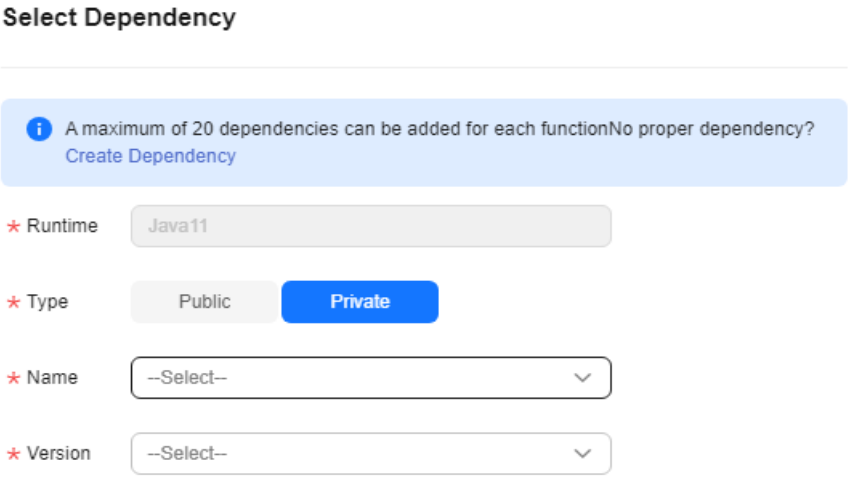
   **Figure 10-3** Obtaining the OBS link URL

**Figure 10-4** Setting the dependency




5. On the function details page, click the **Code** tab, click **Add** in the **Dependencies** area, select the dependency created in **4**, and click **OK**.

**Figure 10-5** Selecting a dependency




**WARNING**

Each dependency package cannot contain a file with the same name as a code file. Otherwise, the two files may be incorrectly merged or overwritten. For example, if dependency package **depends.zip** contains a file named **index.py**, the handler of a function cannot be set to **index.handler**. Otherwise, a code file also named **index.py** will be generated.

# 10.3 Public Dependencies

## 10.3.1 What Is a Public Dependency?

Public dependencies are provided by Huawei Cloud. You can import them to your code on the function details page to implement service logic.

Public dependencies have the following advantages:

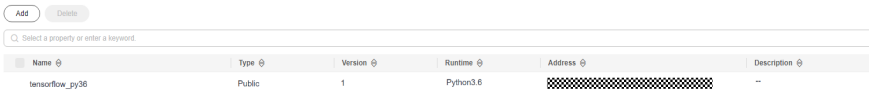
- They provide an out-of-the-box ecosystem, eliminating the need to build and upload dependencies. You can focus on your code and service logic without worrying about the running environment.
- FunctionGraph caches public dependencies for fast access. No network latency will be involved during cold start for accessing dependency files in storage services.
- Public dependencies either smaller or larger than 300 MB can be easily imported and deleted, while private dependencies larger than 300 MB must be split for upload.

# 10.4 Public Dependency Demos

## 10.4.1 Linear Regression with TensorFlow

### Adding TensorFlow on Function Details Page

**Figure 10-6** Adding TensorFlow




### Importing TensorFlow to Code

```
import json
import random
# Import TensorFlow.
import tensorflow as tf
def handler (event, context):

    TRUE_W = random.randint(0,9)
    TRUE_b = random.randint(0,9)

    NUM_SAMPLES = 100

    X = tf.random.normal(shape=[NUM_SAMPLES, 1]).numpy()
    noise = tf.random.normal(shape=[NUM_SAMPLES, 1]).numpy()
    y = X * TRUE_W + TRUE_b + noise
    model = tf.keras.layers.Dense(units=1)

    EPOCHS = 20
    LEARNING_RATE = 0.002
    print("start training")
    for epoch in range(EPOCHS):
```

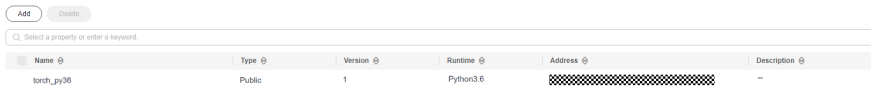
  

```
        with tf.GradientTape() as tape:
            y_ = model(X)
            loss = tf.reduce_sum(tf.keras.losses.mean_squared_error(y, y_))

        grads = tape.gradient(loss, model.variables)
        optimizer = tf.keras.optimizers.SGD(LEARNING_RATE)
        optimizer.apply_gradients(zip(grads, model.variables))

        print('Epoch [{}/{}], loss [{:.3f}]'.format(epoch, EPOCHS, loss))
    print("finished")
    print(TRUE_W,TRUE_b)
    print(model.variables)
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }

class Model(object):
    def __init__(self):
        self.W = tf.Variable(tf.random.uniform([1]))
        self.b = tf.Variable(tf.random.uniform([1]))
    def __call__(self, x):
        return self.W * x + self.b
```

# 10.4.2 Linear Regression with PyTorch

## Adding Torch on Function Details Page

**Figure 10-7** Adding Torch




## Importing Torch to Code

```
# -*- coding:utf-8 -*-
import json
# Import Torch.
import torch as t
import numpy as np
def handler (event, context):
    print("start training!")
    train()
    print("finished!")
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }


def get_fake_data(batch_size=8):
    x = t.rand(batch_size, 1) * 20;
    y = x * 2 + (1 + t.randn(batch_size, 1)) * 3
    return x, y

def train():
    t.manual_seed(1000)
```

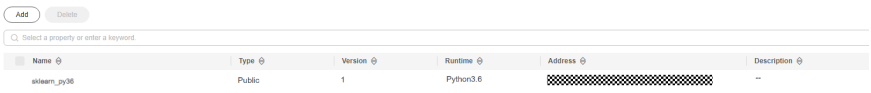
  

```
x, y = get_fake_data()

w = t.rand(1, 1)
b = t.zeros(1, 1)
lr = 0.001


for ii in range(2000):
    x, y = get_fake_data()
    y_pred = x.mm(w) + b.expand_as(y)
    loss = 0.5 * (y_pred - y) ** 2
    loss = loss.sum()

    dloss = 1
    dy_pred = dloss * (y_pred - y)

    dw = x.t().mm(dy_pred)
    db = dy_pred.sum()
    w.sub_(lr * dw)
    b.sub_(lr * db)


    if ii % 10 == 0:
        x = t.arange(0, 20).view(-1, 1)

        y = x.float().mm(w)+ b.expand_as(x)

        x2, y2 = get_fake_data(batch_size=20)

        print("w=",w.item(), "b=",b.item())
```

# 10.4.3 sklearn

## Adding sklearn on Function Details Page

**Figure 10-8** Adding sklearn



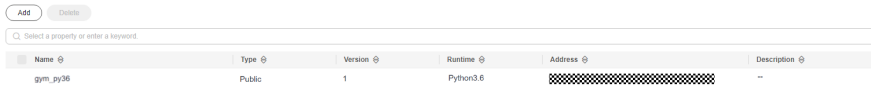

## Importing sklearn to Code

```
# Import sklearn.
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
def handler (event, context):
    iris=datasets.load_iris()
    iris_X=iris.data
    iris_y=iris.target

    X_train,X_test,y_train,y_test=train_test_split(iris_X,iris_y,test_size=0.3)

    knn=KNeighborsClassifier()
    knn.fit(X_train,y_train)

    print(knn.predict(X_test))

    return y_test
```

# 10.4.4 Gym

## Adding Gym on Function Details Page

**Figure 10-9** Adding Gym




## Importing Gym to Code

```
# Import Gym.
import gym
env = gym.make('CartPole-v0')
env.reset()

def handler(event,context):
    for _ in range(10):
        env.render()
        observation, reward, done, info, _ = env.step(env.action_space.sample()) # take a random action
        temp = env.step(env.action_space.sample()) # take a random action
        print('observation:{}, reward:{}, done:{}, info:{}'.format(observation, reward, done, info))
    env.close()
```

# 11 Reserved Instance Management (Old)

## What Is a Reserved Instance?

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 call functions, FunctionGraph automatically allocates execution resources to the requests.
- Reserved instances can be created and released by you as required. After you create reserved instances for a function, FunctionGraph preferentially forwards requests to the reserved instances. If the number of requests exceeds the processing capability of the reserved instances, FunctionGraph will forward the excessive requests to on-demand instances and automatically allocates execution resources to these requests.

  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.

  **NOTE**

  To use reserved instances, **submit a service ticket** to add your account to the whitelist.

You can directly create reserved instances or create them on the function configuration page. Differences between the two modes are described in the following table.

**Table 11-1** Differences between the two modes

| Mode | Advantage | Disadvantage |
| --- | --- | --- |
| Directly creating reserved instances | You can create a reserved instance with just a few clicks. | Only a fixed number of reserved instances can be created. The reserved instances may be insufficient during peak hours and become idle during off-peak hours. |
| Creating reserved instances on the function configuration page | You can create different numbers of reserved instances for different periods, preventing waste of reserved instances during off-peak hours and ensuring enough reserved instances during peak hours. | The procedure is complex. |

## Directly Creating a Fixed Number of Reserved Instances

Ensure that the function, for example, **Objective-func**, for which you want to create reserved instances already exists on the FunctionGraph console.

1. Log in to the FunctionGraph console, and choose **Functions** > **Reserved Instances** in the navigation pane.
2. Click **Configure Reserved Instance**.
3. Set the following parameters:

   **Table 11-2** Reserved instance information

   | Parameter | Description |
   | --- | --- |
   | App | Select the app to which the **Objective-func** function belongs. |
   | Function | Select **Objective-func**. |
   | Version | Select a version of the **Objective-func** function. |
   | RI Quantity | Enter the number of reserved instances to be created.<br>To determine how many reserved instances to create, view the number of used instances in the **Instance Statistics** area or check the usage of the **Objective-func** function. |

4. Click **OK**.

   NOTE

   After creating reserved instances for a function, you can only change the reserved instance quantity.

## Creating Reserved Instances on the Function Configuration Page

The number of function instances varies between different periods. To ensure enough reserved instances during peak hours and prevent waste of reserved instances during off-peak hours, you can create a timer for invoking the corresponding function with different number of reserved instances in different periods.

Ensure that the function, for example, **Objective-func**, for which you want to create reserved instances already exists on the FunctionGraph console.

1. Return to the FunctionGraph console, and choose **Functions** > **Function List** in the navigation pane.
2. Click **Create Function**.
3. Set the following information.

| Parameter | Description |
| --- | --- |
| Template | Select **Create from scratch**. |
| Function Name | Enter a name to identify the function. |
| App | Select **default**. |
| Agency | Select **Use no agency**. |
| Description | Enter a description for the function. This parameter is optional. |
| Runtime | Select **Python 2.7**. |
| Handler | Enter **index.handler**. |
| Code Entry Mode | When creating a function, select **Default code**. On the **Configuration** tab page, select **Edit code inline** and enter the following code: |

```
# -*- coding:utf-8 -*-
import json
import requests
def handler (event, context):
    domainId = "https://{Endpoint}"
    url = "/v2/{project_id}/fgs/functions/{func-urn}/reservedinstances"
    token = context.getToken()
    requrl = domainId + url
    headerdata = {"Content-Type":"application/json","x-auth-token":token}
    r = requests.put(requrl, data=event["user_event"], headers=headerdata,verify=False)
    return r.json
```

Replace the following parameters with the actual values:

- *Endpoint*: Endpoint of the **Objective-func** function. For details, see **Regions and Endpoints**.
- *project_id*: ID of the project to which the **Objective-func** function belongs. For details, see **Obtaining a Project ID**.
- *func-urn*: URN of the **Objective-func** function.

4. Click **Create Function**.
5. On the **Configuration** tab page, click **Create Agency**.
6. Create an agency and grant it the **FunctionGraph User** permission. For details, see **Configuring Agency Permissions**.
7. On the **Configuration** tab page, select the agency created in **6**, and click **Save**.
8. On the **Triggers** tab page, click **Create Trigger**.
9. Set the following information.

| Parameter | Description |
| --- | --- |
| Trigger Type | Select **Timer**. |
| Timer Name | Enter a name to identify the timer. |
| Rule | Select **Cron expression** and enter a cron expression as required. |
| Enable Trigger | By default, this function is enabled. Retain the default setting. |
| Additional Information | Enter the number of reserved instances required in different periods based on the trigger rule. |

**Figure 11-1** Creating a timer trigger

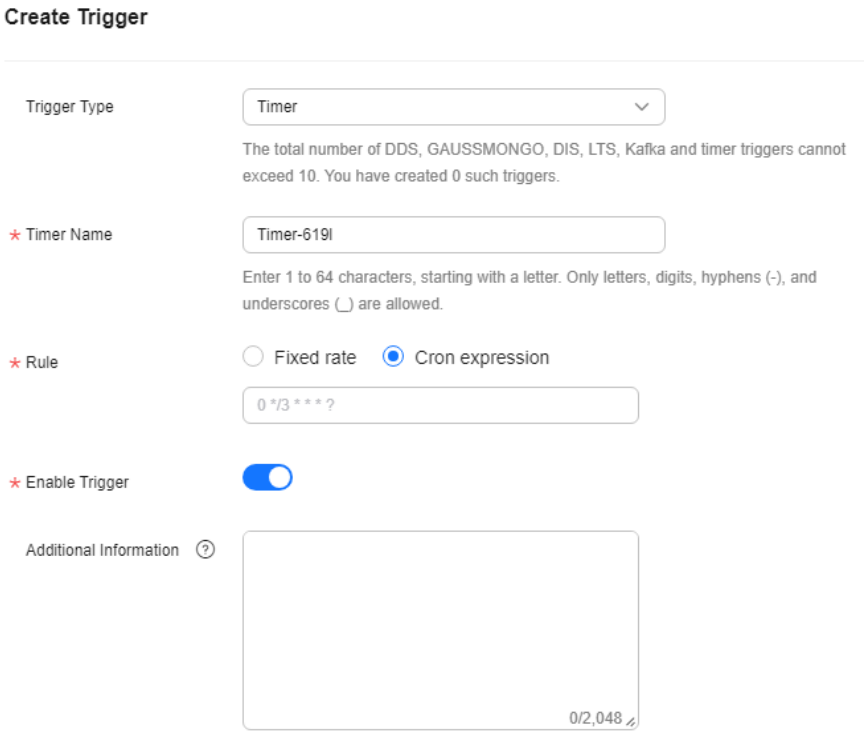



**Figure 11-1** shows a timer trigger that requests FunctionGraph to create two reserved instances every 3 minutes.

10. Click **OK**.

    After trigger creation, different numbers of reserved instances will be created for the **Objective-func** function in different periods based on the trigger rule.

# 12 Reserved Instance Management

## Introduction

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 call functions, FunctionGraph automatically allocates execution resources to the requests.
- Reserved instances can be created and released by you as required. After you create reserved instances for a function, FunctionGraph preferentially forwards requests to the reserved instances. If the number of requests exceeds the processing capability of the reserved instances, FunctionGraph will forward the excessive requests to on-demand instances and automatically allocates execution resources to these requests.

  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.)

  **You can configure a fixed number of reserved instances or scheduled, metric, and intelligent recommendation policies.**

  NOTE

  By default, you do not have permission to use the metric and intelligent recommendation policies. To use them, **submit a service ticket**.

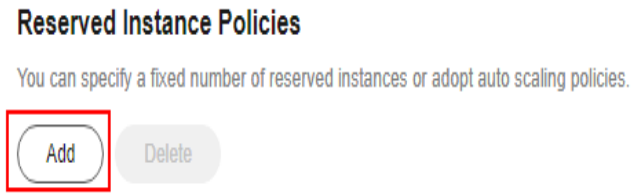
## Configuring a Fixed Number of Reserved Instances

Ensure that the function for which you want to create reserved instances already exists on the FunctionGraph console.

1. Log in to the **FunctionGraph console**. In the navigation pane, choose **Functions** > **Function List**.
2. Click the target function to go to the details page.
3. Choose **Configuration** > **Concurrency**, and click **Add**.

**Figure 12-1** Clicking Add




4. Set parameters by referring to **Table 12-1**.

   You can create a specified number of reserved instances for a function version or alias. This number cannot exceed the maximum number of requests per instance or the maximum number of instances per function.

**Figure 12-2** Basic settings

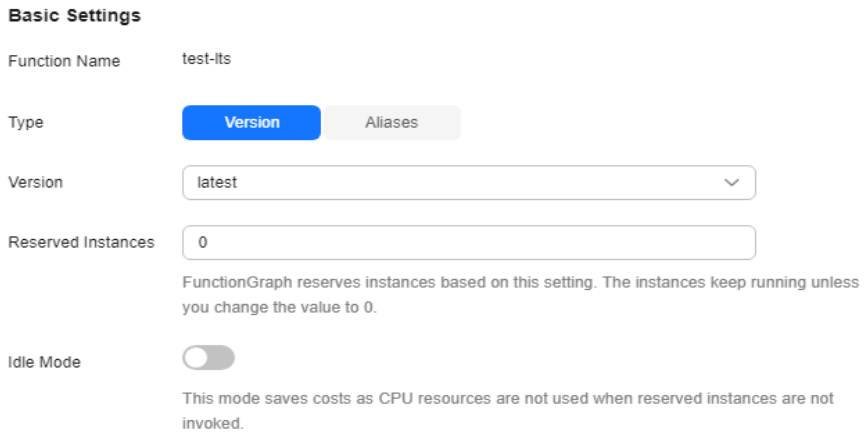



**Table 12-1** Basic settings

| Parameter | Description |
| --- | --- |
| Function Name | Name of the current function. |
| Type | Select **Version** or **Aliases**. |
| Version | Set this parameter when you select **Version** for **Type**. |
| Alias | Set this parameter when you select **Aliases** for **Type**. |
| Reserved Instances | Minimum number of instances. Max.: **1000**. FunctionGraph reserves the specified number of instances for the function. These instances will always run unless you change **Min. Instances** to **0**. |
| Idle Mode | This mode saves costs as CPU resources are not used when reserved instances are not invoked. |

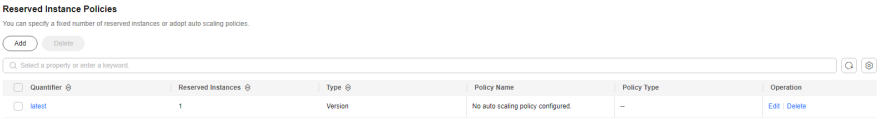
  

NOTE

- 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.

5. Click **OK**. The new policy is displayed in the reserved instance policy list.

**Figure 12-3** Policy list

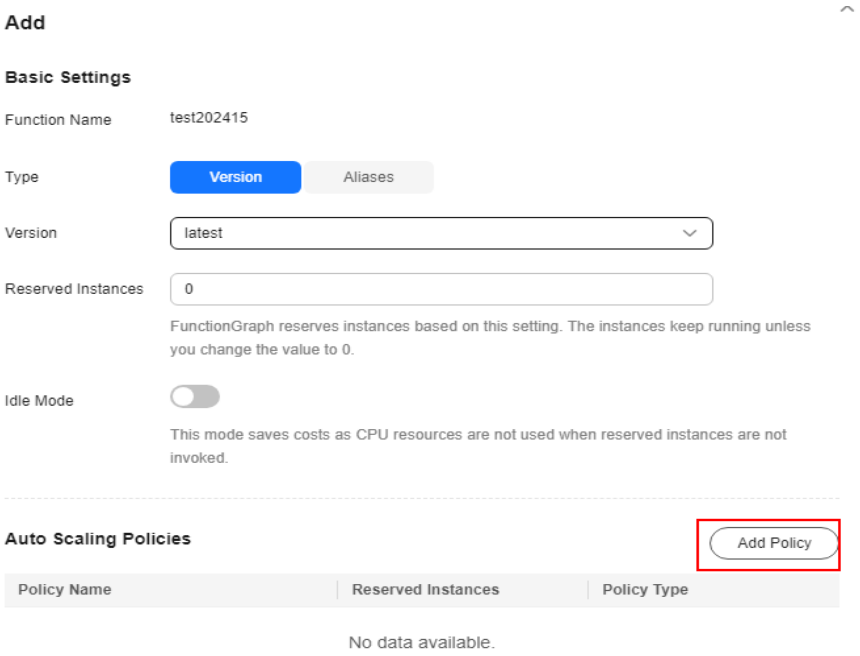



## Configuring a Scheduled Scaling Policy

Configure the number of reserved instances that will run in a specified period and a cron expression. During this period, FunctionGraph adjusts the number of reserved instances based on the cron expression. When the period expires, the fixed number of instances will be reserved.

1. Configure the basic settings by referring to **Table 12-1**, and then click **Add Policy**.

**Figure 12-4** Clicking Add Policy




2. Set parameters by referring to **Table 12-2**.

**Figure 12-5** Adding a policy

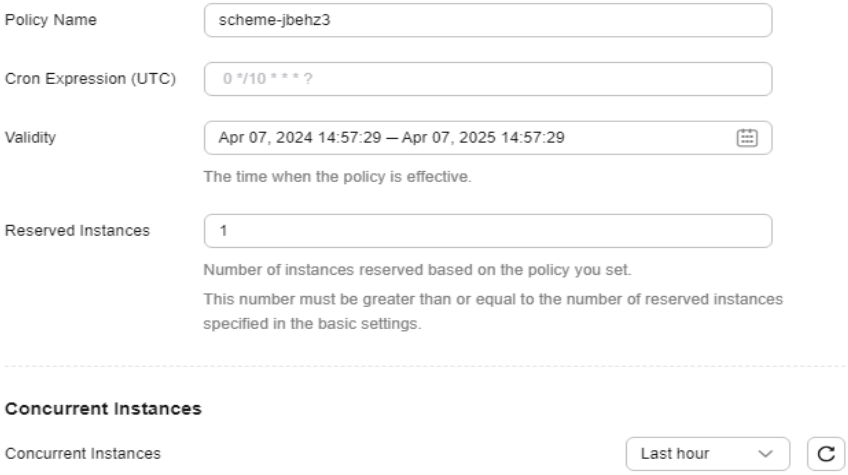



**Table 12-2** Scheduled scaling policy parameters

| Parameter | Description |
| --- | --- |
| Policy Name | Policy name. |
| Cron Expression (UTC) | Set this parameter by referring to **Cron Expressions for a Function Timer Trigger**. |
| Validity | Local time when the cron expression is effective.<br>The scheduled scaling policy is effective only during this validity period. In other time, the **Min. Instances** in the basic settings is used. |
| Reserved Instances | The number of reserved instances to be created when the policy is effective.<br>Set a number that meets your service requirements.<br>**NOTE**<br>The number must be greater than or equal to the **Min. Instances** in the basic settings. |

3. Click **OK**. The new policy is displayed in the reserved instance policy list.

**Figure 12-6** Policy list


4. To modify the reserved instance policy, click **Edit** in the **Operation** column. Then modify or add scheduled scaling policies.
5. To delete a reserved instance policy under a function version or alias, click **Delete** in the **Operation** column.

