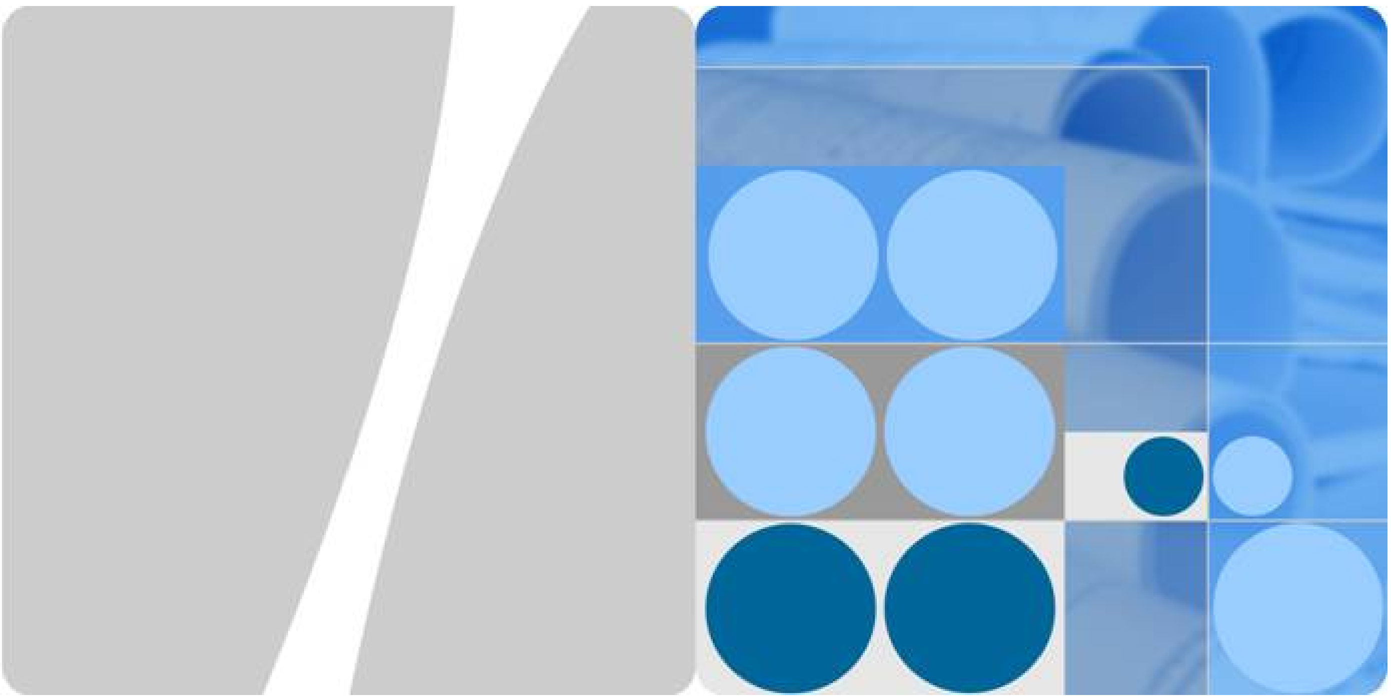


FunctionGraph

User Guide

Date 2023-07-06

Contents

1 Service Overview.......... 1
1.1 What Is FunctionGraph?.......... 1
1.2 Product Features.......... 3
1.3 Product Advantages.......... 5
1.4 Application Scenarios.......... 6
1.5 Function Types.......... 7
1.5.1 Event Functions.......... 7
1.5.2 HTTP Functions.......... 8
1.6 Notes and Constraints.......... 9
1.7 Permissions Management.......... 10
1.8 Concepts.......... 12
1.9 Relationships Between FunctionGraph and Other Services.......... 14
2 Getting Started.......... 15
2.1 Introduction.......... 15
2.2 Creating a Function from Scratch.......... 16
2.3 Creating a Function Using a Template.......... 19
2.4 Deploying a Function Using a Container Image.......... 21
2.4.1 Developing an HTTP Function.......... 21
3 Before You Start.......... 26
3.1 Use of FunctionGraph.......... 26
3.2 Permissions Management.......... 28
3.2.1 Creating a User and Granting Permissions.......... 28
3.2.2 Creating a Custom Policy.......... 30
3.3 Supported Programming Languages.......... 31
3.3.1 Node.js.......... 31
3.3.2 Python.......... 32
3.3.3 Java.......... 32
3.3.4 Go.......... 32
3.3.5 Custom Runtime.......... 32
4 Building a FunctionGraph Function.......... 40
4.1 Creating a Deployment Package.......... 40
4.2 Creating a Function from Scratch.......... 45