6. To view concurrent instances, click a quantifier in the reserved instance policy list, and click a scheduled scaling policy name.

   NOTE

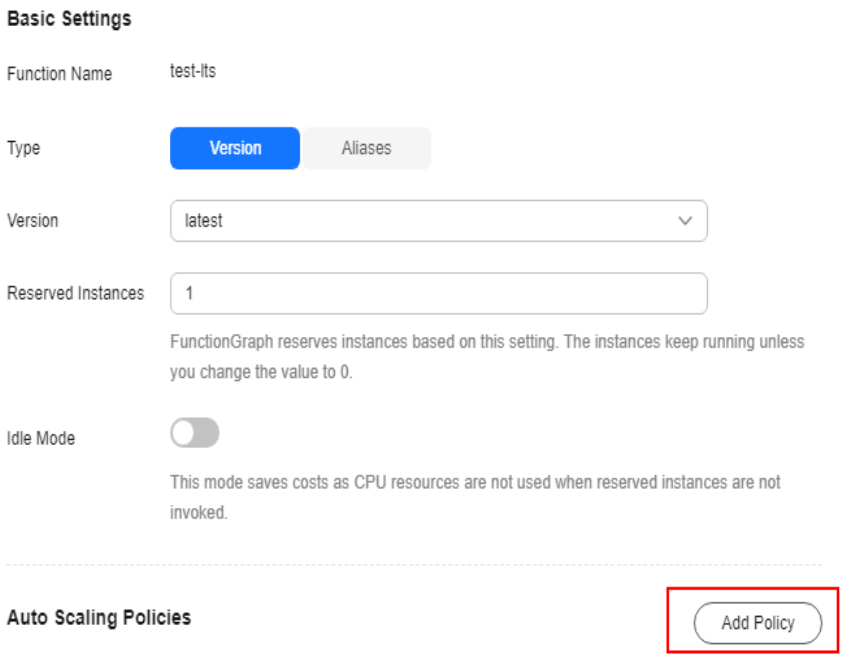
   Multiple scheduled policies can be configured. For example, the number of reserved instances at 08:00 and 21:00 is updated to 100 and 10 respectively.

## Configuring a Metric Scaling Policy

You can dynamically adjust the number of reserved instances based on service metrics (currently, only the number of concurrent users is supported). When configuring metric policies, you need to configure an agency that has permission to query AOM metrics and query and configure functions.

**Step 1** Configure the basic settings by referring to **Table 12-1**, and then click **Add Policy**.

**Figure 12-7** Adding a policy




**Step 2** Set parameters by referring to **Table 12-3**.

**Figure 12-8** Configuring a metric policy

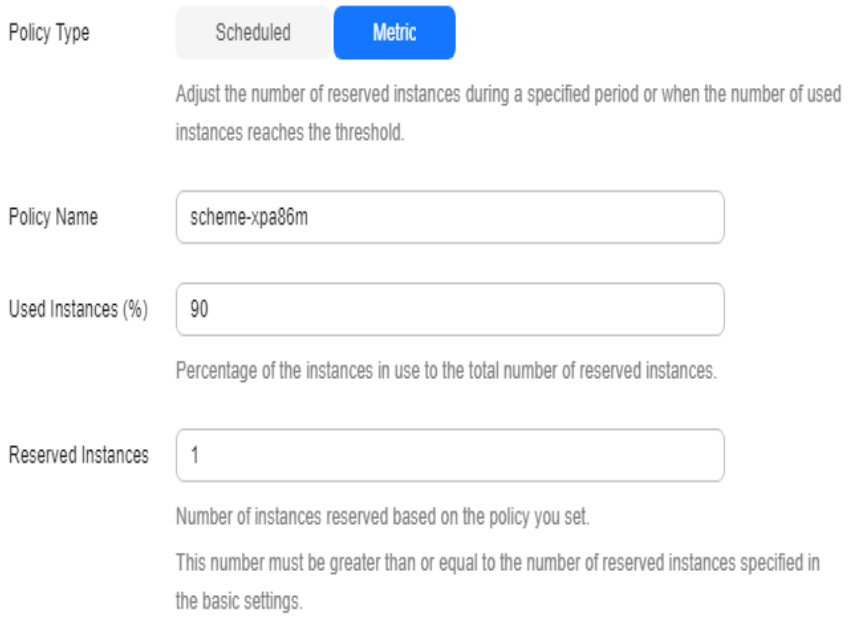



**Table 12-3** Metric policy parameters

| Parameter | Description |
| --- | --- |
| Policy Name | Policy name. |
| Used Instances (%) | The value ranges from 1 to 99, that is, the percentage of actually used reserved instances to the total number of reserved instances of the function. If the actual value is higher than the threshold, FunctionGraph gradually reduces the number of reserved instances. If the actual value is lower than the threshold, FunctionGraph directly creates the corresponding number of reserved instances every minute. (The metric is also generated every minute. Therefore, the adjustment will be delayed for 1 to 2 minutes.) |
| Reserved Instances | The number of reserved instances to be created when the threshold is not reached.<br>It specifies the latest number of reserved instances based on the metric policy. For example, if only one reserved instance needs to be created based on the policy and the minimum number of reserved instances is set to **5**, five reserved instances will be created.<br>**NOTE**<br>The number must be greater than or equal to the **Min. Instances** in the basic settings. |

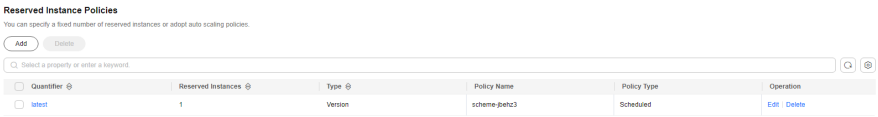
  

**Table 12-4** Configuring agency permissions

| Service | Common Permission | Fine-Grained Permission |
| --- | --- | --- |
| FunctionGraph | FunctionGraph ReadOnlyAccess | functiongraph:function:getConfig |

**Step 3** Click **OK**. The new policy is displayed in the reserved instance policy list.

**Figure 12-9** Policy list




**Step 4** To modify the reserved instance policy, click **Edit** in the **Operation** column. Then modify or add scheduled scaling policies.

**Step 5** To delete a reserved instance policy under a function version or alias, click **Delete** in the **Operation** column.

**Step 6** To view concurrent instances, click a quantifier in the reserved instance policy list, and click a scheduled scaling policy name.

NOTE

Scheduled policies and metric policies can be added together. When both policies are added, the minimum number of reserved instances for one of the policy types cannot be less than the latest number of reserved instances for the other policy type. For example, at 08:59, 9 reserved instances are updated by the metric policy, and 5 instances are configured for a scheduled policy at 10:00. Therefore, 9 reserved instances are updated at 10:00. If the next scheduled policy takes effect at 11:00, at least 5 reserved instances will be updated during10:00 and 11:00.

**----End**

## Configuring an Intelligent Recommendation Policy

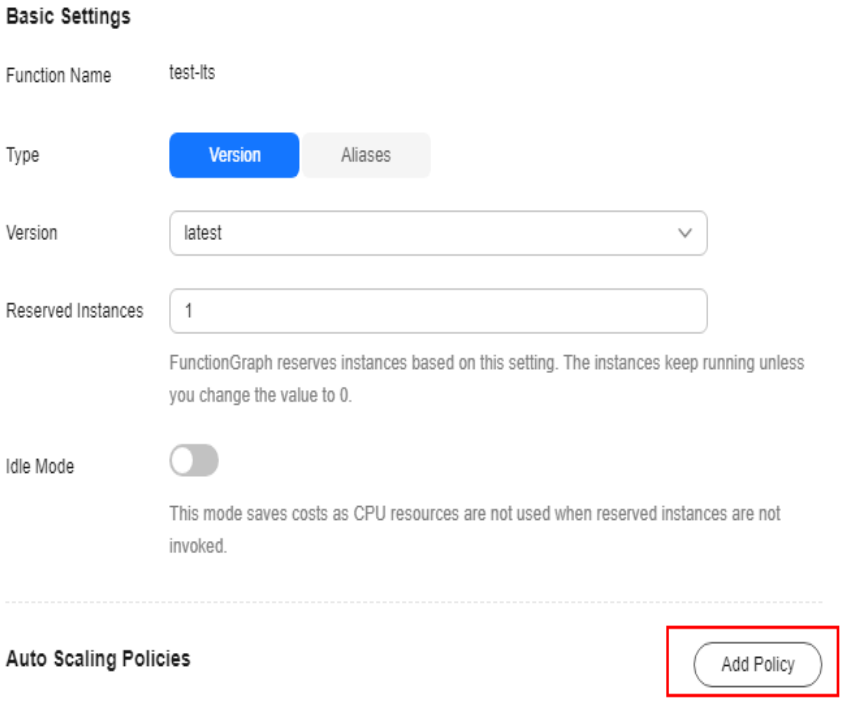
Intelligent recommendation policies are based on feature profiling and load prediction technologies, dynamically adjusting reserved instances for peak and off-peak demands.

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. Click **Add Policy**, as shown in the following figure.

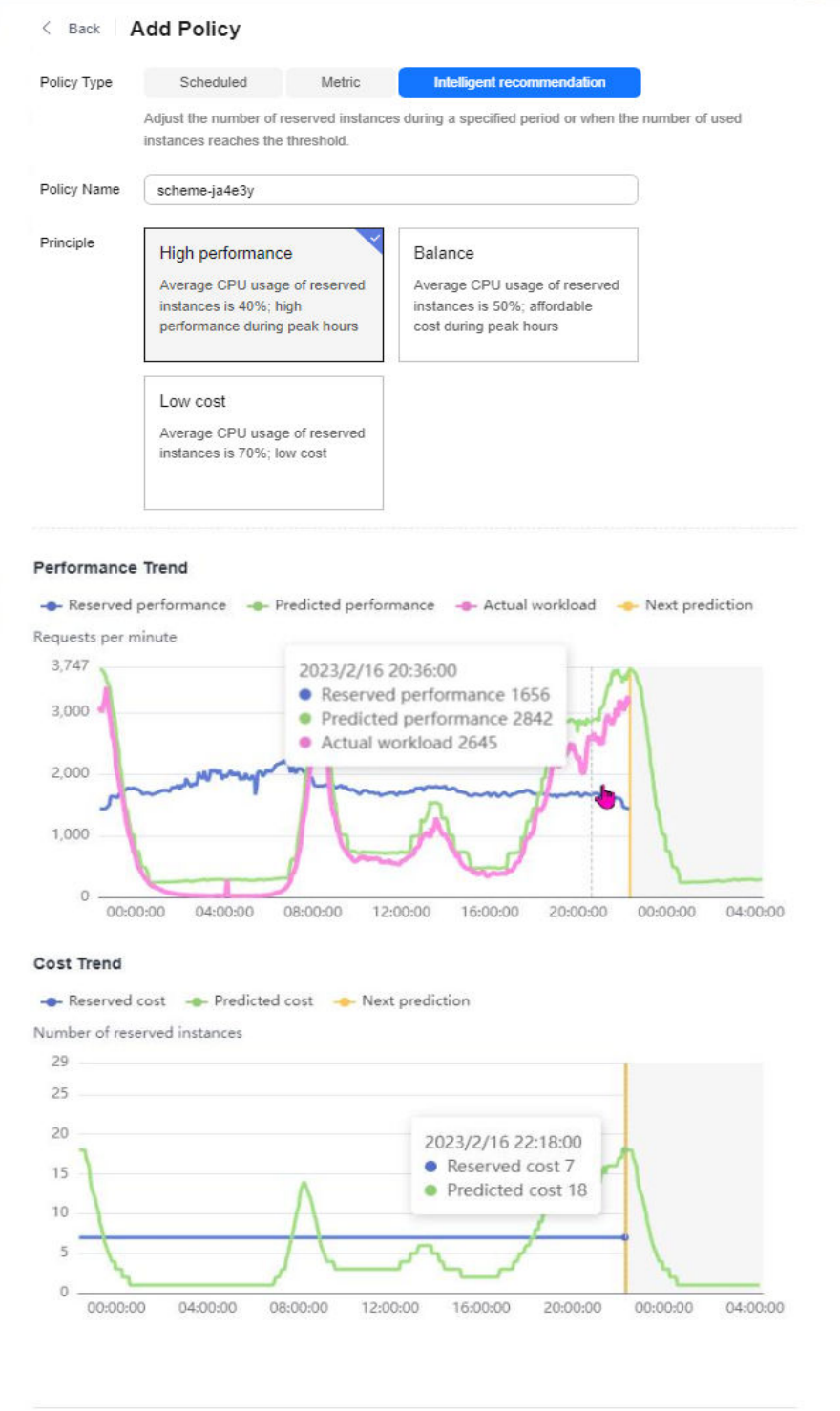
**Figure 12-10** Clicking Add Policy




2. Select **Intelligent recommendation**, and select any of **High performance**, **Balance**, and **Low cost** while referring to the performance and cost trends.

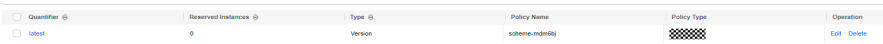
  

**Figure 12-11** Intelligent recommendation




3. Click **OK**. The new policy is displayed in the reserved instance policy list.

**Figure 12-12** Reserved Instance Policies



4. To view and modify a reserved instance policy, click **Edit** in the **Operation** column.

5. To delete a reserved instance policy under a function version or alias, click **Delete** in the **Operation** column.
6. Reserved instances are executed with the intelligent recommendation policy. To view the reserved instance costs and performance, click a quantifier in the policy list, and click a policy name.

# 13 Flow Management

## 13.1 Flow Overview

NOTE

Flows are available only in CN East-Shanghai1 and AP-Singapore.

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 flow components, component orchestration rules, expression operators, and configuration examples.

### Components

There are multiple types of components. Drag, drop, configure, and connect components to visually orchestrate flows. Before using flows, understand the components described in **Table 13-1**.

**Table 13-1** Components

| Category | Name | Description |
| --- | --- | --- |
| Components | Function | FunctionGraph functions. For details, see **Creating a Function**. |
| | EG | EventGrid (EG) publishes events to a specified channel. For details about how to create resources, see the EG documentation. |
| Processors | Callback | Similar to manual review, this node blocks a function flow by conditions until the callback API is called to resume the flow. |

| Category | Name | Description |
| --- | --- | --- |
| | Subflow | A flow can be incorporated into another one as its subflow. |
| | Parallel branch | A flow can have multiple parallel branches, which will be executed at the same time. You can determine the next step of each branch. |
| | Start | Start of a flow. A start node only has triggers, and each flow has one start node. |
| | Error handling | Determines the next step in case of an execution failure. |
| | Loop | Cyclically processes each element in an array. All subflows in a loop are executed each time. |
| | Wait | Determines when the next step will be executed after the previous step is finished. |
| | Service | Abstracts complex operations involving multiple functions and combines these functions into an atomic node for easy management. |
| | Conditional branch | Determines whether to execute the next branch based on specified conditions. |
| | Stop | Indicates the end of a flow. |

## Orchestration Rules

- 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:
  a. Do not connect any conditional, parallel, or start node to the last node of the flow.
  b. Add a stop node as the last node of the flow.
- Component design rules

**Table 13-2** Trigger, function, and EG

| Type | Description | Mandatory |
| --- | --- | --- |
| Trigger | • A flow can have a maximum of 10 triggers.<br>• Triggers must be included in the start node.<br>• Triggers cannot be connected to any other node. | No |

| Type | Description | Mandatory |
| --- | --- | --- |
| Function | • A flow can have a maximum of 99 functions.<br>• A function connected to an error handling node can be connected to another node that is not start or error handling.<br>• A function that is not connected to an error handling node can be connected to only one non-start node. | No |
| EG | • A flow can have a maximum of 10 EG nodes.<br>• An EG node connected to an error handling node can only be connected to another node that is not start or error handling.<br>• An EG node that is not connected to an error handling node can be connected to only one non-start node. | No |

**Table 13-3** Processors

| Type | Description | Mandatory |
| --- | --- | --- |
| Callback | For details about the constraints of callback nodes, see the function parameter in **Table 13-2**. Callback nodes cannot be subnodes of a service node. | No |
| Subflow | A flow. | No |
| Parallel branch | • Indicates that the connected branches will be executed concurrently.<br>• A parallel branch connects 1 to 20 nodes that are not error handling, start, or stop. | No |
| Start | • Indicates the start of a flow. Each flow has only one start node.<br>• The start node must be followed by a node that is not stop or error handling. | Yes |

| Type | Description | Mandatory |
| --- | --- | --- |
| Error handling | Can be connected to a maximum of 10 nodes that are not start, stop, or error handling. | No |
| Loop | Cyclically processes each element in an array. All subflows in a loop are executed each time.<br>The subflows in a loop must satisfy the following rules:<br>1. Start with a function or wait node.<br>2. Only include function, wait, and error handling nodes. | No |
| Wait | Can be connected to zero or one node that is not start or error handling. | No |
| Service | Consists of multiple functions and can be connected to a stop node or an error handling node. | No |
| Conditional branch | Connects 2 to 20 nodes that are not start, stop, or error handling. | No |
| Stop | Not followed by any other node. | No |

## Expression Operators

The expression for an error handling or conditional branch node is in the format "[JsonPath] + [logical operator] + [value]". For example, **$.age >= 20**.

**JsonPath Description**

| Operator | Supported | Description |
| --- | --- | --- |
| $ | Y | The root element to query. This starts all regular expressions. |
| @ | Y | The current node being processed. |
| . | Y | Subnode. |
| [ (, )] | Y | Array indexes. |
| [start:end] | Y | Array slice operator. |
| [?()] | Y | Filter expression, which must be evaluated to a Boolean value. |

**Examples**

- Simple values: JSON data example

```
{
    "fruits": [ "apple", "orange", "pear" ],
    "vegetables": [{
    "veggieName": "potato",
    "veggieLike": true
     },
     {
         "veggieName": "broccoli",
         "veggieLike": false
     }]
}
```

The **$.fruits** expression obtains all values under **fruits**.

The result of **$.fruits** is **["apple","orange","pear"]**.

- Simple filtering: JSON data example

```
{
     "fruits": [ "apple", "orange", "pear" ],
     "vegetables": [{
     "veggieName": "potato",
     "veggieLike": true
     },
     {
         "veggieName": "broccoli",
         "veggieLike": false
     }]
}
```

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]
```

**Logical 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
   }
}
```

These are the supported operators.

| Operator | Description | Example | Return Value | Remarks |
| --- | --- | --- | --- | --- |
| == | Equal to | $.name == 'apple' | true | Supported data types: int, float, string, bool, and nil |

| Operator | Description | Example | Return Value | Remarks |
| --- | --- | --- | --- | --- |
| != | 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. |

NOTE

- A string constant must be enclosed with quotation marks (''). For example, **'apple'**.
- JsonPath expressions cannot contain =, !=, <, >, |, or &.

## Configuration Examples

**Example 1**: Parallel branch

1. Create three Python 3.9 functions with the following code:
   - Function 1: Returning the value of the event input

     ```
     import json
     def handler (event, context):
         input = event.get('input',0)
         return {
             "result": input
             }
     ```

   - Function 2: Returning the value of the event input plus 2

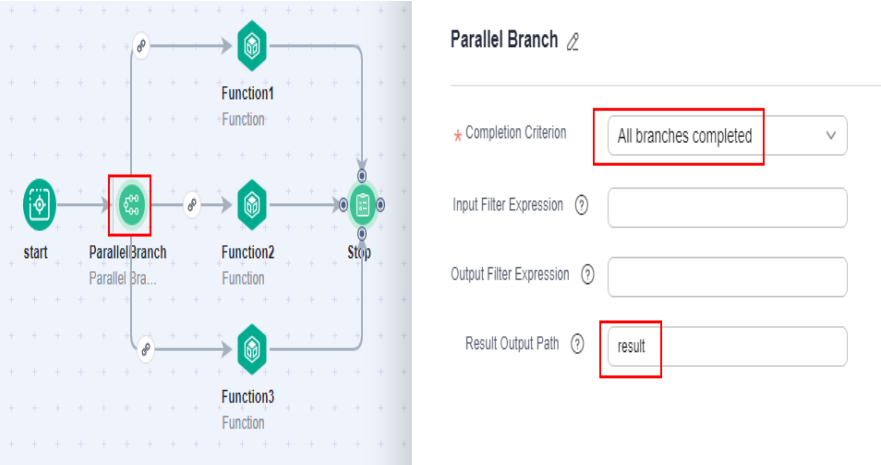
```
import json
def handler (event, context):
    input = event.get('input',0)
    return {
        "result": input+2
        }
```

- Function 3: Returning the square value of the event input

```
import json
def handler (event, context):
    input = event.get('input',0)
    return {
        "result": input*input
        }
```

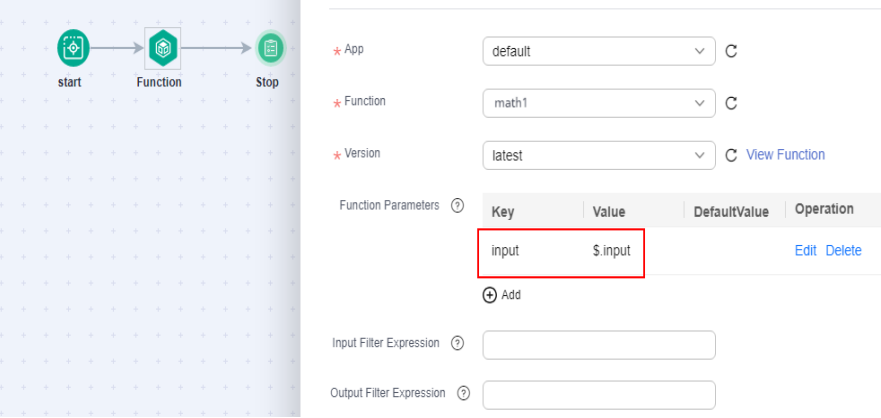
2. Add components to a flow by drag and drop and configure a parallel branch node according to the following figure.

**Figure 13-1** Configuring a parallel branch node




3. Configure three function nodes by referring to **Table 13-2**.

**Figure 13-2** Configuring a function node



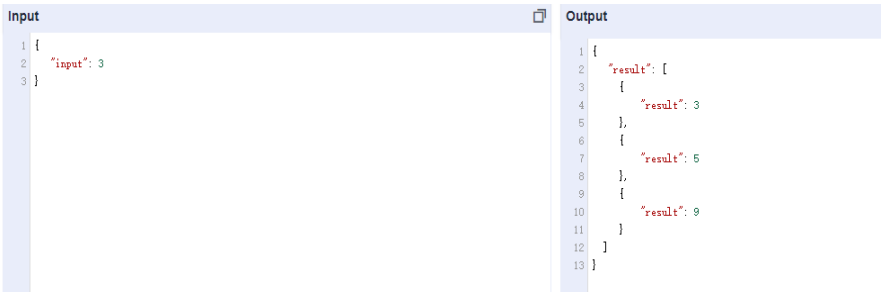

4. Save the flow. Then start execution with the following input:

```
{
  "input":3
}
```

5. Click the flow to view the execution result.

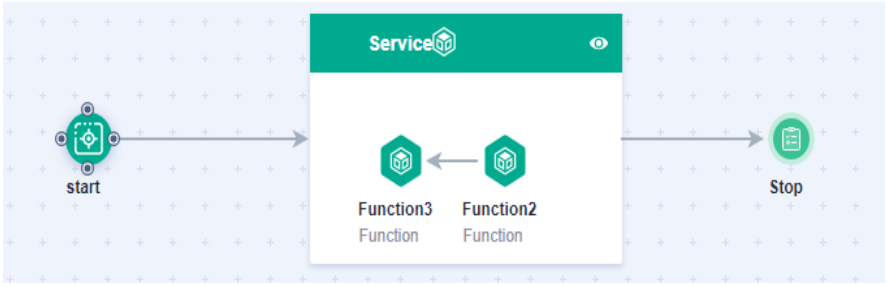
**Figure 13-3** Execution result




**Example 2**: Service node in serial mode

1. Orchestrate components by drag and drop, and set the execution mode of the service node to **Serial**.

**Figure 13-4** Configuring a service node




2. Add **function 2** and **function 3** to the service node and configure them according to the following figures.

**Figure 13-5** Configuring function node 2

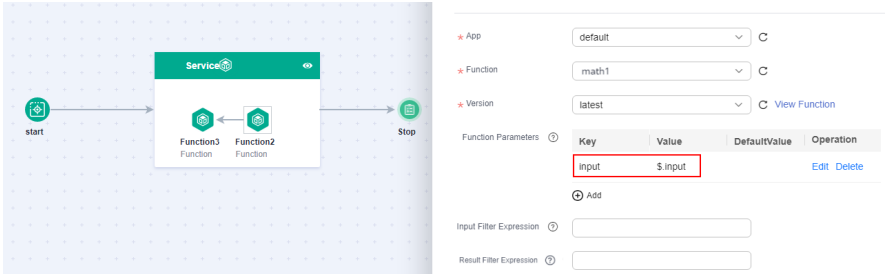
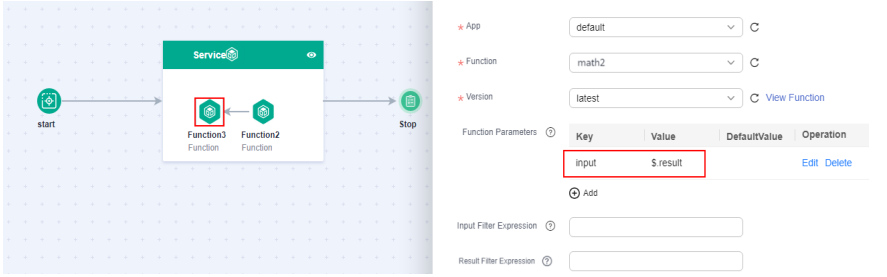



**Figure 13-6** Configuring function node 3



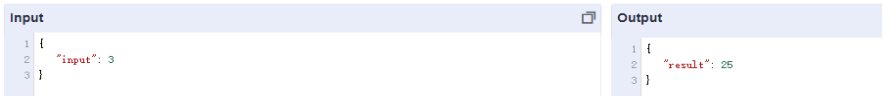

3. Save the flow. Then start execution with the following input:

```
{
  "input":3
}
```

4. Click the flow to view the execution result.

**Figure 13-7** Execution result



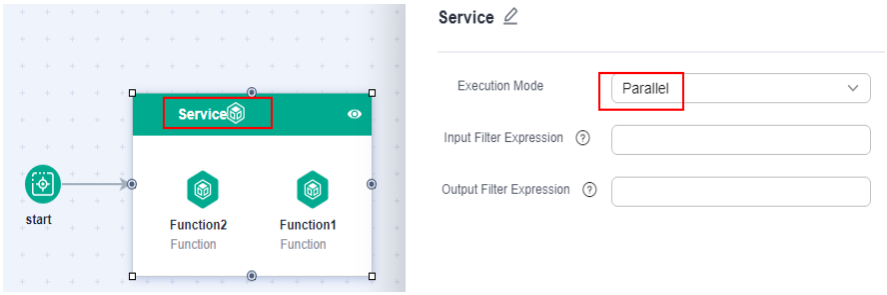

**Example 3**: Service node in parallel mode

1. Create two Python 3.9 functions with the following code:

```
import json
def handler (event, context):
    print(event)
    return {"result":"success"}
```

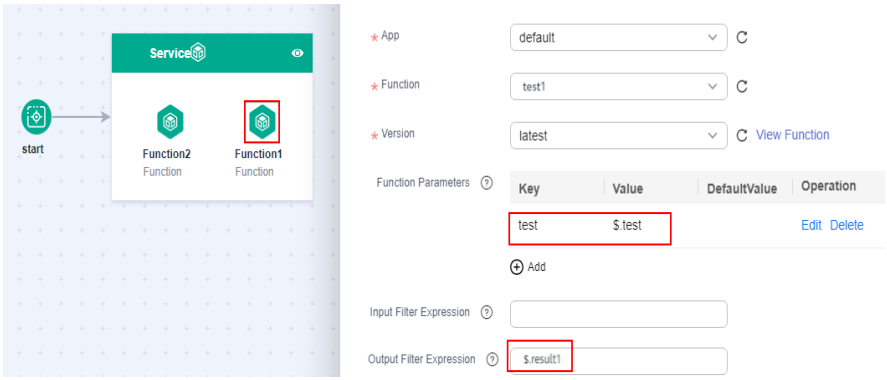
2. Add components to a flow by drag and drop and configure a service node according to the following figure.

**Figure 13-8** Configuring a service node




3. Configure function node 1.

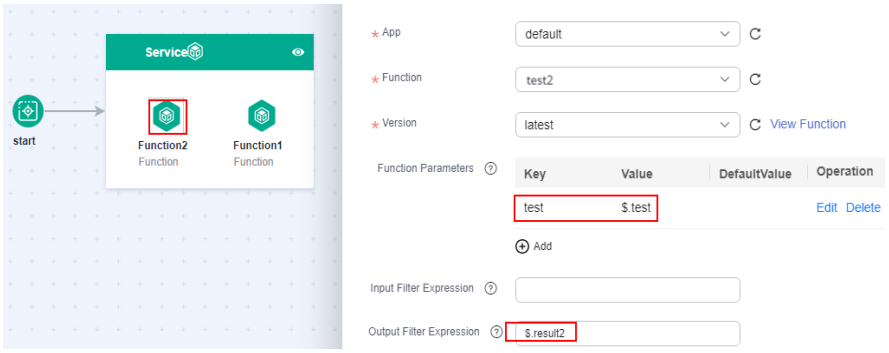
**Figure 13-9** Configuring function node 1




4. Configure function node 2.

**Figure 13-10** Configuring function node 2




5. Save the flow. Then start execution with the following input:

```
{
  "test":123
}
```

6. Click the flow to view the execution result. Results of the two functions have been combined.

**Figure 13-11** Execution result


**NOTE**

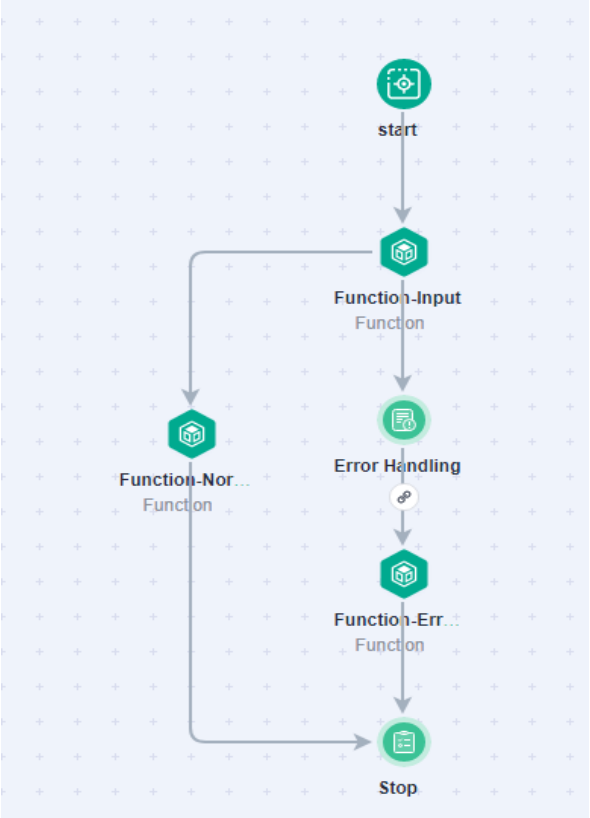
If the result structures of the two functions are the same, only one result will be returned.

**Example 4**: Error handling

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).

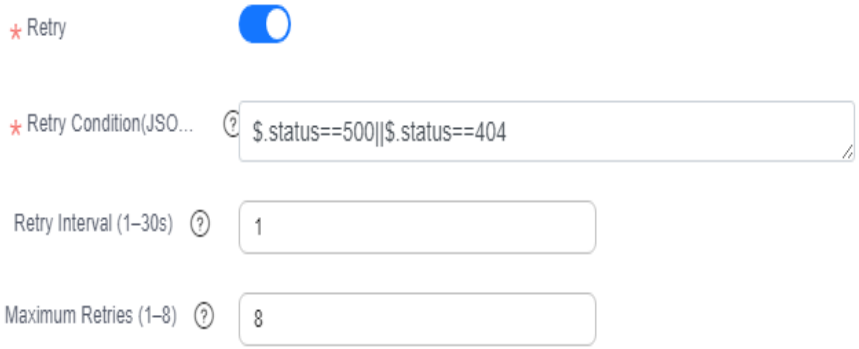
1. Add components to a flow by drag and drop and configure the following nodes:
   - **Function-Input**: Obtains the value of **input** from **event** and outputs it for **status**.
   - **ErrorHandling**: Retries the function when its status is **500** or **404**. This node retries for a maximum of eight times at an interval of 1s.
   - **Function-ErrorRecord**: Records an error if the status of the previous function is still **500** or **404** after eight retries.
   - **Function-NormalOutput**: Executed if the status of **Function-Input** is not **500** or **404**.



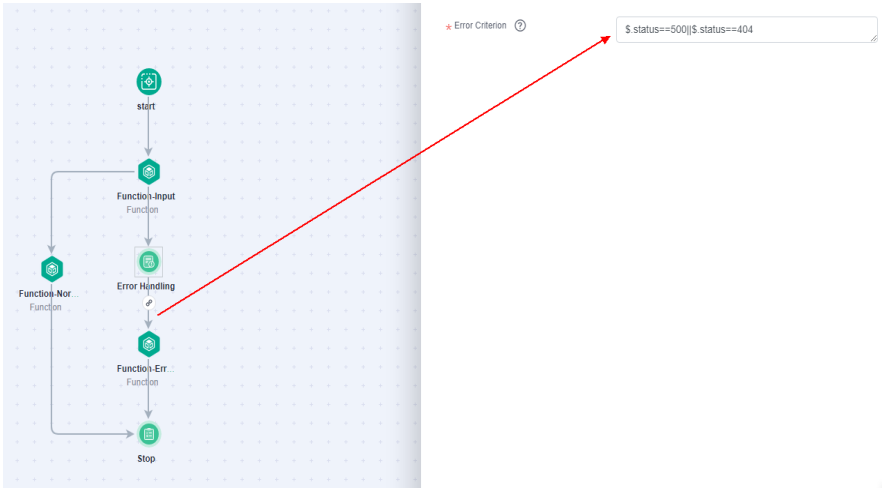

2. Configure the **Error Handling** node with retry condition **$.status==500|| $.status==404**.

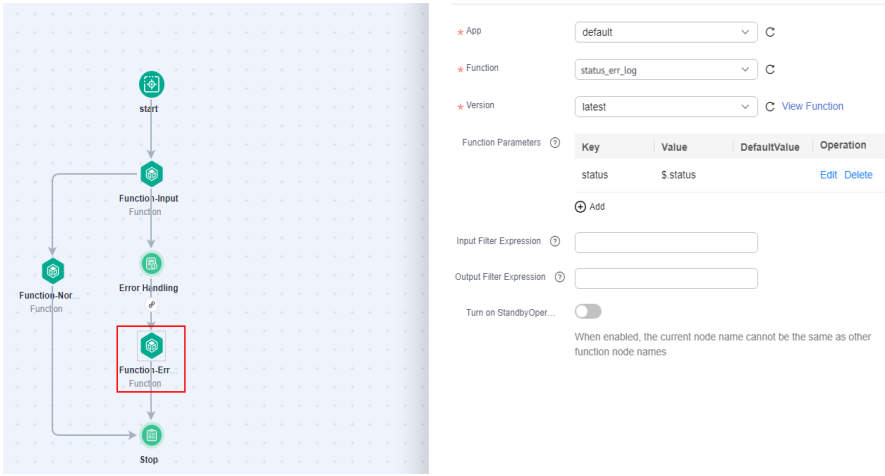
   **Figure 13-12** Configuring exception handling

3. Configure the **Function-ErrorRecord** node to be executed when all retries fail.



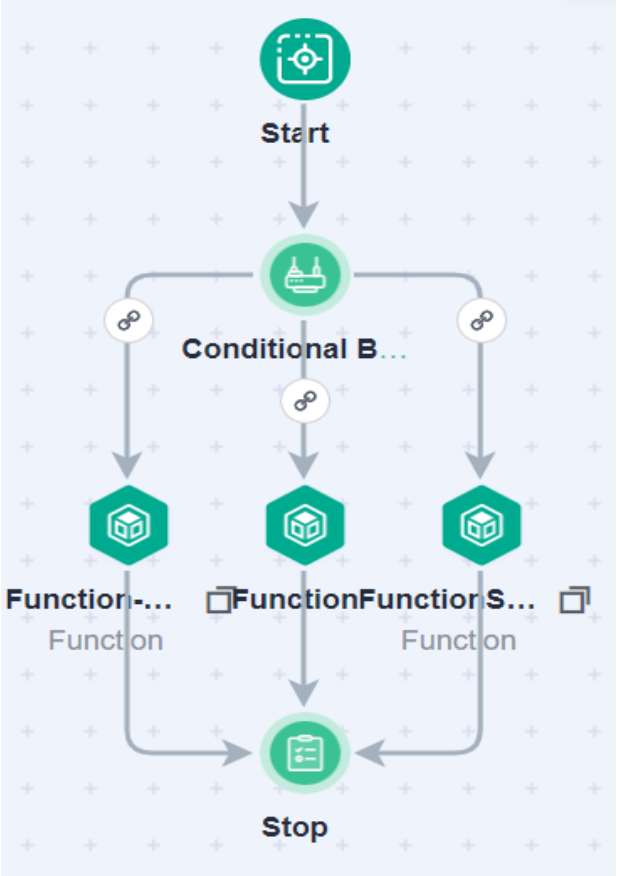

4. Enter **200**. The execution is successful.

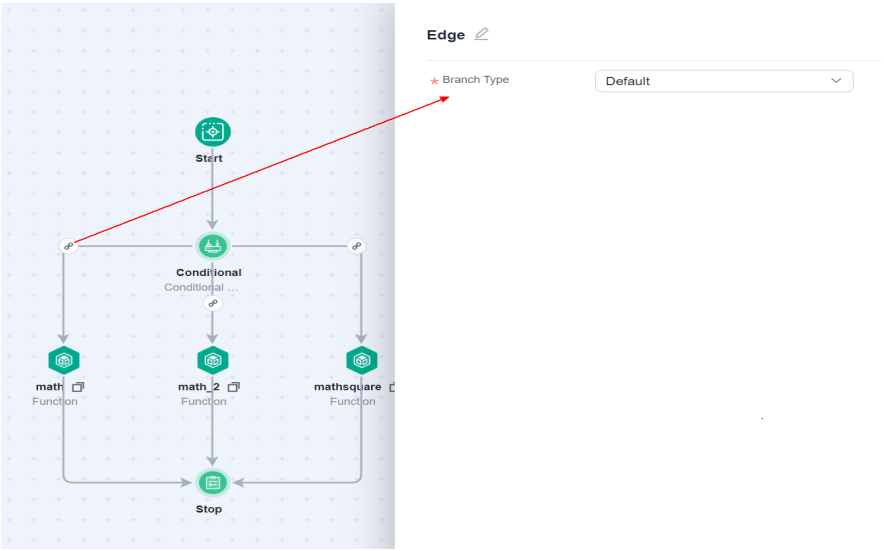

5. Enter **500**. The input function retries and an error is recorded.


**Example 5**: Conditional branch

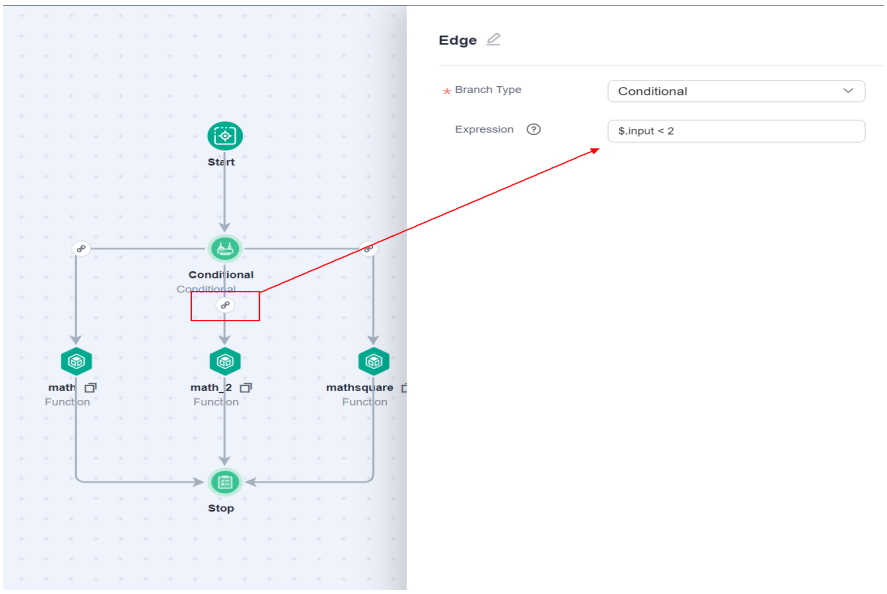
- Use the conditional branch to implement the following logic:
  Input value < 2: **result** is the input value plus 2.
  Input value > 2: **result** is 2 times the input value.
  Input value = 2: **result** is 2.
  The flow design is as follows:

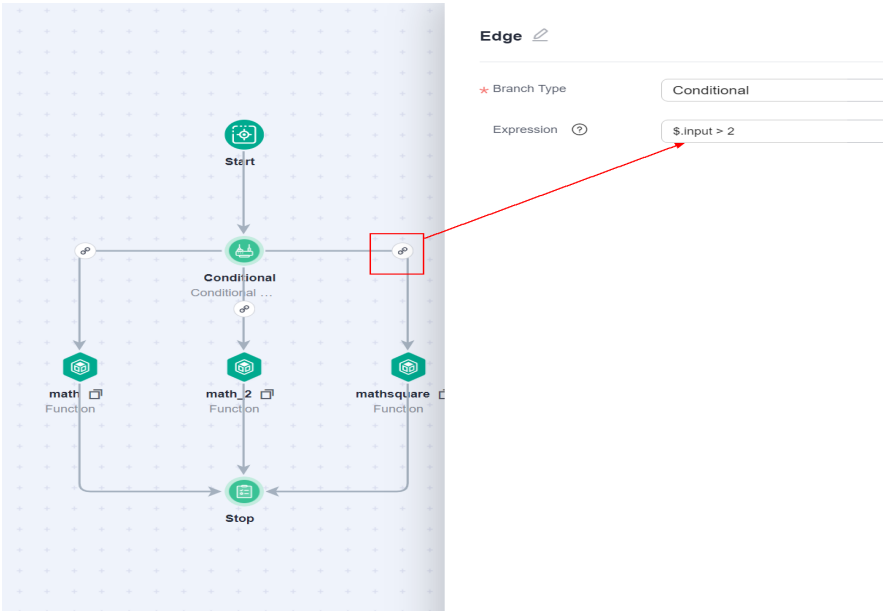
  



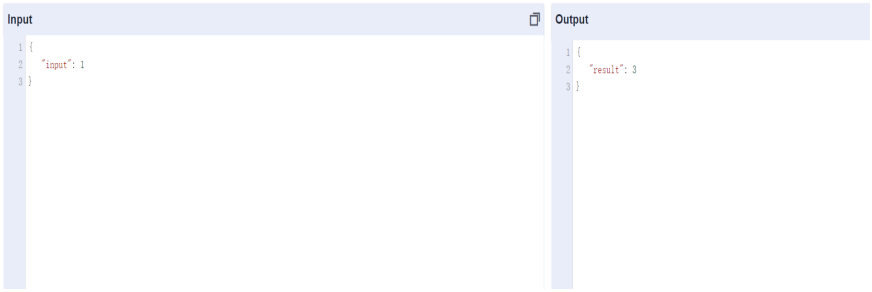



Input **3** to output the square of 3.