4.2.1 Creating an Event Function....45
4.2.2 Creating an HTTP Function....48
4.3 Creating a Function Using a Template....52
4.4 Deploying a Function Using a Container Image....53
5 Configuring the Function....57
5.1 Configuring Initialization....57
5.2 Configuring Basic Settings....58
5.3 Configuring Agency Permissions....60
5.4 Configuring the Network....63
5.5 Configuring Disk Mounting....65
5.6 Configuring Environment Variables....70
5.7 Configuring Asynchronous Execution Notification....73
5.8 Configuring Single-Instance Multi-Concurrency....75
5.9 Managing Versions....78
5.10 Managing Aliases....80
5.11 Configuring Dynamic Memory....82
6 Online Debugging....84
7 Creating Triggers....88
7.1 Managing Triggers....88
7.2 Using a Timer Trigger....88
7.3 Using an APIG (Dedicated) Trigger....90
7.4 Using an OBS Trigger....93
7.5 Using a Kafka Trigger....94
7.6 Using an LTS Trigger....96
7.7 Using a CTS Trigger....97
7.8 Appendix: Cron Expressions for a Function Timer Trigger....99
8 Invoking the Function....103
8.1 Synchronous Invocation....103
8.2 Asynchronous Invocation....103
8.3 Retry Mechanism....105
9 Monitoring....106
9.1 Metrics....106
9.1.1 Function Monitoring....106
9.1.2 Function Metrics....107
9.1.3 Creating an Alarm Rule....109
9.2 Logs....111
9.2.1 Querying Function Logs....111
9.2.2 Managing Function Logs....112
10 Function Management....114
11 Dependency Management....116