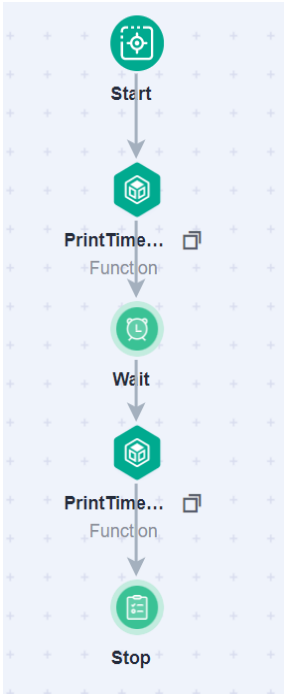

- Input **2** to output 2.


Input **1** to output 1 plus 2.



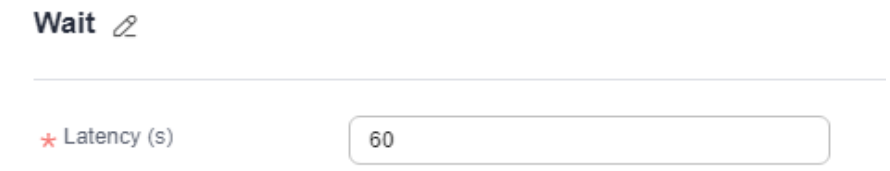

**Example 6**: Wait node

- Use the wait node to delay function invocation. The flow design is shown in the following figure.

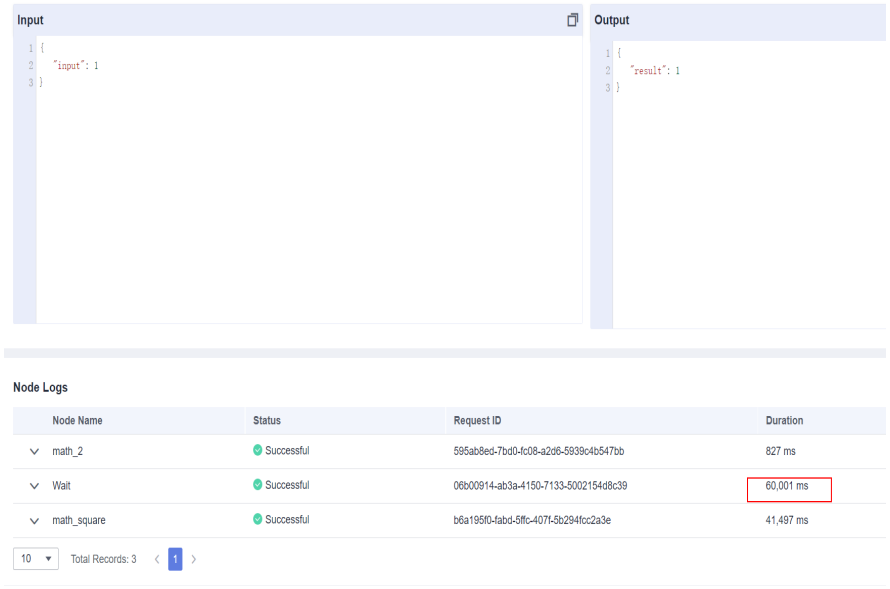



- Set **Latency (s)** to **60**.




– View the execution result.

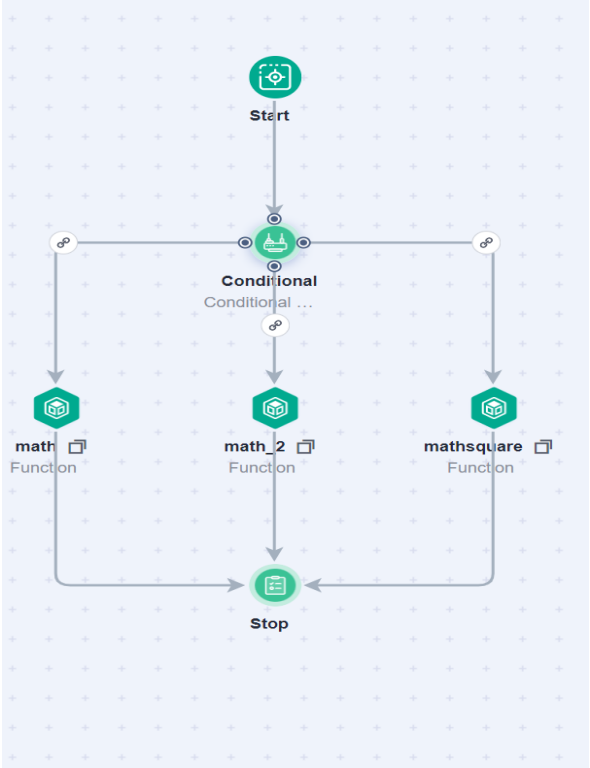


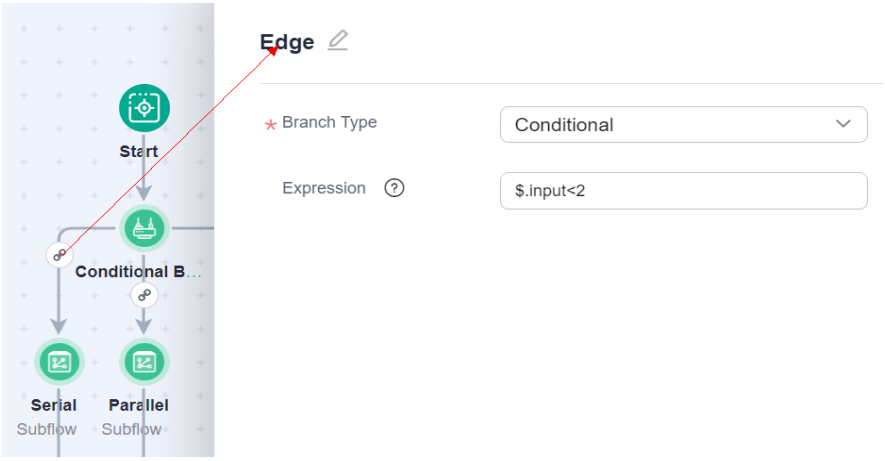

**Example 7**: Subflow

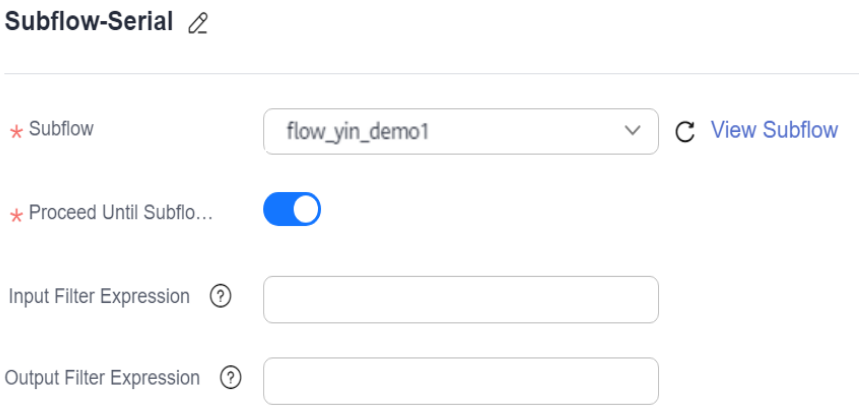
Multiple function flows can be combined as a new flow and reused to avoid repeated development.

- Design a function flow. **input** < 2: Execute the subflows in serial mode to output the input value plus 2 and then squared; **input** = 2: Execute the default branch to output the input value; **input** > 2: Execute the subflows in parallel mode to output the input value, the input value plus 2, and the square of the input value.

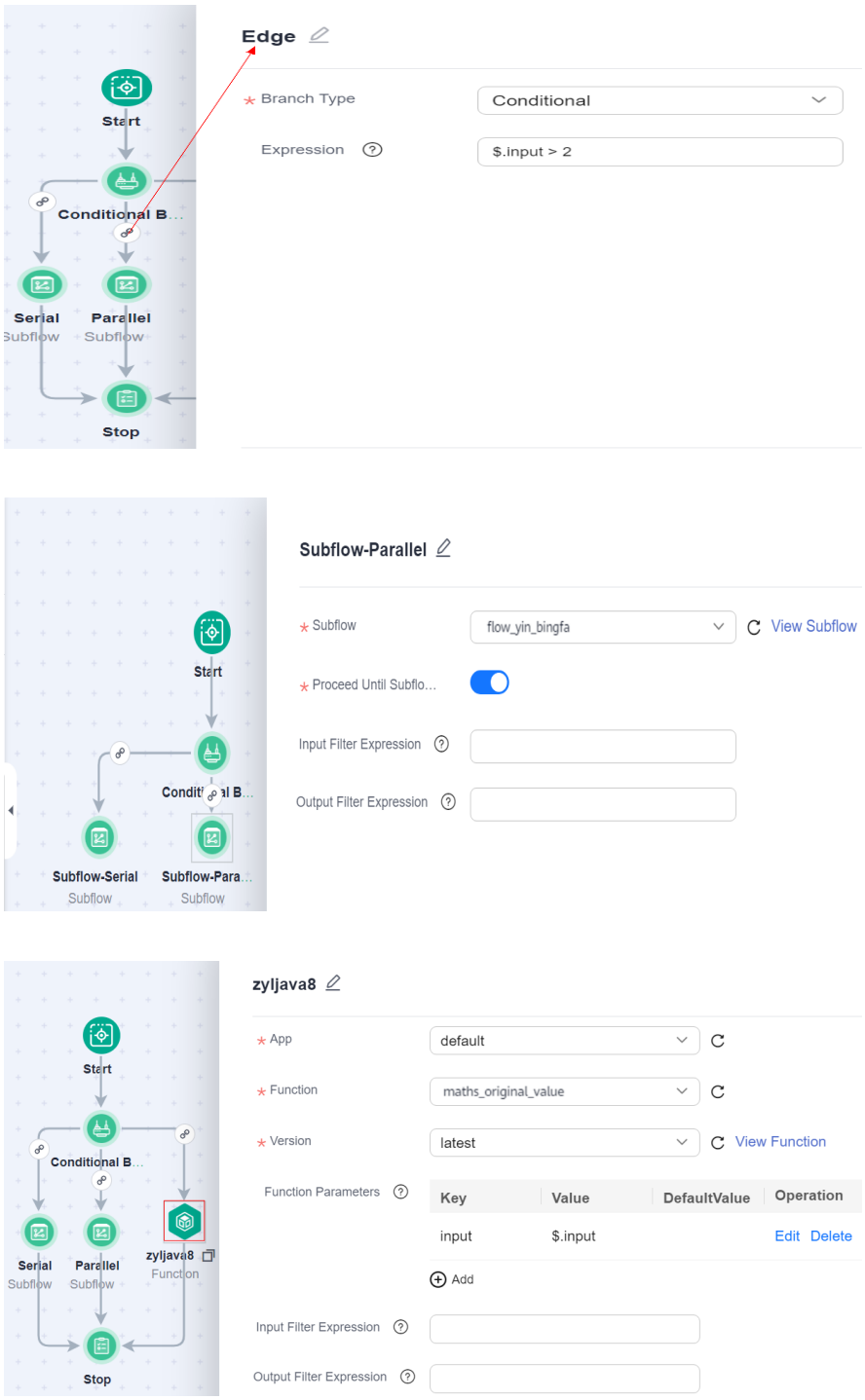
  



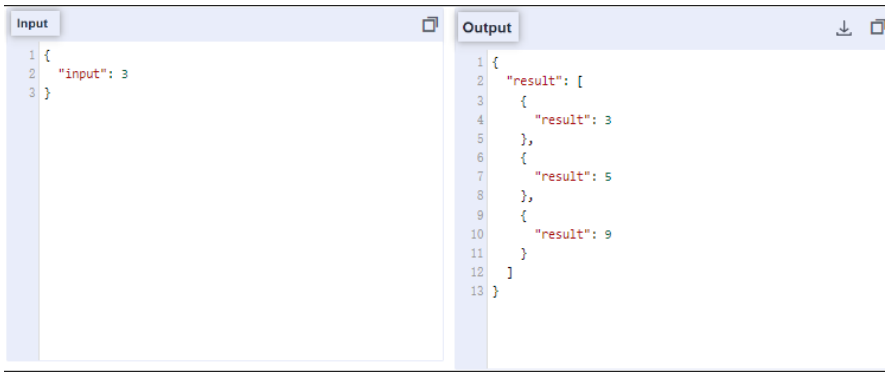






- Input **3** to execute the subflows in parallel mode and output the following result.




- Input **1** to execute the subflows in serial mode and output the following result.

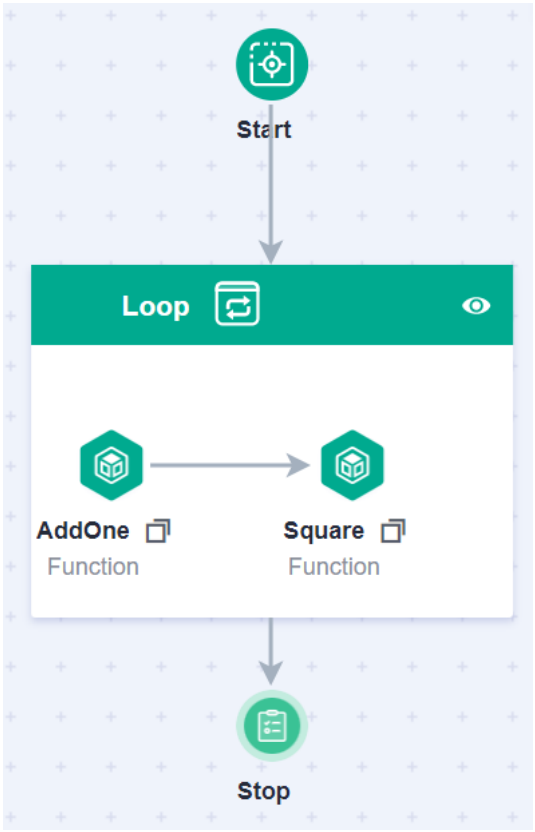

- Input **2** to execute the default branch and output the input value, that is, **2**.


**Example 8**: Loop node

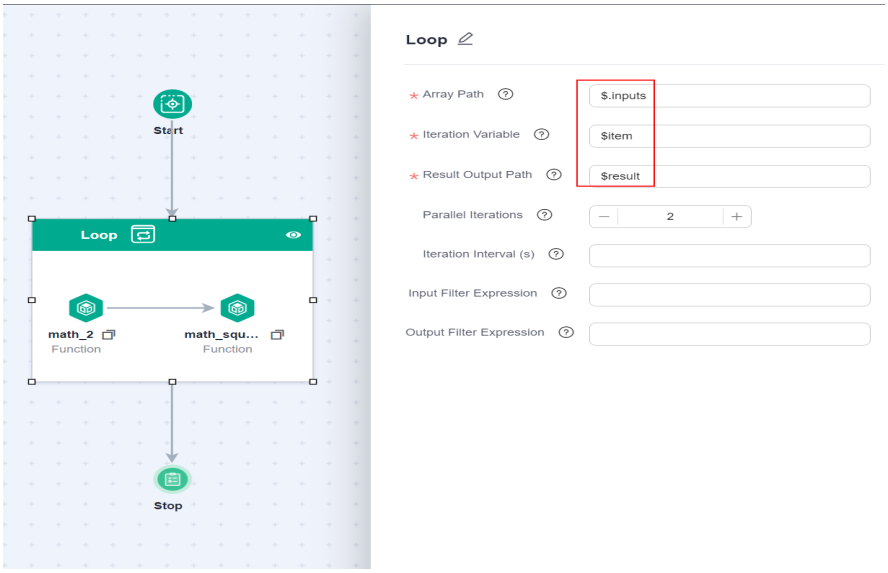
Use the loop node to cyclically process each element in an input array. All subflows in a loop are executed each time.

- Design a function flow with a loop node. The node cyclically adds **2** to each element in an integer array and then squares the sum. The flow is shown in the following figure.

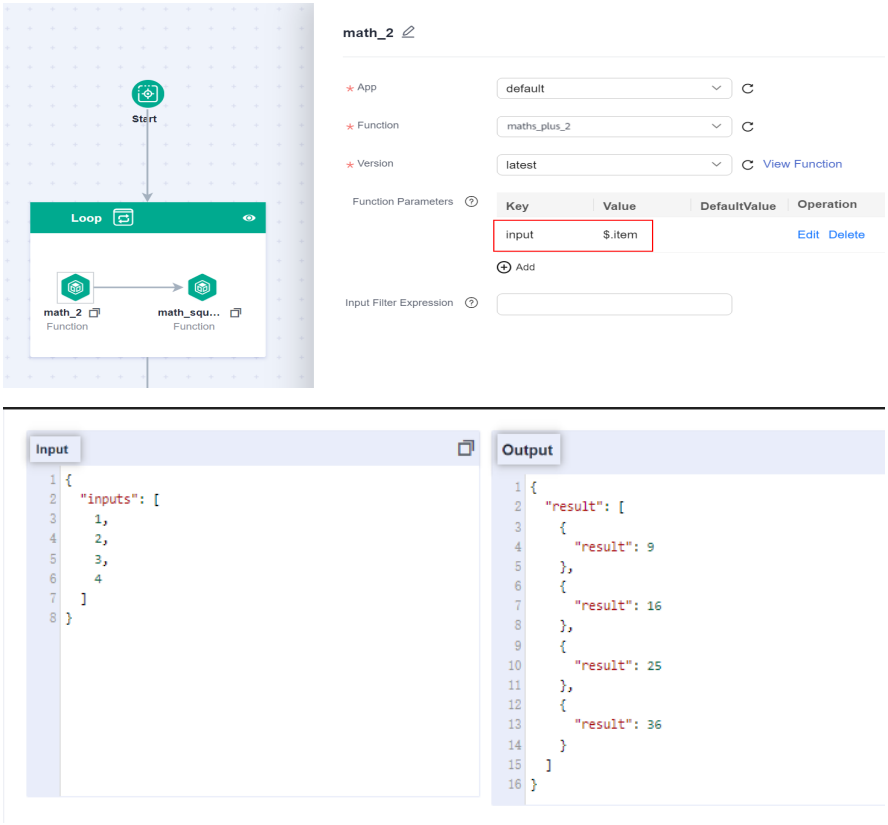



- Set **Array Path** to **$.inputs** and **Iteration Variable** to **$.item**.




- Configure function **math_2** to output a result that is the input value plus 2. Ensure that the parameter value in **Function Parameters** is the same as the iteration variable in the loop node.




# 13.2 Creating a Flow

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.

## Prerequisites

- You have created a function. For details, see **Creating a Function from Scratch**.
- You have understood the **components**, **orchestration rules**, and **expression operators**.

## Procedure

**Step 1** Log in to the FunctionGraph console, and choose **Flows** in the navigation pane.

**Step 2** On the **Function Flows** page, click **Create** next to **Standard Flow** or **Express Flow**.

NOTE

Function flows created on OBS DWR cannot be edited or deleted on the FunctionGraph console. If you need to do so, go to OBS DWR.

**Step 3** Orchestrate a flow for your application.

1. Orchestrate the flow by dragging components.

   Take the flow in **Figure 13-13** as an example. Drag a start node, function, and stop node to the edit box, and connect them using lines.

   **Figure 13-13** Orchestrating a flow

2. Click the function node to edit it. For details about the function parameters, see **Table 13-4**. Parameters marked with an asterisk (*) are mandatory.

   NOTE

   For the function node, select **function 2**, and configure this node by referring to **Figure 13-14**. The result will be the value of the event input plus 2.

   **Figure 13-14** Configuring a function node

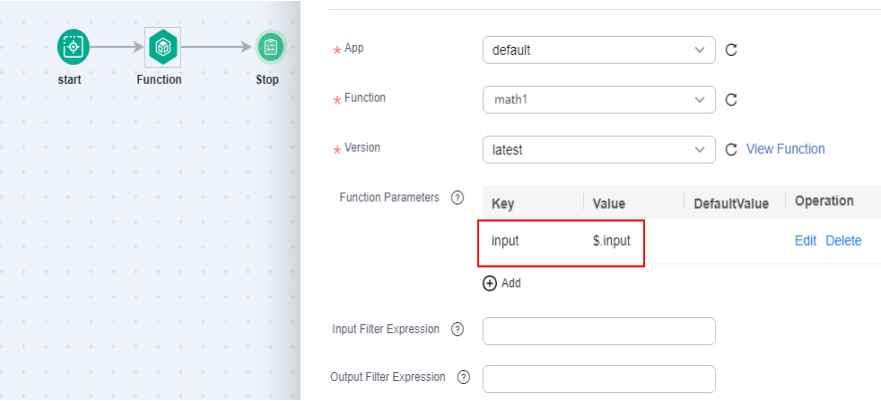
   
   

**Table 13-4** Function parameters

| Parameter | Description |
| --- | --- |
| * App | App to which the desired function belongs. You can create multiple functions under an app. |

| Parameter | Description |
| --- | --- |
| * Function | Function you want to use.<br>**NOTE**<br>– The data returned by the configured function must be in JSON format. Otherwise, it cannot be parsed.<br>– Go functions can return stream data. To enable this feature, choose **Configuration** > **Advanced Settings** on the function details page, and select **Streaming Response**. |
| * Version | Function version you want to use. |
| * CallMode | By default, function nodes in a flow are invoked synchronously.<br>**NOTE**<br>– Synchronous invocation does not support long-term running functions and is max. 15 minutes.<br>– Asynchronous invocation supports long-term running functions. The max. duration of a function node is the same as that supported by FunctionGraph. |
| Function Parameters | 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**: Edit or delete the parameter. |
| Input Filter Expression | Selects part of the JSON output of the previous step to use it as the input of the current step. |
| Output Filter Expression | Selects part of the JSON output of the current step to pass it to the next step. |

3. Click the EG node to edit it. For details about the EG parameters, see **Table 13-5**. Parameters marked with an asterisk (*) are mandatory.

   NOTE

   Before configuring an EG node, ensure that you have created an event source and channel on EG. For details, see **Figure 13-15**.

**Figure 13-15** Configuring an EG node

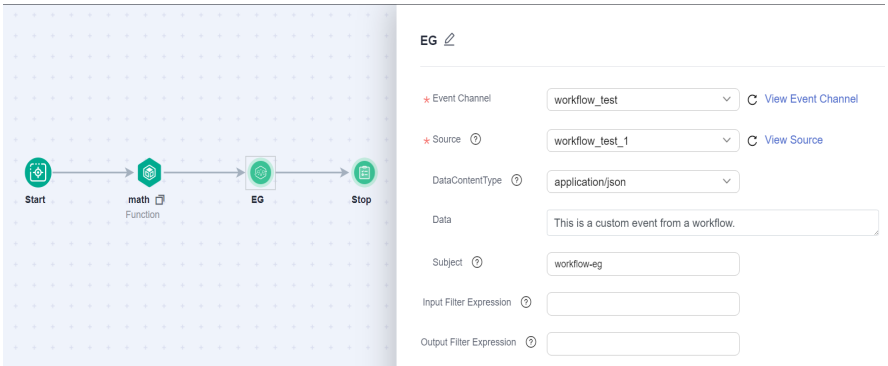



**Table 13-5** EG node parameters

| Parameter | Description |
| --- | --- |
| * Event Channel | Event channels receive events from event sources. Function flows support only custom event channels, which you create to receive events from custom sources. For details, see the event channel description in the EG documentation. |
| * Source | Event sources are applications that produce events. Function flows support only custom application event sources, which publish events to EG through a custom channel.<br>For details, see the event source description in the EG documentation. |
| DataContentType | The format of **Data**, that is, event payload.<br>**NOTE**<br>Only **application/json** is supported. |
| Data | Event content. |
| Subject | A subject or object on which an event occurs. |
| Input Filter Expression | Selects part of the JSON output of the previous step to use it as the input of the current step. |
| Output Filter Expression | Selects part of the JSON output of the current step to pass it to the next step. |

4. Configure processors by referring to **Table 13-6**. Parameters marked with an asterisk (*) are mandatory.

**Table 13-6** Processor parameters

| Category | Parameter | Description |
| --- | --- | --- |
| Callback | * Callback timeout (min) | If no response is returned within the specified timeout, the node fails. |
| | * Callback Type | Type of the service node that can be called back by a function flow.<br>**NOTE**<br>Only function nodes are supported. |
| | * App | App to which the desired function belongs. You can create multiple functions under an app. |
| | * Function | Function you want to use.<br>**NOTE**<br>– The data returned by the configured function must be in JSON format. Otherwise, the data cannot be parsed.<br>– Go functions can return stream data. To enable this feature, choose **Configuration** > **Advanced Settings** on the function details page, and select **Streaming Response**. |
| | * Version | Function version you want to use. |
| | * CallMode | By default, function nodes in a flow are invoked synchronously.<br>**NOTE**<br>– Synchronous invocation does not support long-term running functions and is max. 15 minutes.<br>– Asynchronous invocation supports long-term running functions. The max. duration of a function node is the same as that supported by FunctionGraph. |
| | Function Parameters | 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**: Edit or delete the parameter. |
| | Input Filter Expression | Selects part of the JSON output of the previous step to use it as the input of the current step. |

| Category | Parameter | Description |
| --- | --- | --- |
| | Output Filter Expression | Selects part of the JSON output of the current step to pass it to the next step. |
| Subflow | Subflow | Select a flow. |
| | Proceed Until Subflow Completed | This option is enabled by default. |
| | Input Filter Expression | Selects part of the JSON output of the previous step to use it as the input of the current step. |
| | Output Filter Expression | Selects part of the JSON output of the current step to pass it to the next step. |
| Parallel Branch | * Completion Criterion | – **All branches completed**: suitable when there are two or more branches<br>– **1 branch completed**: suitable when there is only one branch<br>– **Set number of branches completed**: suitable for one of the two or more branches if possible |
| | Input Filter Expression | Selects part of the JSON output of the previous step to use it as the input of the current step. |
| | Output Filter Expression | Selects part of the JSON output of the current step to pass it to the next step. |
| | Set Branches | Set the number of branches that must be completed if you set **Completion Criterion** to **Set number of branches completed**. |
| | * 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**. |
| Start | Triggers | Start of a flow. Each flow has only one start node. For details about how to create a trigger, see **Creating a Flow Trigger**. |

| Category | Parameter | Description |
| --- | --- | --- |
| Error Handling | * Retry | Determines the next step in case of an execution failure.<br>– **Retry Condition(JSONPath)**: Example: **$.status == 500**<br>– **Retry Interval (1-30s)**: The default value is 1s.<br>– **Maximum Retries (1-8)**: The default value is 3. |
| | * Retry Condition | A JSONPath expression. Result **true** indicates that an error is captured. The error handling node is executed then. |
| Loop | * Array Path | Path of the array variable to be traversed. |
| | * Iteration Variable | This variable will be replaced by the name of an element in the array during cyclic iteration. |
| | * Result Output Path | Output path of the result array for all iteration branches. |
| | Parallel Iterations | Number of iteration branches to be run concurrently. Value range: 0–100. **0** indicates unlimited. |
| | Iteration Interval (s) | Interval between parallel iterations. |
| | Input Filter Expression | Selects part of the JSON output of the previous step to use it as the input of the current step. |
| | Output Filter Expression | Selects part of the JSON output of the current step to pass it to the next step. |
| Wait | * Latency (s) | Default value: 1000s |
| Service | Execution Mode | Way that functions in this node will be executed.<br>– **Serial**: Functions will be executed in the order they are connected.<br>– **Parallel**: Functions will be executed concurrently. |
| | 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. |

| Category | Parameter | Description |
| --- | --- | --- |
| Conditional Branch | * Branch Type | – Conditional<br>– Default<br>If there is a conditional branch, a default branch is required. |
| | Expression | JSONPath expression required for a conditional branch. For details, see **Expression Operators**. |
| | 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. |
| Stop | Stop | Not followed by any other node. |

5. After configuring all nodes, click **Save** in the upper right corner.

   NOTE

   If a node in the flow is modified, save the changes before starting the flow.

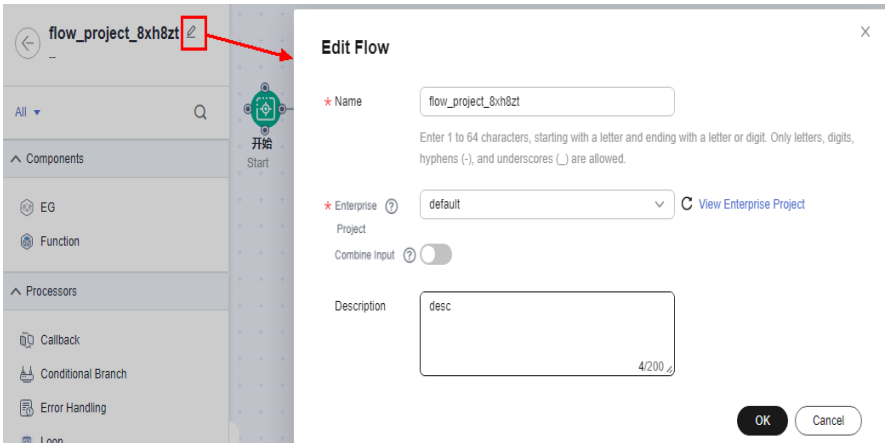
6. On the **Create Flow** page, click on the right of the function flow name in the upper left corner, set parameters, and click **OK**.

**Table 13-7** Configuration

| Parameter | Description |
| --- | --- |
| * Name | Flow name. |
| * Enterprise Project | Select an enterprise project. |
| Combine Input | Whether to combine the previous node's output with the current node's input. |
| Description | Description about the flow. |

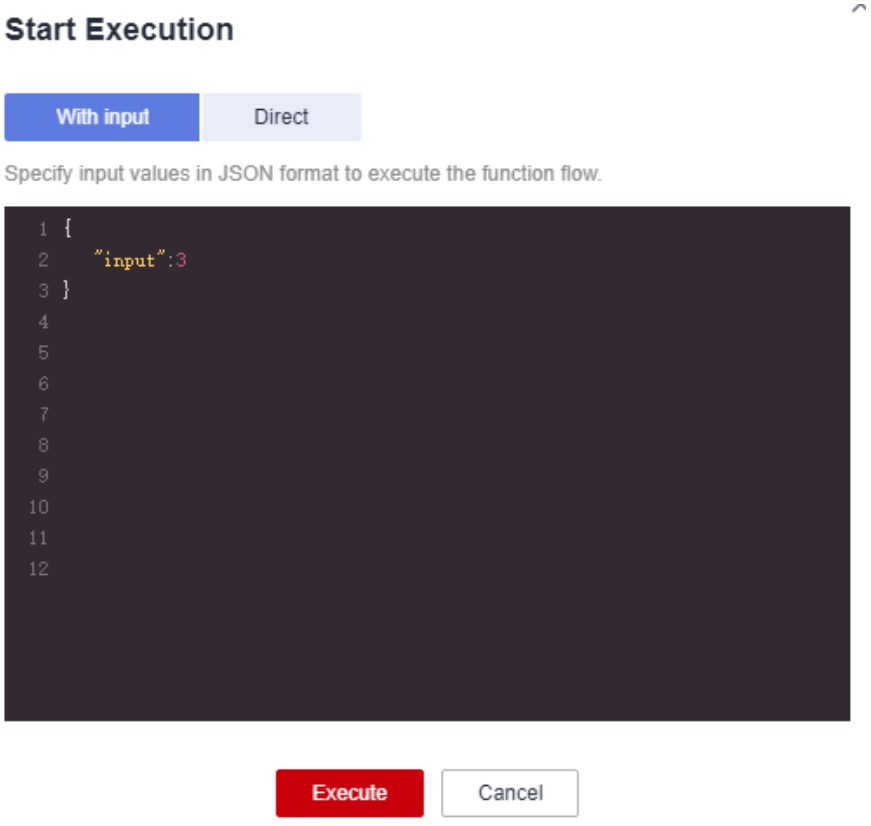
**Figure 13-16** Editing a flow




7. Click **Start**. On the **Start Execution** page, specify input or start the flow directly. In this example, select **With input**.

```
{
  "input":3
}
```

**Figure 13-17** Starting execution




**NOTE**

The input values must be in JSON format.

8. Click **Execute**. A message indicating that the flow is started successfully is displayed in the upper right corner.

9. Click the flow name to go to the details page and view the execution result.

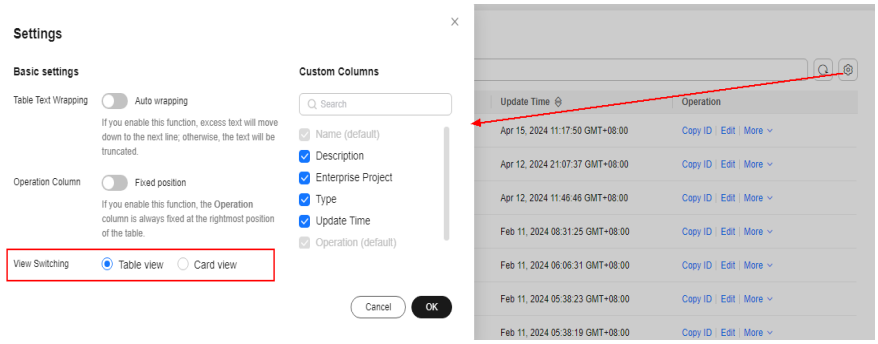
**----End**

## Viewing Flows

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

**Step 2** View flows in card or list mode.

**Figure 13-18** Selecting a display mode



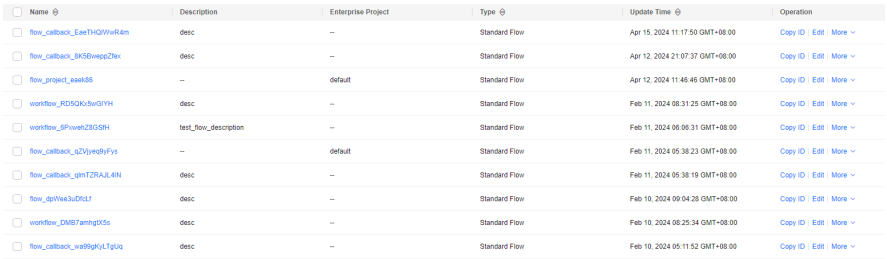

- Card mode

  **Figure 13-19** Card mode

- List mode

  **Figure 13-20** List mode

**Step 3** View all flows that you have created, and perform the operations described in **Table 13-8** as required.

**Table 13-8** Operation description

| Operation | Description |
| --- | --- |
| Edit | Click **Edit** to modify a flow. |
| Start | Choose **More** > **Start** to execute a flow. |

| Operation | Description |
| --- | --- |
| 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 execution records

  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 metrics

  On the **Monitoring** tab page, view the number of invocations, duration, number of errors, and number of executions that are in progress. **Table 13-9** describes the metrics.

  **Table 13-9** 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. |
| Throttles | Count | Number of times that a flow failed to be executed due to rate limiting. |

**----End**

# 13.3 Managing Flow Executions

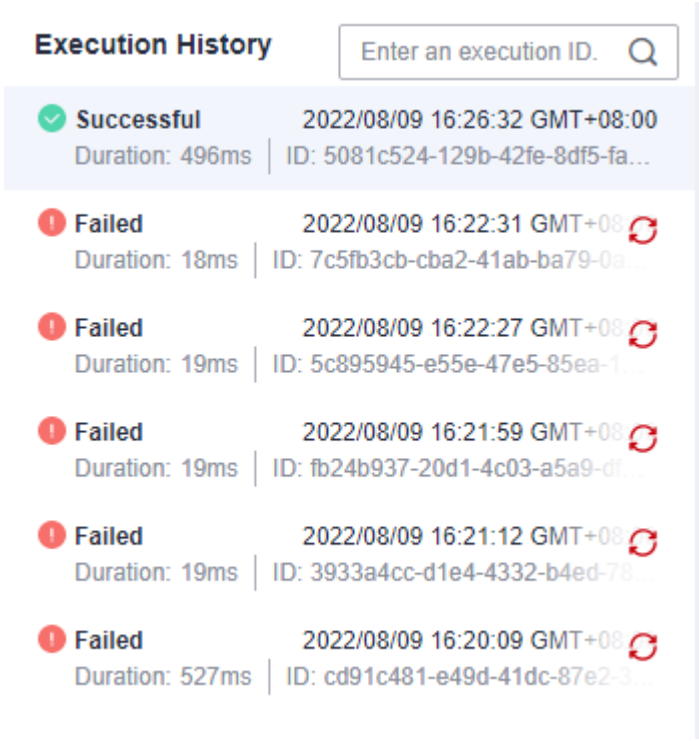
## Viewing the Execution History of a Standard Flow

**Step 1** Log in to the FunctionGraph console, and choose **Flows** in the navigation pane.

**Step 2** On the **Flows** page, click the target flow to go to the details page.

**Step 3** Click the **Executions** tab and view the executions.

**Step 4** Search for an execution by ID. On the left shows the latest 100 executions of the flow.




- 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.

  **Figure 13-21** Execution failed

- Below the canvas is the input and output of this flow. Click a node on the canvas to view its input and output.

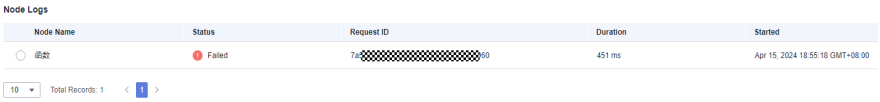
  


**NOTE**

Output fields with value **null** will not be displayed.

- The logs area in the bottom displays the execution records of all nodes.

**Figure 13-22** Node logs




**----End**

## Viewing Execution Logs of an Express Flow

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

**Step 2** On the **Flows** page, click the target function flow to go to the **Basic Information** page.

**Step 3** Click the **Executions** tab and view the execution logs.

**Step 4** Filter logs by request ID or time range. Then click a request ID to view log details.

**Figure 13-23** View execution history


**----End**

## Retrying an Execution

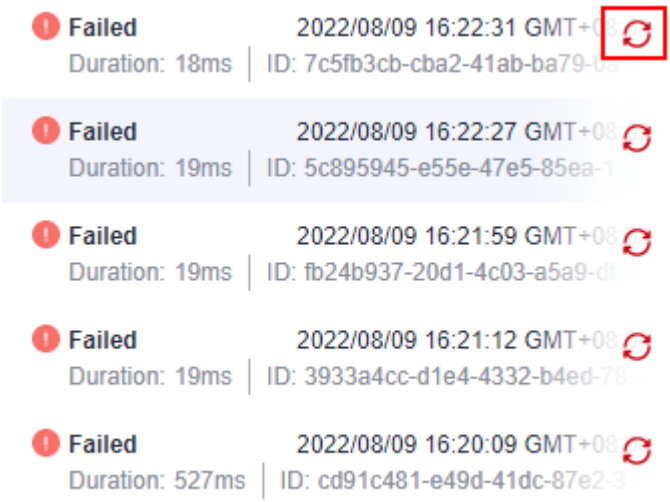
**Step 1** Return to the FunctionGraph console, and choose **Flows** in the navigation pane.

**Step 2** On the **Flows** page, click the target flow to go to the details page.

**Step 3** Click the **Executions** tab and view the executions.

**Step 4** Click the retry icon on the right of a failed execution. If the retry is successful, a new record is generated.




**----End**

### Stopping an Execution

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

**Step 2** On the **Flows** page, click the target flow to go to the details page.

**Step 3** Click the **Executions** tab, and click the stop icon on the right of the target execution. After the execution is stopped, its status changes to **Cancel**.

**----End**

## 13.4 Creating a Flow Trigger

Flow trigger types include APIG, SMN, and timer.

### Creating a Timer Trigger

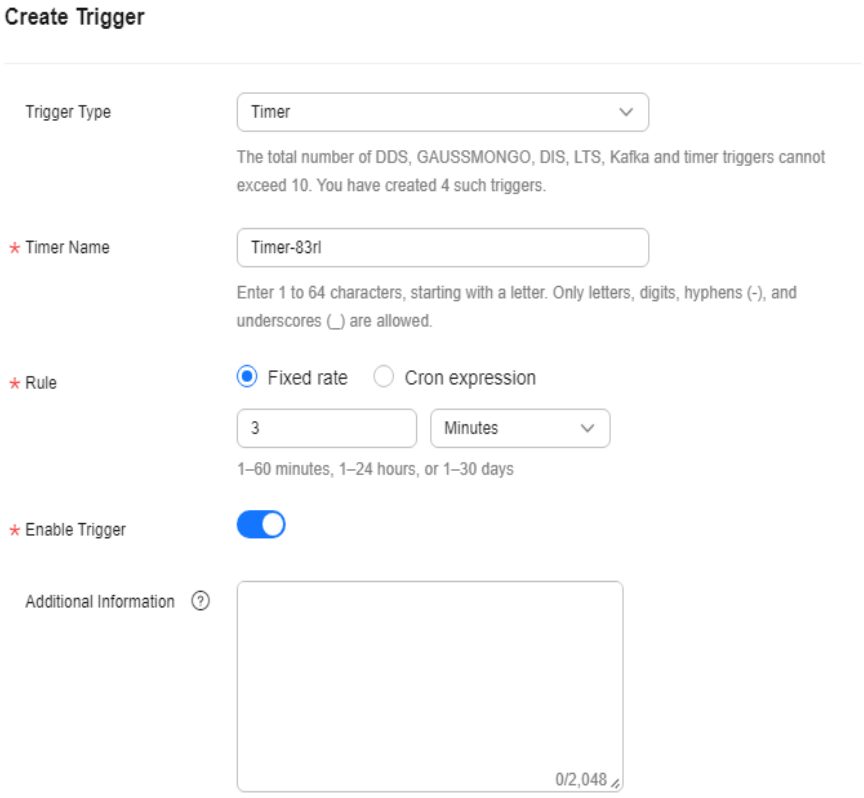
**Step 1** Log in to the FunctionGraph console, and choose **Flows** in the navigation pane.

**Step 2** On the **Flows** page, find the target flow, and click **Edit**.

**Step 3** Click the **Start** node. On the displayed page, click **Create Trigger** and set **Trigger Type** to **Timer**.

**Figure 13-24** Timer trigger




**Step 4** Set the trigger information. As shown in **Table 13-10**, parameters with an asterisk (*) are mandatory.

**Table 13-10** Timer trigger information

| Parameter | Description |
| --- | --- |
| * 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 Expressions for a Function Timer Trigger**. |
| Additional Info | Must be in JSON format and contain an input. The value of input will be passed to the flow. |

**Step 5** Click **Create**.

**----End**

## Creating an APIG (Shared) Trigger

NOTE

The shared gateway is no longer available. Only existing customers who previously used this feature can continue using it.