12 Reserved Instance Management.......... 123

13 Audit.......... 130

13.1 Operations Logged by CTS.......... 130

13.2 Viewing Audit Logs.......... 131

14 FAQs.......... 132

14.1 General FAQs.......... 132

14.1.1 What Is FunctionGraph?.......... 132

14.1.2 Do I Need to Apply for Any Compute, Storage, or Network Services When Using FunctionGraph? 132

14.1.3 Do I Need to Deploy My Code After Programming?.......... 132

14.1.4 What Runtimes Does FunctionGraph Support?.......... 132

14.1.5 How Much Disk Space Is Allocated to Each FunctionGraph Function?.......... 133

14.1.6 Does FunctionGraph Support Function Versioning?.......... 133

14.1.7 How Does a Function Read or Write Files?.......... 133

14.1.8 Does FunctionGraph Support Function Extension?.......... 133

14.1.9 Which Permissions Are Required for an IAM User to Use FunctionGraph?.......... 134

14.1.10 How Can I Create an ODBC Drive-based Python Dependency Package for Database Query?.......... 134

14.1.11 What Is the Quota of FunctionGraph?.......... 134

14.1.12 How Does a Container Image–based Function Resolve a Private DNS Domain Name?.......... 134

14.1.13 How Do I Use a Domain Name to Access an API Registered with API Gateway (Dedicated)?.......... 135

14.1.14 What Are the Common Application Scenarios of FunctionGraph?.......... 135

14.1.15 Why Can't the API Gateway Domain Name Bound to a Service Be Resolved During Function Invocation?.......... 136

14.1.16 Does FunctionGraph Support Synchronous Transmission at the Maximum Intranet Bandwidth? 136

14.1.17 What If the VPC Quota Is Used Up?.......... 136

14.1.18 How Can I Print Info, Error, or Warn Logs?.......... 136

14.1.19 Can I Set the Domain Name of an API to My Own Domain Name?.......... 136

14.1.20 Can I Change the Runtime?.......... 136

14.1.21 Can I Change a Function's Name?.......... 136

14.1.22 Why Is Message "failed to mount exist system path" Displayed?.......... 136

14.1.23 How Do I Obtain Uploaded Files?.......... 137

14.1.24 Why Can't I Receive Responses for Synchronous Invocation?.......... 137

14.1.25 What Should I Do If the **os.system("command &")** Execution Logs Are Not Collected?.......... 137

14.1.26 Which Directories Can Be Accessed When a Custom Runtime Is Used?.......... 137

14.1.27 Which Minor Versions of Python 3.6 and 3.9 Are Supported?.......... 137

14.1.28 Which Actions Can Be Used Instead of a VPC Administrator Agency for VPC Access?.......... 137

14.1.29 What Are the Possible Causes for Function Timeout?.......... 138

14.1.30 How Do I Obtain the Code of a Function?.......... 138

14.1.31 Do You Have Sample Code for Initializers?.......... 138

14.1.32 How Do I Enable Structured Log Query?.......... 139

14.2 Function Creation FAQs.......... 142

14.2.1 Can I Add Threads and Processes in Function Code?......142
14.2.2 What Are the Rules for Packaging a Function Project?......142
14.2.3 How Does FunctionGraph Isolate Code?......146
14.2.4 How Do I Create the Bootstrap File for an HTTP Function?......146
14.3 Trigger Management FAQs......147
14.3.1 What If Error Code 500 Is Reported When Functions that Use APIG Triggers Return Strings?......147
14.3.2 What Do LATEST and TRIM_HORIZON Mean in DIS Trigger Configuration?......148
14.3.3 Why Can't I Enable or Disable OBS Triggers by Calling APIs?......148
14.3.4 How Do I Use an APIG Trigger to Invoke a Function?......148
14.3.5 How Does a Function Obtain the Request Path or Parameters When Using an APIG Trigger?......148
14.3.6 Can I Create an OBS Trigger with an Existing Bucket?......149
14.4 Dependency Management FAQs......149
14.4.1 What Is a Dependency?......149
14.4.2 When Do I Need a Dependency?......149
14.4.3 What Are the Precautions for Using a Dependency?......149
14.4.4 What Dependencies Does FunctionGraph Support?......149
14.4.5 Does FunctionGraph Support Class Libraries?......151
14.4.6 How Do I Use Third-Party Dependencies on FunctionGraph?......151
14.4.7 How Do I Create Function Dependencies?......151
14.4.8 How Do I Create a Dependency on the FunctionGraph Console?......153
14.4.9 How Do I Add a Dependency to a Function?......153
14.5 Function Execution FAQs......153
14.5.1 How Long Does It Take to Execute a FunctionGraph Function?......153
14.5.2 Which Steps Are Included in Function Execution?......154
14.5.3 How Does FunctionGraph Process Concurrent Requests?......154
14.5.4 What If Function Instances Have Not Been Executed for a Long Time?......154
14.5.5 How Can I Speed Up Initial Access to a Function?......154
14.5.6 How Do I Know the Actual Memory Used for Function Execution?......154
14.5.7 Why Is My First Request Slow?......154
14.5.8 What Do I Do If an Error Occurs When Calling an API?......155
14.5.9 How Do I Read the Request Header of a Function?......155
14.5.10 Why Does a Function Use More Memory Than Estimated and Even Trigger the Out of Memory Alarm?......155
14.5.11 How Do I Check the Memory Usage When Seeing "runtime memory limit exceeded"?......155
14.5.12 How Do I Troubleshoot "CrashLoopBackOff"?......156
14.5.13 After I Updated an Image with the Same Name, Reserved Instances Still Use the Old Image. What Can I Do?......156
14.6 Function Configuration FAQs......156
14.6.1 Can I Set Environment Variables When Creating Functions?......156
14.6.2 Can I Enter Sensitive Information in Environment Variables?......156
14.7 External Resource Access FAQs......156
14.7.1 How Does a Function Access the MySQL Database?......156
14.7.2 How Does a Function Access Redis?......157

14.7.3 How Do I Configure External Network Access?........ 158
14.8 Other FAQs........ 158
14.8.1 How Do I View the Alarm Rules Configured for a Function?........ 158
14.8.2 Does FunctionGraph Support ZIP Decompiling During Video Transcoding?........ 158