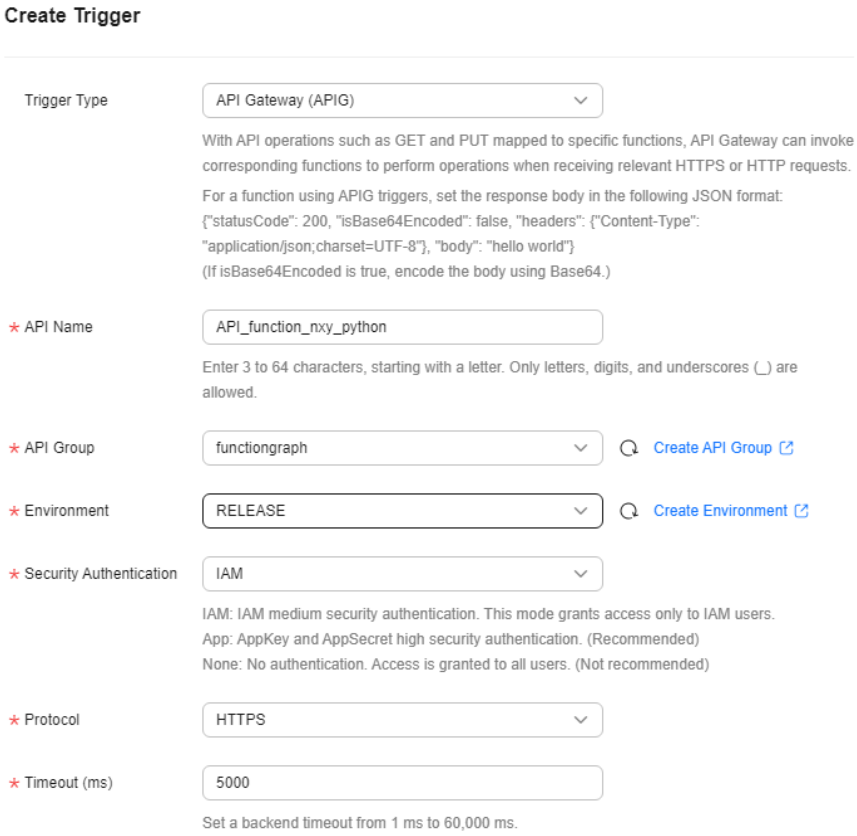
APIG flow triggers support only IAM authentication.

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

**Step 2** On the **Flows** page, find the target flow, and click **Edit**.

**Step 3** Click the **Start** node. On the displayed page, click **Create Trigger** and set **Trigger Type** to **API Gateway (shared)**.

**Figure 13-25** API Gateway (shared) trigger




**Step 4** Set the trigger information. As shown in **Table 13-11**, parameters with an asterisk (*) are mandatory.

**Table 13-11** APIG (shared) trigger information

| Parameter | Description |
| --- | --- |
| * API Group | An API group facilitates management of APIs used for the same service.<br>In this example, select **APIGroup_test**. |

| Parameter | Description |
| --- | --- |
| * Environment | An API can be called in different environments, such as production, test, and development. APIG supports environment management, allowing you to define different request paths for an API in different environments.<br>To ensure that the API can be called, select **RELEASE**. |
| * Visibility | Options: **Public** and **Private**. |
| * 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. |

**Step 5** Click **Create**.

**----End**

## Creating an APIG (Dedicated) Trigger

NOTE

- 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**.

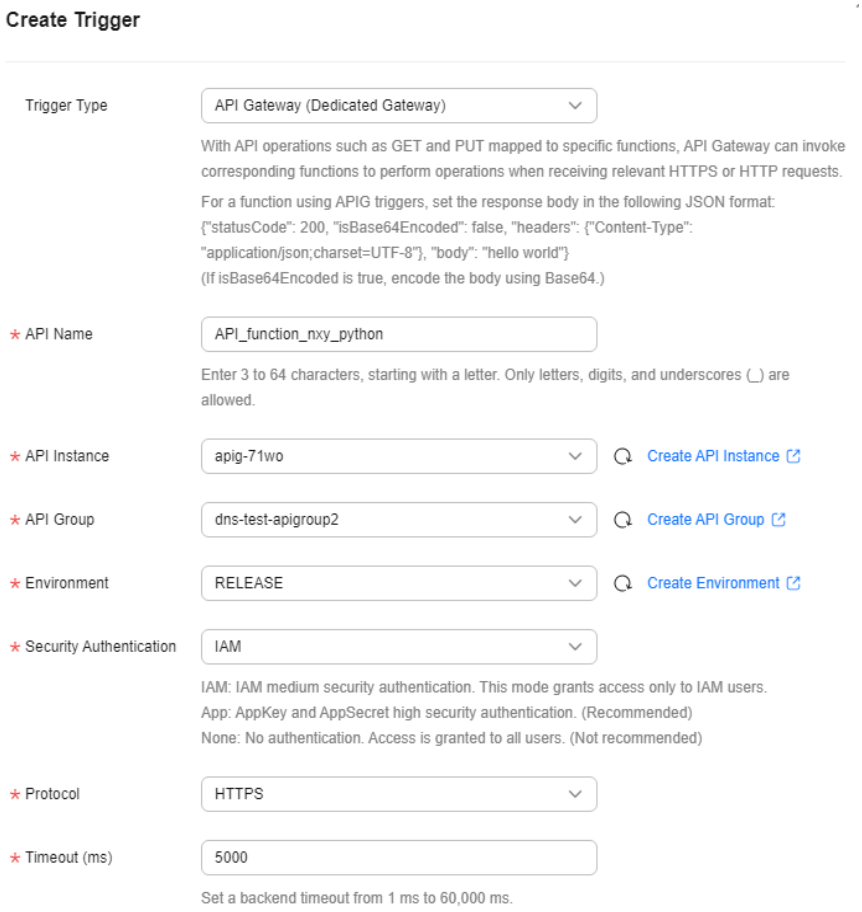
**Step 1** Return to the FunctionGraph console, and choose **Flows** in the navigation pane.

**Step 2** On the **Flows** page, find the target flow, and click **Edit**.

**Step 3** Click the **Start** node. On the displayed page, click **Create Trigger** and set **Trigger Type** to **API Gateway (dedicated)**.

**Figure 13-26** API Gateway (dedicated) trigger




**Step 4** Set the trigger information. As shown in **Table 13-12**, parameters with an asterisk (*) are mandatory.

**Table 13-12** APIG (dedicated) trigger information

| Parameter | Description |
| --- | --- |
| * Gateway | Select a dedicated gateway. |
| * API Group | An API group facilitates management of APIs used for the same service.<br>In this example, select **APIGroup_test**. |
| * Environment | An API can be called in different environments, such as production, test, and development. APIG supports environment management, allowing you to define different request paths for an API in different environments.<br>To ensure that the API can be called, select **RELEASE**. |
| * Visibility | Options: **Public** and **Private**. |
| * Path | The path for requesting the API.<br>Format: **/users/projects** |

| Parameter | Description |
| --- | --- |
| * 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. |

**Step 5** Click **Create**.

**----End**

## Creating an EG Trigger

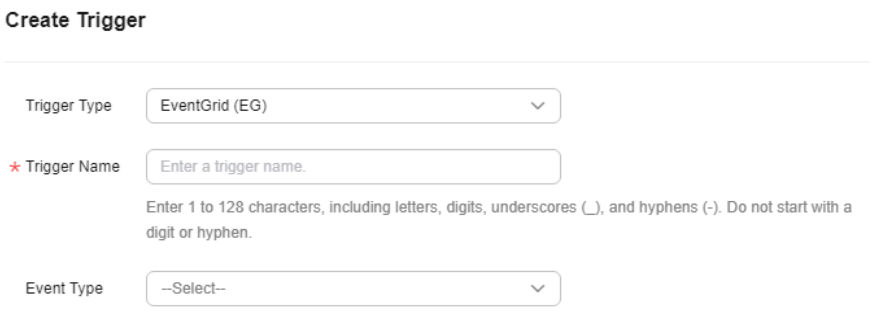
**Step 1** Return to the FunctionGraph console, and choose **Flows** in the navigation pane.

**Step 2** On the **Flows** page, find the target flow, and click **Edit**.

**Step 3** Click the **Start** node. On the displayed page, click **Create Trigger** and set **Trigger Type** to **EventGrid**.

**Step 4** Select an event channel and source. Events from this source will trigger the flow.

**Figure 13-27** EG trigger




**Step 5** Set the trigger information, as shown in **Table 13-13**. Parameters with an asterisk (*) are mandatory.

**Table 13-13** Timer trigger information

| Parameter | Description |
| --- | --- |
| * Provider | EG event source provider. Huawei Cloud and custom event sources are supported. |
| * Event channel | Receives events from the specified source to trigger the function flow. |
| * Event source name | Event sources include Huawei Cloud services and custom applications. They produce events and publish them to EG. |

| Parameter | Description |
| --- | --- |
| Event types | Types of events that will trigger the function flow. |

**Step 6** Click **Create**.

**----End**

# 13.5 Processing Stream Files

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

## Background and Value

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

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

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

## Principles

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

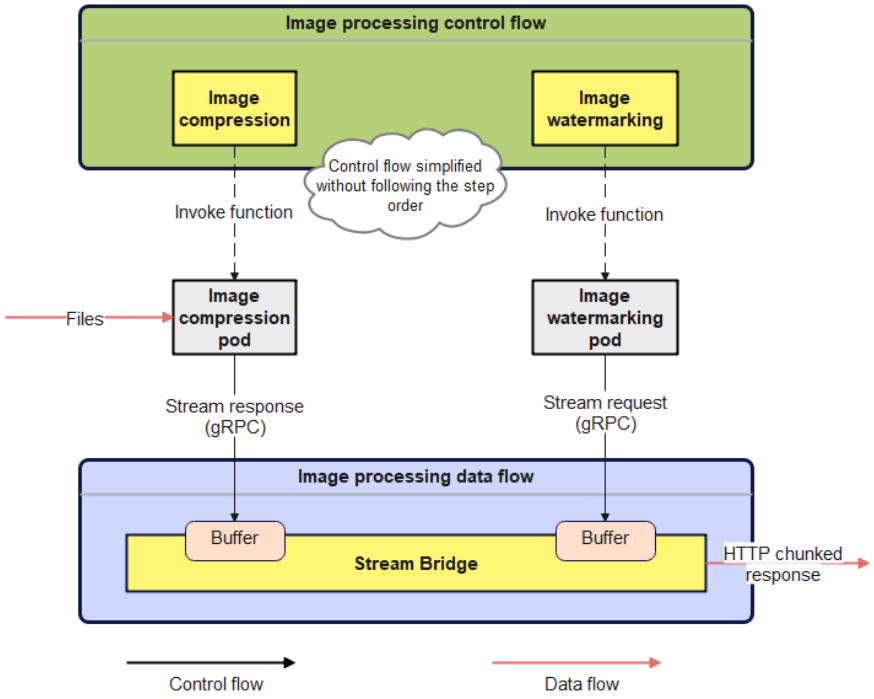
A workflow system needs to process two parts:

- Control flow: Controls the flow between steps and the execution of serverless functions in the steps.
- Data flow: Controls the data flowing through the entire workflow. Generally, the output of a previous step serves as the input of the next step. For example, in the preceding image processing workflow, the image compression result is the input of the watermarking step.

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

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




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

## Procedure

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

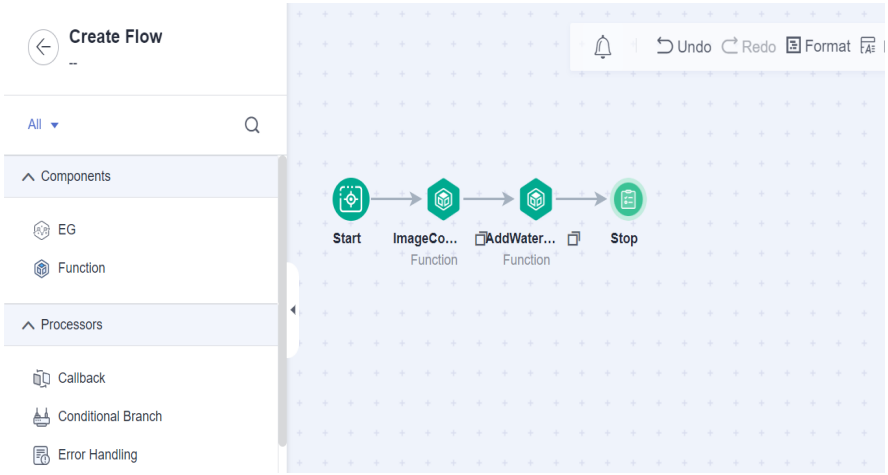
NOTE

Currently, only Go functions are supported.

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

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

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

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

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

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

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




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

**----End**

# 14 Increasing Resource Quota

## Overview

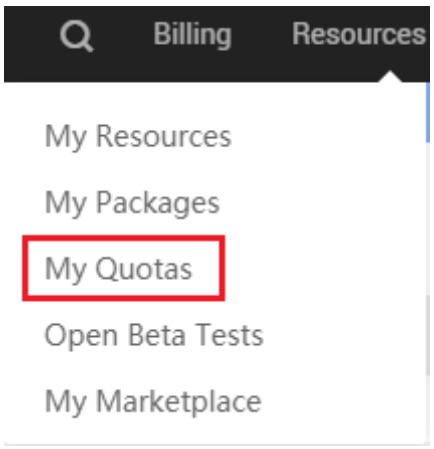
A quota is a limit on the quantity or capacity of a certain type of service resources that you can use. For example, the maximum number of ECSs or EVS disks that can be created.

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 14-1** Going to the Quotas page

3. On the **Quotas** page, view the used and total quotas of resources.

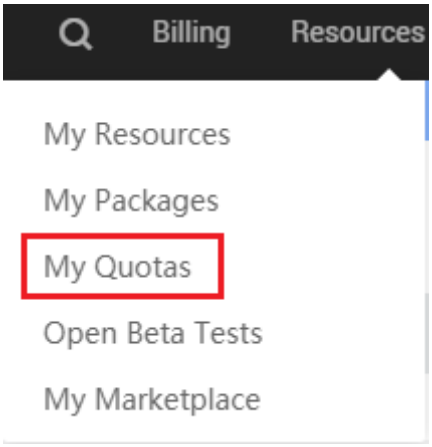
   If a quota cannot meet your needs, apply for a higher quota by performing the following operations.

## 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 14-2** 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**.

# 15 GPU Function Management

## 15.1 Serverless GPU Usage

### 15.1.1 Overview

Serverless GPU is a new cloud computing service that allows for flexible, efficient, and on-demand allocation of GPU computing resources. It provides on-demand GPU computing resources to effectively solve issues such as low resource utilization, high costs, and low scalability caused by long-term use of GPUs. This document describes the functions and advantages of serverless GPUs.

The long-term use of a traditional GPU has many issues, such as pre-planning resource requirements and potential resource wastage. Serverless GPU offers a flexible approach to utilizing GPU computing resources. Users simply need to choose the appropriate GPU model and computing resource scale to efficiently handle acceleration workloads, such as AI model inference and training, accelerated audio and video production, and graphics and image acceleration.

GPU functions provide GPU hardware acceleration for simulation, scientific computing, audio/videos, AI, and image processing to improve service efficiency.

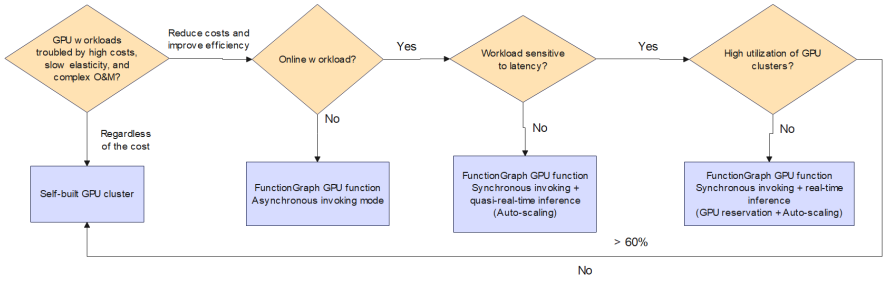
**Table 15-1** GPU function specifications

| Card Type | vGPU Memory (GB) | vGPU Computing Power | Feature |
|---|---|---|---|

| | | | |
|---|---|---|---|
| NVIDIA-T4 | 1–16<br>Must be an integer. | Automatically allocated by the system. | T4 is specifically designed for AI inference workloads, such as neural networks that process video, voice, search engines, and images. The T4 GPU boasts impressive specs, including 16 GB GDDR6 memory, 320 Turing Tensor Cores, and 2,560 Turing CUDA Cores. It also offers exceptional performance and computing capabilities across multiple precisions, including FP32, FP16, INT8, and INT4. Its peak performance is 65T for FP16, 130T for INT8, and 260T for INT4. |

**Figure 15-1** GPU cloud product selection guide




**NOTE**

- This feature is supported only in **CN East-Shanghai1**.
- GPU functions do not support the following network segments: **192.168.64.0/18**, **192.168.128.0/18**, **10.192.64.0/18**, and **10.192.128.0/18**.

## 15.1.2 Scenarios

### 15.1.2.1 Quasi-Real-Time Inference

This section describes what is quasi-real-time inference and how to use on-demand GPU instances to build a cost-effective quasi-real-time inference service.

## Characteristics

In quasi-real-time inference scenarios, a workload has one or more of the following characteristics:

- **Infrequent calls**

  The number of daily calls varies greatly, ranging from a few to tens of thousands. Additionally, the GPUs are typically used for less than 6 to 10 hours per day, resulting in a significant amount of idle resources.

- **Time-consuming processing**

  Quasi-real-time inference usually takes only seconds to minutes. For example, a typical computer vision task can be completed within seconds, while a typical video processing or AIGC task may take a few minutes.

- **Tolerance of cold start**

  The service can tolerate the GPU cold start latency, or the service traffic pattern corresponds to a low probability of cold starts.

## Advantages

FunctionGraph offers the following benefits for quasi-real-time inference workloads:

- **Cloud-native serverless**

  FunctionGraph provisions on-demand GPU resources by default. GPU instances are automatically scaled based on the number of service requests. The minimum number of instances is 0, and the maximum number of instances can be configured.

- **Optimal specifications**

  FunctionGraph offers the best GPU choices, allowing you to select a GPU type and configure the GPU memory based on service requirements.

- **Cost-effectiveness**

  The pay-per-use billing mode reduces costs by more than 70% for workloads with low GPU resource utilization.

# 15.1.2.2 Real-Time Inference

## Characteristics

In real-time inference application scenarios, a workload has one or more of the following characteristics:

- **Low latency**

  Each request needs to be handled promptly, with a strict response time (RT) delay. 90% of the long-tail latency is typically within the range of hundreds of milliseconds.

## Advantages

FunctionGraph offers the following benefits for real-time inference workloads:

- **Reserved GPU instances**

  In addition to the default on-demand GPU instances, FunctionGraph provides reserved GPU instances. You can use them to eliminate the impact of cold start latency to meet the low-latency response requirements of real-time inference services. For details, see **Reserved Instance Management**.

- **Prioritized quality with optimized cost**

  The billing cycle of a reserved GPU instance is different from that of an on-demand GPU instance. A reserved GPU instance is billed based on its lifecycle, regardless of whether the instance is active or idle (not charged by request). Compared to on-demand GPU instances, the overall cost is higher, but it is reduced by over 50% when compared to self-built long-term GPU clusters.

- **Optimal specifications**

  FunctionGraph offers the best GPU choices, allowing you to select a GPU type and configure the GPU memory with the minimum specifications as small as 1 GB based on service requirements.

- **Scaling for traffic surge**

  FunctionGraph offers ample GPU resources that can be quickly scaled out to prevent service disruptions due to insufficient or delayed GPU computing power during traffic surges.

### 15.1.2.3 Offline Asynchronous Task

## Characteristics

In an offline asynchronous application scenario, a workload has one or more of the following characteristics:

- **Long execution time**

  Service processing takes minutes to hours, and the response time is insensitive.

- **Immediate return after submission**

  The system provides an immediate response upon receiving the call, ensuring that the lengthy processing does not obstruct the execution of the primary service logic.

- **Real-time task status monitoring**

  None.

- **Parallel processing**

  To handle large amounts of data and meet high GPU resource demands, offline GPU tasks require parallel processing through APIs to accelerate processing.

- **Data source integration**

  An offline GPU task has various requirements on data sources, and needs to frequently interact with multiple storage products (for example, OBS) and multiple message products (for example, message queue) during processing.

## Advantages

FunctionGraph offers the following benefits for offline asynchronous application workloads:

- **Simplified service architecture**

  Asynchronous processing improves system response, resource utilization, and availability.
- **Sufficient GPU resources**

  FunctionGraph provides sufficient GPU resources for any offline services that are always idle or have unpredictable traffic.
- **Data source integration**

  FunctionGraph supports multiple data source triggering modes, such as OBS and message queue.

# 15.2 Deployment Mode

GPU functions can be deployed with custom images or runtimes.

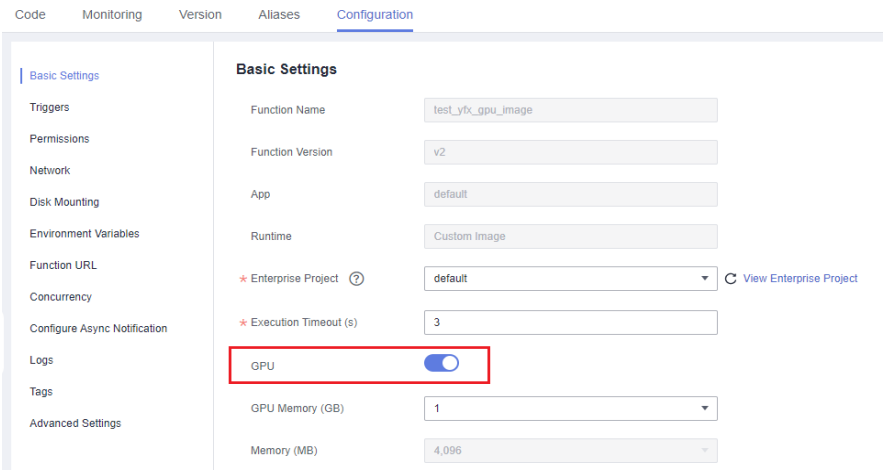
## 15.2.1 Deployment with a Custom Image

NVIDIA Tesla GPUs are supported. For example, Tesla T4.

For details about how to deploy a function with a custom image, see **Deploying a Function Using a Container Image**.

For a function with a custom image, go to the function details page, choose **Configuration** > **Basic Settings**, and enable **GPU**.

**Figure 15-2** Enabling GPU




## 15.2.2 Deployment with a Custom Runtime

GPU functions can be deployed with a custom runtime.

For details about custom runtimes, see **Custom Runtime**.

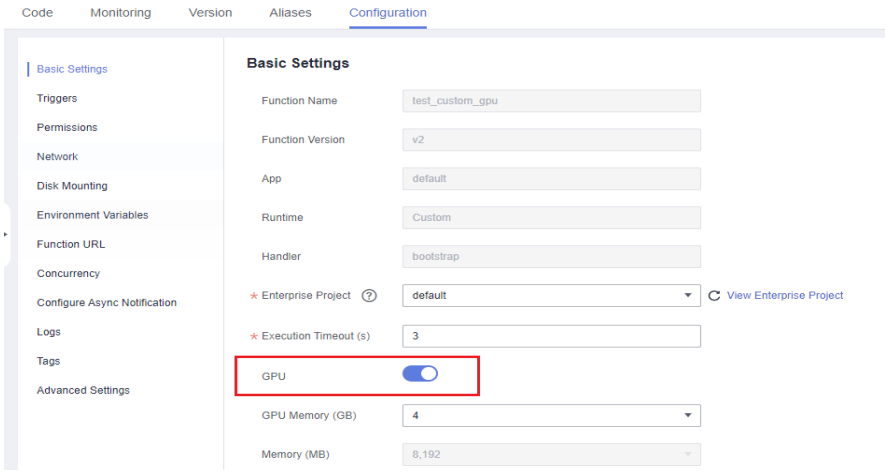
These built-in runtimes are available: Python (Python 2.7.15), Python 3 (Python 3.6.8), Python 3.7 (Python 3.7.4), and Python 3.9 (Python 3.9.2).

Constraint: CUDA 11.6 is pre-integrated for function development. For other CUDA versions, use them with custom image–based functions.

For a function with a custom runtime, go to the function details page, choose **Configuration** > **Basic Settings**, and enable **GPU**.

**Figure 15-3** Enabling GPU




# 15.3 Function Types

GPU functions are available in two types: on-demand and reserved. For details, see **Reserved Instance Management**.

# 16 Application Center

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 (This feature is supported only in **CN North-Beijing4** and **CN East-Shanghai1**.)

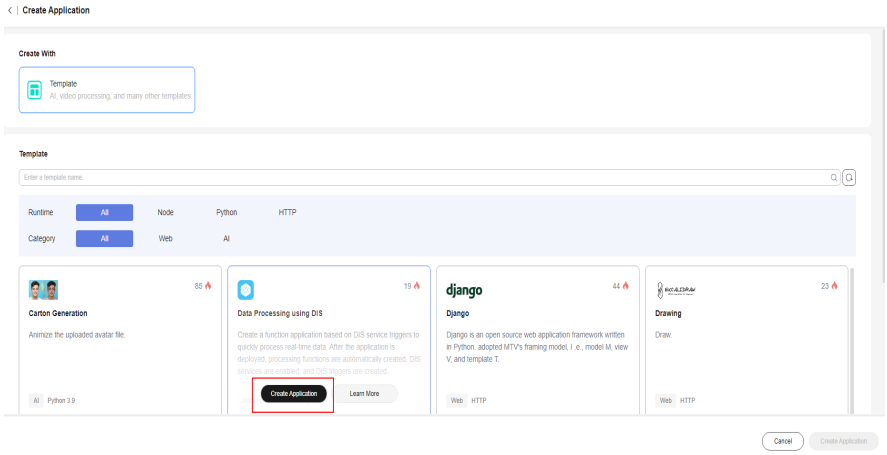
## Procedure

**Step 1** Log in to the **FunctionGraph console**. In the navigation pane, choose **Apps**.

**Step 2** Click **Create** in the upper right corner. The **Select Template** page is displayed.

**Step 3** In the navigation pane on the left, there are two filters: **Runtime** and **Category**. You can filter templates based on service requirements. This section describes how to create a DIS data processing template. After the template is determined, click **Create Application** to go to the application configuration page.

**Figure 16-1** Selecting a template




**Step 4** On the **Application Configuration** page, enter the following information:

- **Region**: Retain the default setting.
- **Project**: Retain the default setting.
- **Application Name**: Enter a custom name.
- **Runtime Language**: Retain the default setting.

- **Agency**: Determine whether to use an agency based on the requirements. For example, in **Step 3**, the DIS data processing template is created and you need to create an agency to authorize the function to access DIS. For details, see **Configuring Agency Permissions**.
- **Description**: Describe the application.

Click **Create Now** after the configuration is complete.

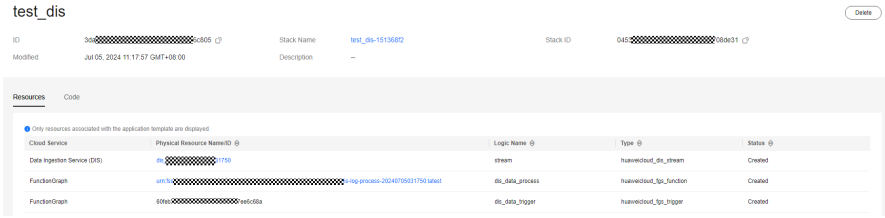
**----End**

## Parameter Description

After the application is created, you can click the application name in the application list to view details. The main parameters are described as follows:

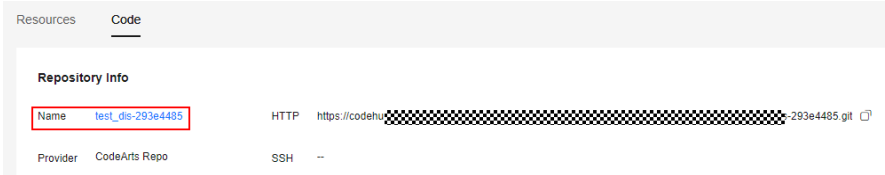
- **Stack Name**: You can click this link to go to the task details page after the stack is successfully deployed.
- **ID**: Unique ID of the currently deployed application in the system. You can locate faults based on the ID and application name.
- **DIS Data Service (DIS)**: You can click this link to go to the details page of the created DIS data service.
- **FunctionGraph**: You can click this link to go to the details page of a created function.

**Figure 16-2** Parameters on the Summary page




- **Name** in the **Repository Info**: You can click this link to go to the code repository where the function code is hosted. You can browse and download related code.

**Figure 16-3** Parameters on the Code page




## Common Problems and Troubleshooting Methods

1. The code repository fails to be created, and the message shown in **Figure 16-4** is displayed.

**Figure 16-4** Code repository creation failed

Troubleshooting method: Check whether CodeArts is enabled for your account. For details, see **Login Methods** to check and enable CodeArts. If the problem persists, contact Huawei Cloud FunctionGraph engineers for further help.

2. The stack fails to be deployed, and the message shown in **Figure 16-5** is displayed.

**Figure 16-5** Stack deployment failed


Troubleshooting method: Click the application name link to go to the **Summary** page of the application. Click the link of **Stack Name** to go to the RFS page. Then click **View Reason** to view the specific problem.

**Figure 16-6** Summary page

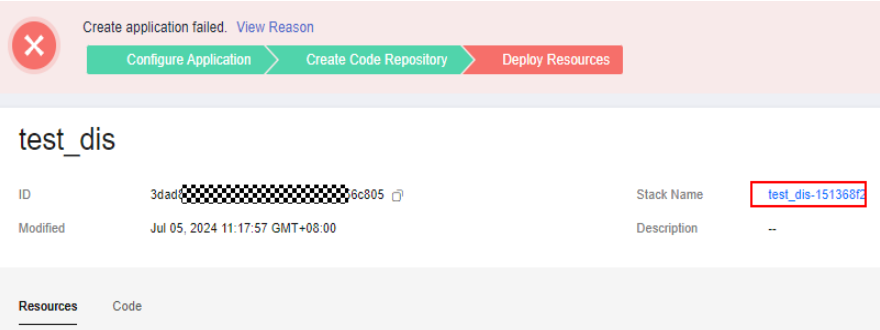


**Figure 16-7** RFS page

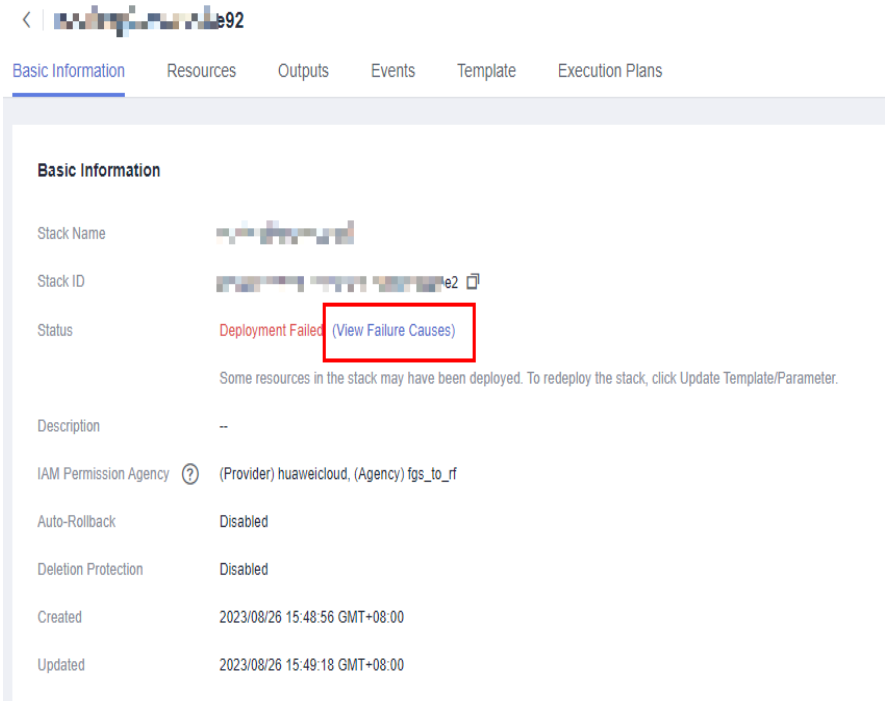
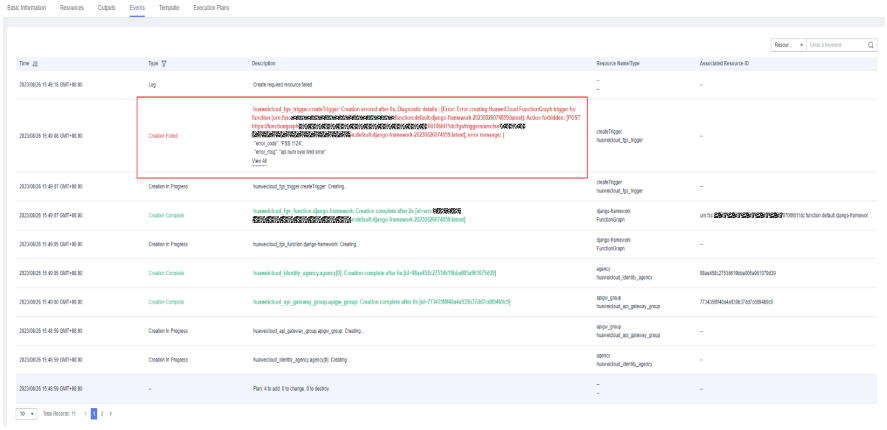
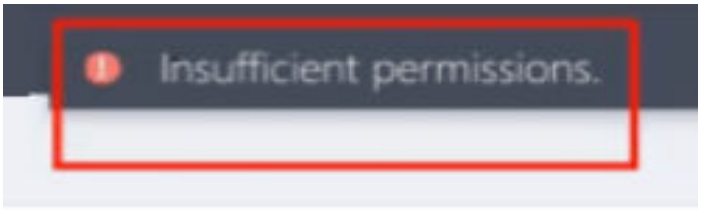



**Figure 16-8** View failure causes



3. If the permission is insufficient, the message shown in **Figure 16-9** is displayed.

**Figure 16-9** Insufficient permissions




Troubleshooting method: 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 and try again.

4. If the deletion fails, the message shown in **Figure 16-10** is displayed.

**Figure 16-10** Deletion failed


Troubleshooting method: Click the application name link to go to the **Summary** page of the application. Click the **Stack Name** link to go to the RFS page. Then click **View Reason** to view the specific problem. The preceding figure is used as an example. The application fails to be deleted because the API group contains APIs. Go to the **Summary** page of the application, click **Physical Resource Name/ID** of the FunctionGraph. On the function details page that is displayed, choose **Settings** > **Triggers** to view the API name. Log in to the APIG console, take the API offline, delete the API, and try again.

**Figure 16-11** Clicking the FunctionGraph link

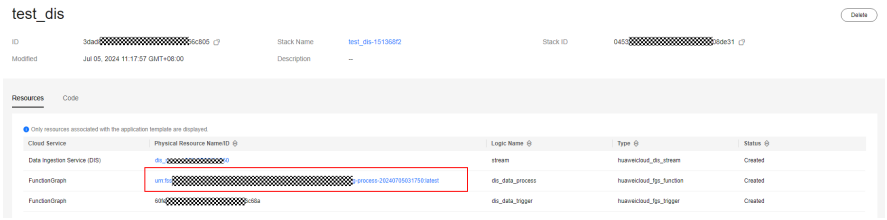
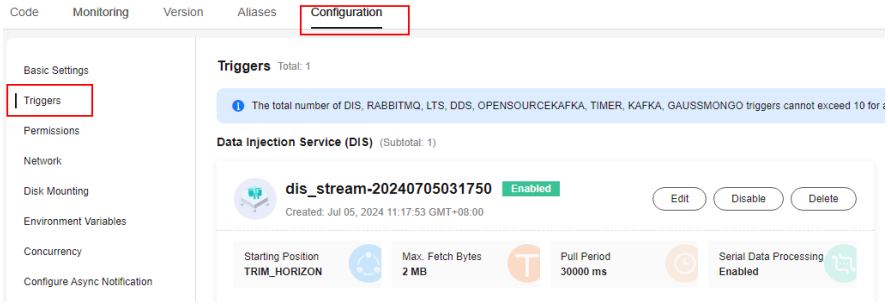



**Figure 16-12** Viewing the API name



# 17 Sharing

## Introduction to Function Sharing

Based on the Resource Access Manager (RAM) service, FunctionGraph allows you to share functions across accounts. 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**.

## Constraints

The prerequisites and constraints for sharing functions are as follows:

- You must be the function owner. You cannot share the functions shared by others.
- 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.

## 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**.

NOTE

- 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**.

## Viewing Share Details

Log in to the FunctionGraph console and choose **Functions** > **Function List** > **Shared Functions** to view functions shared by other accounts.

**Figure 17-1** Viewing shared functions


NOTE

- 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**.

## Using the Shared Functions

**Step 1** Return to the FunctionGraph console and choose **Functions** > **Function List** > **Shared Functions**.

**Step 2** In the list of functions shared by other accounts, click a function name to view and execute the function.

**----End**

NOTE

Before using the shared functions, you need to accept the invitation on the RAM management console. For details, see **Responding to a Resource Sharing Invitation**.

## Stopping a Share

- 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**.

## Permissions of Shared Functions

The owner and principal have different operation permissions on shared functions, as shown in **Table 17-1**.

**Table 17-1** Permission to use shared functions

| Resource | Owner | Principal |
| --- | --- | --- |
| Functions | Full permissions on the functions. | View and execute shared functions. |

## Supported Resource Types and Regions

Table 17-2 lists the resource types that can be shared by FunctionGraph and the regions where the share is supported.

**Table 17-2** Resources and regions supported by FunctionGraph

| Cloud Service | Resource Type | Applicable Region |
| --- | --- | --- |
| FunctionGraph | Functions | AP-Bangkok<br>AP-Jakarta<br>CN East-Shanghai1<br>CN North-Beijing4<br>CN-Hong Kong<br>AP-Singapore<br>TR-Istanbul<br>ME-Riyadh |

## Billing

None.

# 18 Programmable CDN Function

## 18.1 Creating a CDN Function

### Overview

FunctionGraph provides programmable CDN functions in the AP-Singapore region. You can use FunctionGraph to debug functions on the cloud and deliver them to off-premises CDN environments (available only in AP-Singapore).

NOTE

By default, you do not have the permission to create CDN functions. To use CDN functions, contact Huawei Cloud FunctionGraph engineers to apply for a whitelist.

### Prerequisite

An agency for CDN with permission to call functions has been created.

### Creating an Agency

Before using CDN, you need to create an agency so that CDN can call functions created in FunctionGraph.

**Step 1** Log in to the **CDN console**. In the navigation pane on the left, choose **Domains**.

**Step 2** In the domain list, click the target domain name or click **Configure** in the **Operation** column.

**Step 3** Click the **Functions** tab. If you use FunctionGraph for the first time, click **Authorize Now**. The authorization page is displayed.

**Step 4** After the authorization is configured, click **Authorize**. An agency named **CDNAccessFunctionGraph** will be created on the **IAM console**. CDN then has permission to call your functions in FunctionGraph.

NOTE

Do not delete this created agency. Otherwise, CDN cannot call functions in FunctionGraph.

**----End**

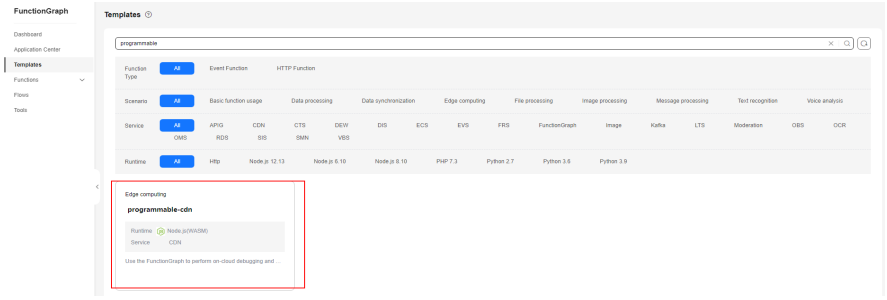
  

## Procedure

**Step 1** Log in to the **FunctionGraph console** and choose **Templates** in the navigation pane.

**Step 2** Enter **programmable-cdn** in the search box in the upper right corner to query the programmable CDN function template. Then click **Configure**.

**Figure 18-1** Programmable CDN template




**Step 3** Configure function details.

- **Region**: Select a region.
- **Project**: Use **default**.
- **Function Name**: Enter a custom name.
- **Enterprise Project**: Retain the default value **default**.

Review the function configuration and click **Create Function**.

**Step 4** On the function details page, click **Code** to compile a function.

**Step 5** Select **Version**, click **Public New Version**, set **Version** and **Description**, and click **OK**.

**Figure 18-2** Publishing a new function version

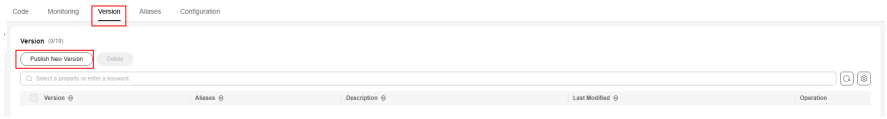
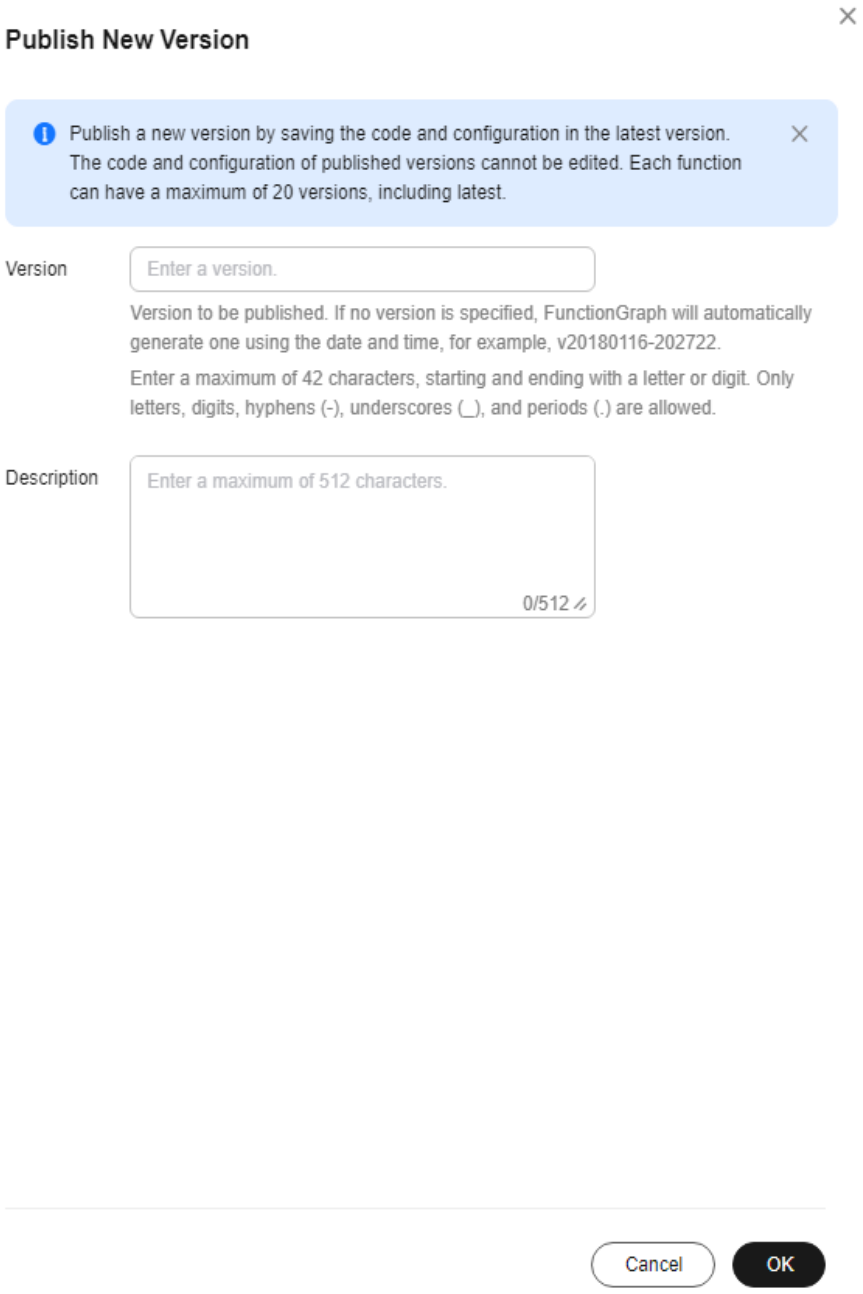


**Figure 18-3** Configuring the version




**----End**

# 18.2 Managing CDN Functions

## Precautions

The priorities for function configurations in CDN and FunctionGraph vary in different stages. The details are as follows:

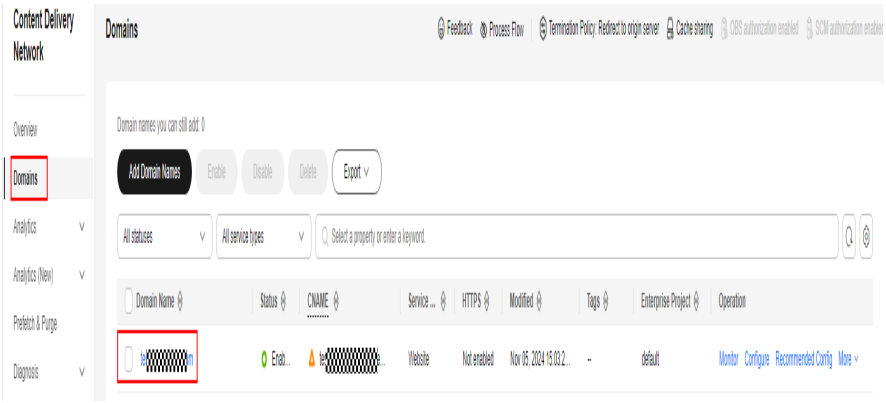
- Client request received: The function is executed in FunctionGraph first and then configured by CDN, so it is the CDN configuration that takes effect at last.