15 Change History........ 159

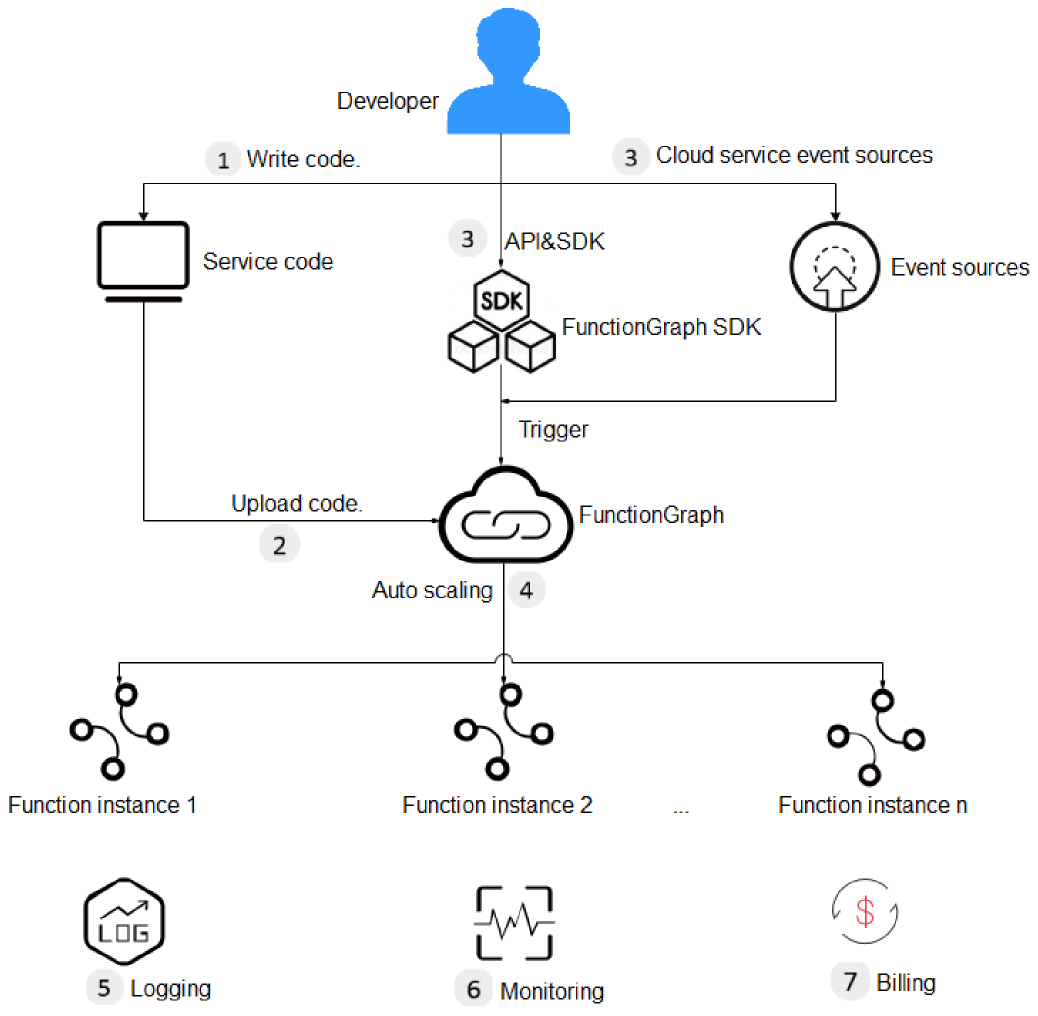
1 Service Overview

1.1 What Is FunctionGraph?

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

Figure 1-1 shows the process of using FunctionGraph.

Figure 1-1 Usage process



1. Write code.

Write code in Node.js, Python, Java, or Go.

2. Upload code.

Currently, you can edit code inline, upload a ZIP or JAR file, or obtain a ZIP file from OBS. For details, see **Table 1-2**.

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

Call RESTful APIs or use cloud service event sources to trigger function execution and generate instances to implement service functions.

4. Auto scaling is implemented.

During function execution, FunctionGraph scales automatically based on the number of requests without the need for configurations. For details about the maximum number of function instances that can be run concurrently, see **Notes and Constraints**.

5. View logs.

View run logs of functions as FunctionGraph is interconnected with Log Tank Service (LTS).

6. View monitoring information.

View graphical monitoring information as FunctionGraph is interconnected with Cloud Eye.

1.2 Product Features

Function Management

FunctionGraph provides console-based function management.

- The Node.js, Java, Python, Go, and custom runtimes are supported. **Table 1-1** provides the details.

  NOTE

  You are advised to use the latest runtime version.

  Table 1-1 Runtimes

Runtime	Supported Version
Node.js	6.10, 8.10, 10.16, 12.13, and 14.18
Python	2.7, 3.6, and 3.9
Java	8.0 and 11
Go	1.x
Custom	-

- Multiple code entry modes

  FunctionGraph allows you to edit code inline, upload a ZIP file from Object Storage Service (OBS), or directly upload a ZIP or JAR file. **Table 1-2** lists the code entry modes supported for each runtime.

  Table 1-2 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
Python	Supported	Supported	Not supported	Supported
Java	Not supported	Supported	Supported	Supported

Runtime	Editing Code Inline	Uploading a ZIP File	Uploading a JAR File	Uploading a ZIP File from OBS
Go	Not supported	Supported	Not supported	Supported
Custom	Supported	Supported	Not supported	Supported

Trigger

Table 1-3 lists the invocation modes for different trigger types.

Table 1-3 Function invocation modes

Trigger	Function Invocation Mode
APIG trigger	Synchronous invocation
OBS trigger	Asynchronous invocation
Timer trigger	Asynchronous invocation
Log Tank Service (LTS) trigger	Asynchronous invocation
Cloud Trace Service (CTS) trigger	Asynchronous invocation
Kafka trigger	Asynchronous invocation

Logs and Metrics

FunctionGraph graphically displays function monitoring metrics and collects function running logs, enabling you to view function statuses, and locate problems by querying logs.

To query logs, see **Managing Function Logs**.

For details about monitoring metric, see **Function Monitoring**.

For details about tenant-level function monitoring metrics, see **Function Monitoring**.

Function Initialization

The initializer interface is introduced to:

- Isolate function initialization and request processing to enable clearer program logic and better structured and higher-performance code.
- Ensure smooth function upgrade to prevent performance loss during the application layer's cold start initialization. Enable new function instances to automatically execute initialization logic before processing requests.

- Identify the overhead of application layer initialization, and accurately determine the time for resource scaling and the quantity of required resources. This feature makes request latency more stable when the application load increases and more function instances are required.

HTTP Functions

You can set **Function Type** to **HTTP Function** on the function creation page. HTTP functions are designed to optimize web services. You can send HTTP requests to URLs to trigger function execution. HTTP functions support APIG and API Connect (APIC) triggers only.

Custom Images

You can directly package and upload container images. The images are loaded and started by the platform and can be called in a similar way as HTTP functions. Unlike the previous code upload mode, you can use a custom code package, which is flexible and reduces migration costs.

1.3 Product Advantages

No Servers to Manage

FunctionGraph automatically runs your code and frees you from provisioning and managing servers, allowing you to focus on business innovation.

Auto Scaling

FunctionGraph automatically scales to suit fluctuations in resource demands and ensures that the service remains accessible even during peaks and spikes.

It automatically scales in/out resources based on the number of service requests, and distributes requests to function instances through automatic load balancing.

In addition, the system preheats instances for the traffic loads to reduce the impact of cold start on your services.

Event-based Triggering

FunctionGraph integrates with multiple cloud services using an event-based triggering mechanism to meet service requirements.

It is interconnected with the LTS and Cloud Eye services, allowing you to view function logs and metrics without the need for any configurations.

High Availability

If an instance becomes faulty, FunctionGraph starts another instance to process new requests and releases resources from the unhealthy instance.

Dynamic Resource Adjustment

Resource specifications can be dynamically adjusted to minimize resource usage and reduce costs.

1.4 Application Scenarios

FunctionGraph is suitable for various scenarios, such as real-time file processing, real-time data stream processing, web & mobile application backends, and AI application.

Scenario 1: Event-Driven Applications

Services are executed in event-driven mode and resources are provisioned based on demands. Developers do not need to be concerned about service peaks or troughs. Idle resources are not billed, reducing O&M costs. Event-driven applications include file processing, image processing, live streaming/transcoding, real-time data stream processing, and IoT rule/event processing.