- Before origin pull: The function is configured by CDN first and then executed in FunctionGraph, so it is the FunctionGraph configuration that takes effect at last.

## Procedure

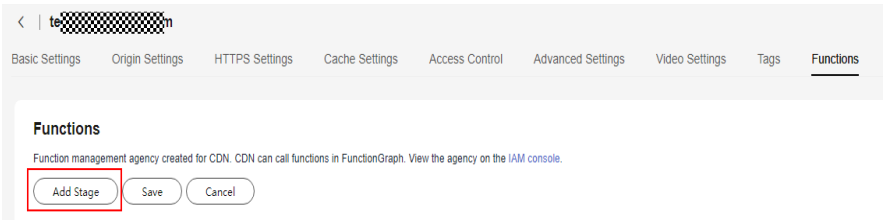
**Step 1** Log in to the **CDN console**. In the navigation pane, choose **Domains**. In the domain name list, click the domain name for which you want to configure a function. The details page is displayed.

**Figure 18-4** Domain name management




**Step 2** Select **Functions**. If you do not have permission to create an agency, see **Creating an Agency**. If you have created an agency, click **Edit** and then click **Add Stage**.

**Figure 18-5** Adding a stage




**Step 3** Configure the function stage and rule.

- Select **Client request received** for **Stage**. For details, see **Table 18-1**.
- Set **Content Type** to **All files**, **Function** to **programmable-cdn**, **Version** to **V1**, and **Priority** to **1**. For details, see. **Table 18-2**. A maximum of 40 rules can be added.

**Figure 18-6** Configuring function stage and rule

**Table 18-1** Function stage description

| Stage | Description |
| --- | --- |
| Client request received | Functions are executed between the time when a client initiates a request and the time when a CDN node receives the request, for example, URL rewriting. |
| Before origin pull | Functions are executed before a CDN node receives a request that is not cached and retrieves the requested resource, for example, rewriting the retrieval URL. |

**Table 18-2** Rule parameters

| Parameter | Description |
| --- | --- |
| Content Type | **All files**: The rule takes effect for all files.<br>**File type**: Files with the specified types comply with the current rule.<br>**Directory**: Files in the specified directory comply with the current rule.<br>**Full path**: Files in the full path comply with the current rule. |
| Content | If **Content Type** is set to **All files**, you do not need to set this parameter.<br>If **Content Type** is set to **File type**:<br>• Enter a period (.) as the first character.<br>• Use commas (,) to separate multiple file types, for example, **.jpg,.zip,.exe**.<br>• Enter up to 20 file types.<br>If **Content Type** is set to **Directory**:<br>• Enter a directory starting with a slash (/).<br>• Use commas (,) to separate multiple directory paths, for example, **/test/folder01,/test/folder02**.<br>• Enter up to 20 directory paths.<br>If **Content Type** is set to **Full path**:<br>• Enter a path start with a slash (/), for example, **/test/index.html** or **/test/*.jpg**.<br>• A file in the specified directory or file with the wildcard * can be matched. |
| Function | Select a function created in FunctionGraph. If no function is available, click **Create Function** to create one. |
| Function version | Select the related function version. |

| Parameter | Description |
| --- | --- |
| Priority | If there are multiple rules in a function stage, the priority determines which rule is executed first.<br>• Enter an integer ranging from 1 to 100. A greater number indicates a higher priority.<br>• The priority must be unique. Two rules cannot have the same priority. |

**Step 4** Click **Save**.

The deployment takes about 5 minutes. After it is complete, the function configuration will be deployed on the edge node. Then you can access CDN to trigger functions and run scripts on CDN edge nodes.

**Figure 18-7** Saving configurations


**----End**

# 19 CLI Command Reference

## 19.1 Introduction to KooCLI

FunctionGraph provides the command line interface (CLI) for you to manage functions, triggers, and aliases, and invoke functions.

### KooCLI Download Links

KooCLI can run on a 64-bit Linux x86 operating system (OS), 64-bit Windows OS, or macOS. **Table 19-1** provides the download links of KooCLI.

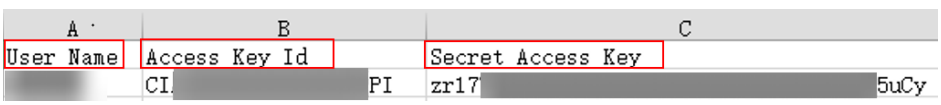
**Table 19-1** Download links of CLI

| OS | Software Package and Verification File | Reference |
| --- | --- | --- |
| Linux | KooCLI and Verification File | KooCLI Overview |
| Windows | | |
| macOS | | |

## 19.2 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 19-1** 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 19-2** Obtaining region information

| Region Name | Region |
| --- | --- |
| AF-Johannesburg | af-south-1 |
| AP-Bangkok | ap-southeast-2 |
| AP-Singapore | ap-southeast-3 |

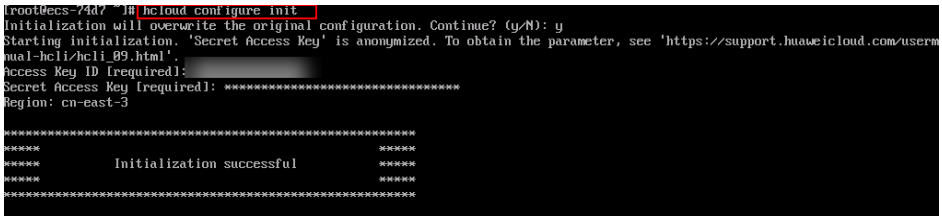
4. Initialize KooCLI.

   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 19-3** is displayed, the initialization is successful.

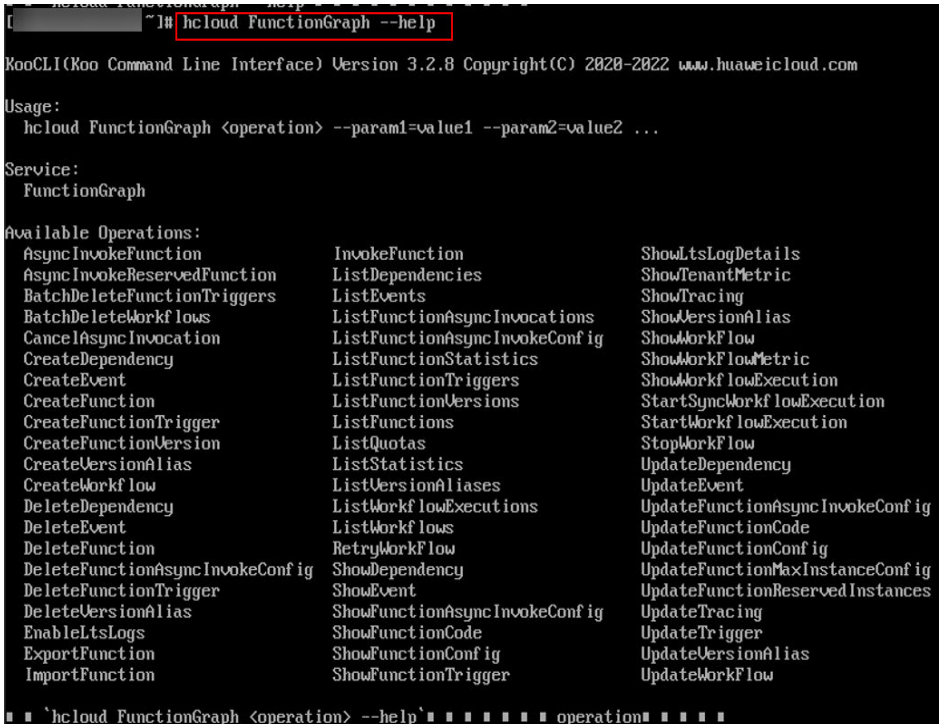
**Figure 19-3** Initialization successful




5. Run the following command to view the commands supported by FunctionGraph. As shown in **Figure 19-4**, **Available Operations** lists the operations supported by FunctionGraph.

   ```
   hcloud FunctionGraph --help
   ```

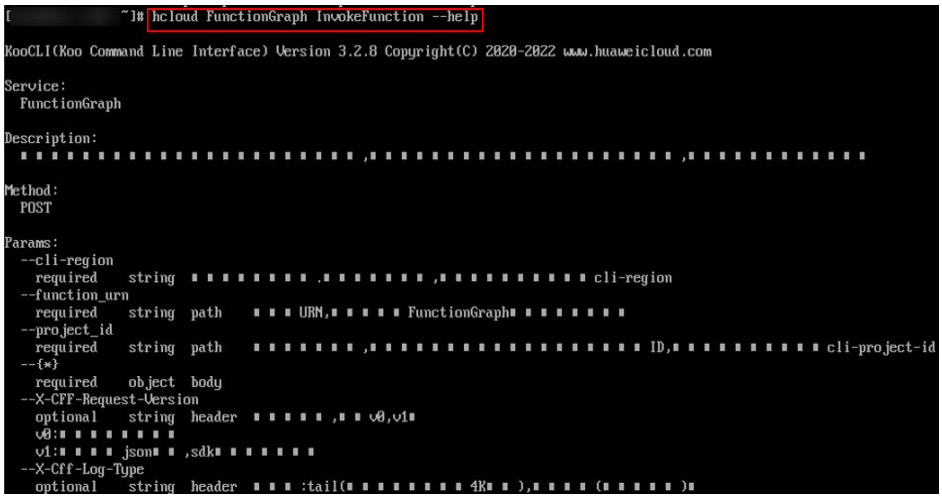
**Figure 19-4** Operations supported by FunctionGraph




Run the following command to obtain help information about operation **InvokeFunction**. If the command is successfully executed, the information shown in **Figure 19-5** is displayed.

```
hcloud FunctionGraph InvokeFunction --help
```

**Figure 19-5** Help information about operation InvokeFunction




# 19.3 Invoking a Function

Before invoking a function, obtain the URN, as shown in **Figure 19-6**.

**Figure 19-6** Obtaining a function URN


## Synchronous Invocation

The following is an example command for synchronous invocation. For details about the parameters, see **Table 19-2**.

```
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 19-2** Parameter description

| 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 19-7** shows the output result. For details about the response parameters, see **Table 19-3**.

**Figure 19-7** Output result

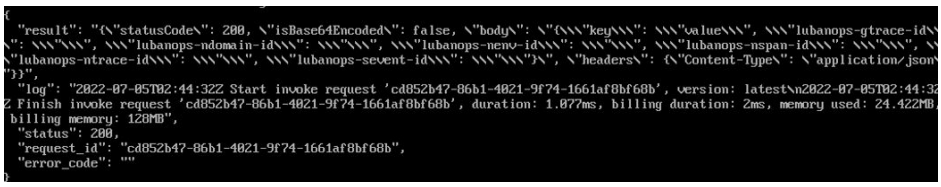


**Table 19-3** Response parameters

| Parameter | Type | Description |
| --- | --- | --- |
| request_id | String | Request ID. |
| result | String | Execution result. |
| log | String | 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 19-4**.

```
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 19-4** Parameter description

| 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 19-8** shows the output result. For details about the response parameters, see **Table 19-3**.

**Figure 19-8** Output result

```
{
  "request_id": "3977431d-c254-4e65-979e-30541936651b"
}
```

**Table 19-5** Response parameters

| Parameter | Type | Description |
| --- | --- | --- |
| request_id | String | Request ID. |

# 20 Audit

## 20.1 Operations Logged by CTS

Table 20-1 lists the FunctionGraph operations that can be logged by CTS.

**Table 20-1** 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 |

# 20.2 Viewing CTS Traces in the Trace List

## Scenarios

After you enable CTS and the management tracker is created, CTS starts recording operations on cloud resources. After a data tracker is created, the system starts recording operations on data in Object Storage Service (OBS) buckets. Cloud Trace Service (CTS) stores operation records (traces) generated in the last seven days.

This section describes how to query or export operation records of the last seven days on the CTS console.

- **Viewing Real-Time Traces in the Trace List of the New Edition**
- **Viewing Real-Time Traces in the Trace List of the Old Edition**

## Constraints

- Traces of a single account can be viewed on the CTS console. Multi-account traces can be viewed only 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.
- You can only query operation records of the last seven days on the CTS console. To store operation records for longer than seven days, you must configure transfer to OBS or Log Tank Service (LTS) so that you can view them in OBS buckets or LTS log groups.
- After performing operations on the cloud, you can query management traces on the CTS console one minute later and query data traces five 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.
- These operation records are retained for seven days on the CTS console and are automatically deleted upon expiration. Manual deletion is not supported.

## Viewing Real-Time Traces in the Trace List of the New Edition

1. Log in to the management console.
2. Click  in the upper left corner and choose **Management & GovernanceManagement & Deployment** > **Cloud Trace Service**. The CTS console is displayed.
3. Choose **Trace List** in the navigation pane on the left.
4. On the **Trace List** page, use advanced search to query traces. You can combine one or more filters.
   - **Trace Name**: Enter a trace name.
   - **Trace ID**: Enter a trace ID.
   - **Resource Name**: Enter a resource name. 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.

- **Resource ID**: Enter a resource ID. Leave this field empty if the resource has no resource ID or if resource creation failed.
- **Trace Source**: Select a cloud service name from the drop-down list.
- **Resource Type**: Select a resource type from the drop-down list.
- **Operator**: Select one or more operators from the drop-down list.
- **Trace Status**: Select **normal**, **warning**, or **incident**.
  - **normal**: The operation succeeded.
  - **warning**: The operation failed.
  - **incident**: The operation caused a fault that is more serious than the operation failure, for example, causing other faults.
- **Enterprise Project ID**: Enter an enterprise project ID.
- **Access Key**: Enter a temporary or permanent access key ID.
- Time range: Select **Last 1 hour**, **Last 1 day**, or **Last 1 week**, or specify a custom time range within the last seven days.

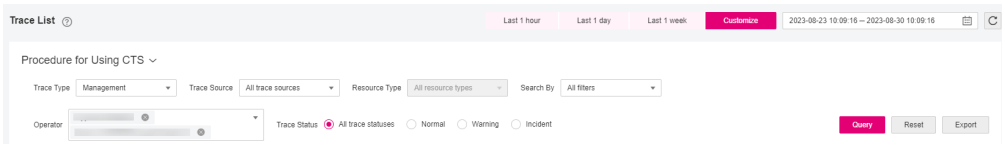
5. On the **Trace List** page, you can also export and refresh the trace list, and customize columns to display.
   - Enter any keyword in the search box and press **Enter** to filter desired traces.
   - Click **Export** to export all traces in the query result as an .xlsx file. The file can contain 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.
6. For details about key fields in the trace structure, see **Trace Structure**section "Trace References" > "Trace Structure" and **Example Traces**section "Trace References" > "Example Traces".
7. (Optional) On the **Trace List** page of the new edition, click **Go to Old Edition** in the upper right corner to switch to the **Trace List** page of the old edition.

## Viewing Real-Time Traces in the Trace List of the Old Edition

1. Log in to the management console.
2. Click in the upper left corner and choose **Management & GovernanceManagement & Deployment** > **Cloud Trace Service**. The CTS console is displayed.
3. Choose **Trace List** in the navigation pane on the left.
4. Each time you log in to the CTS console, the new edition is displayed by default. Click **Go to Old Edition** in the upper right corner to switch to the trace list of the old edition.
5. Set filters to search for your desired traces, as shown in **Figure 20-1**. The following filters are available.

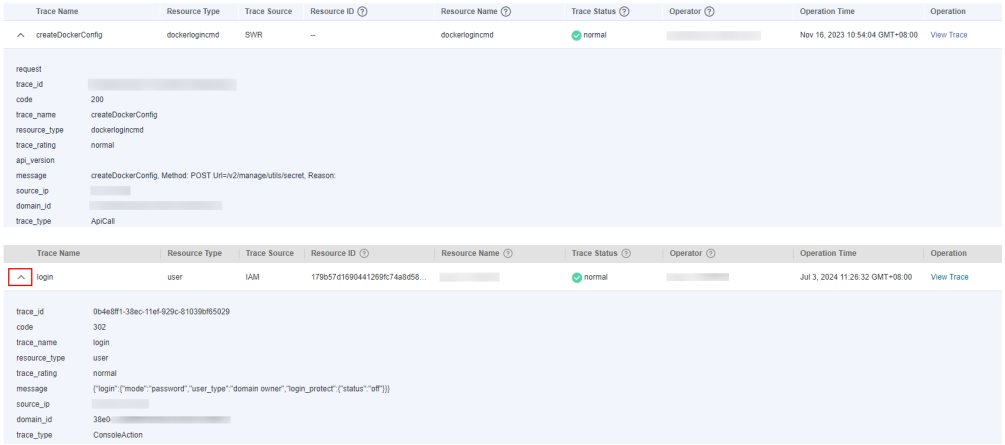
  

**Figure 20-1** Filters



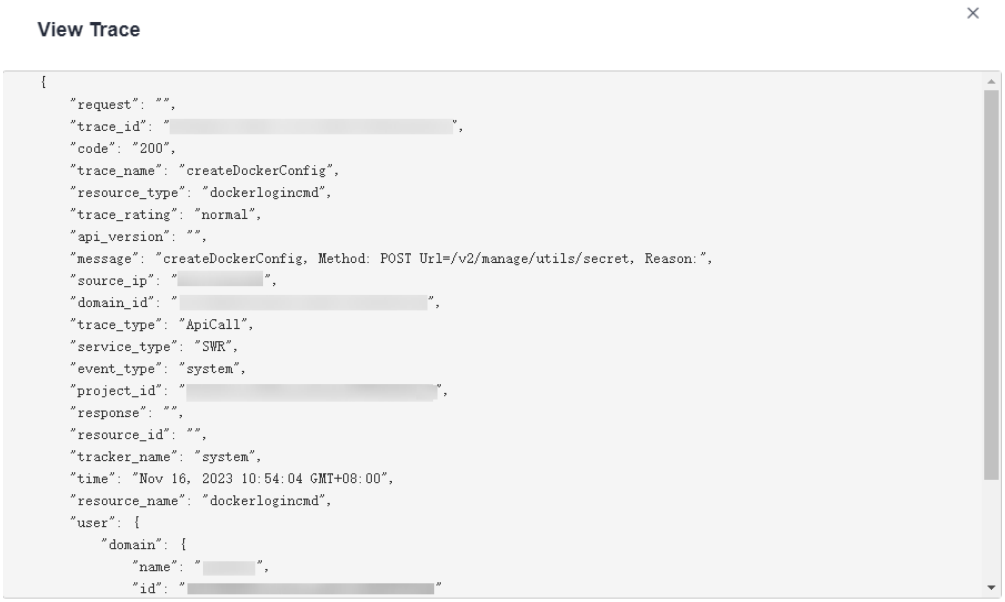

- **Trace Type**, **Trace Source**, **Resource Type**, and **Search By**: Select a filter from the drop-down list.
  - If you select **Resource ID** for **Search By**, specify a resource ID.
  - If you select **Trace name** for **Search By**, specify a trace name.
  - If you select **Resource name** for **Search By**, specify a resource name.
- **Operator**: Select a user.
- **Trace Status**: Select **All trace statuses**, **Normal**, **Warning**, or **Incident**.
- Time range: Select **Last 1 hour**, **Last 1 day**, or **Last 1 week**, or specify a custom time range within the last seven days.

6. Click **Query**.
7. 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.
8. Click on the left of a trace to expand its details.




9. Click **View Trace** in the **Operation** column. The trace details are displayed.




10. For details about key fields in the trace structure, see **Trace Structure**section "Trace References" > "Trace Structure" and **Example Traces**section "Trace References" > "Example Traces" in the *CTS User Guide*.
11. (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.