- **Real-time file processing**

  When files are uploaded from a client to OBS, functions can be triggered to create image thumbnails in real time, convert video formats, aggregate and filter data files, or implement other file operations.

  Advantages:

  - FunctionGraph automatically allocates resources to run more function instances as the number of received requests increases.
  - Files are uploaded to OBS to trigger file processing functions.
  - You will be billed only for resources used to process files as needed (you are not billed for idle resources during lows in demand).

- **Real-time data stream processing**

  FunctionGraph works with DIS to process data streams in real time. FunctionGraph supports application activity tracking, sequential transaction processing, data stream analysis, data sorting, metric generation, log filtering, indexing, social media analysis, and IoT device data telemetry and metering.

  Advantages:

  - Data is collected by means of DIS streams to trigger data processing functions.
  - FunctionGraph automatically allocates resources to run more function instances as the number of received requests increases.
  - You will be billed only for resources used to process files as needed (you are not billed for idle resources during lows in demand).

Scenario 2: Web Applications

Interconnect FunctionGraph with other cloud services or your VMs to quickly build highly available and scalable web & mobile backends. Web applications include mini programs, web pages/apps, chatbots, and Backends for Frontends (BFF).

Advantages:

- FunctionGraph ensures high reliability of website data using OBS and CloudTable, and high-availability of website logic using API Gateway.
- FunctionGraph automatically allocates resources to run more function instances as the number of received requests increases.
- You will be billed only for resources used to process files as needed (you are not billed for idle resources during lows in demand).

Scenario 3: AI Applications

Intelligence evolution requires various services to be integrated for quick rollout. These services include third-party service integration, AI inference, and license plate recognition.

Advantages:

- FunctionGraph works with EI services for text recognition and content moderation to suit a wide range of scenarios – make adjustments whenever you need as demands change.
- You only need to apply for related services and write service code without having to provision or manage servers.
- You will be billed only for function execution and used EI services without having to pay for idle resources when service demands are low.

1.5 Function Types

1.5.1 Event Functions

Overview

FunctionGraph supports event functions. An event can trigger function execution. Generally, it is in JSON format. You can create an event to trigger your function through the cloud service platform or CloudIDE. All types of triggers supported by FunctionGraph can trigger event functions.

NOTE

1. On the function creation page, **Function Type** is set to **Event Function** by default.
2. During testing, a function can be triggered by simply entering the specified event in JSON format.
3. You can also use triggers to trigger event functions.

Advantages

- Easy single-node programming

  You can edit event functions on FunctionGraph or CloudIDE or upload code packages there and deploy them with just a few clicks. There is no need for you to care about function concurrency or fault rectification.
- High-performance, high-speed runtimes

  Event functions can be started, scaled, and called within milliseconds. Faults can be detected and rectified within seconds.

- Complete tool chain

  FunctionGraph provides comprehensive logging, tracing, debugging, and monitoring, allowing developers to roll out functions in just three steps.

Restrictions

Event functions face event source restrictions. You need to comply with the function development rules of the function platform.

1.5.2 HTTP Functions

Overview

FunctionGraph supports event functions and HTTP functions. 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

1. HTTP functions support the HTTP/1.1 protocol.
2. On the function creation page, **HTTP Function** is newly added.
3. The HTTP function must be set to **bootstrap**. You can directly write the startup command and **allow access over port 8000**.

Advantages

- Support for multiple frameworks

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

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

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

Restrictions

- HTTP functions support APIG (shared), APIG (dedicated), and APIC triggers only.
- Multiple API triggers can be bound to the same function, but all the APIs must belong to the same APIG service.
- For HTTP functions, the size of the HTTP response body cannot exceed 6 MB.
- HTTP functions cannot be executed for a long time, invoked asynchronously, or retried.

1.6 Notes and Constraints

Account Resource Constraints

Table 1-4 Account resource constraints

Resource	Limit
Maximum number of functions that can be created under an account	400
Maximum number of versions allowed for a function	10
Maximum number of aliases allowed for a function	10
Size of a code deployment package (in ZIP or JAR format) that can be uploaded to the FunctionGraph console	40 MB
Size of a code deployment package (in ZIP or JAR format) that can be edited inline during function API invocation	50 MB
Size of an original code deployment package allowed during function API invocation	● ZIP: 1500 MB (after decompression) ● OBS bucket: 300 MB (after compression)
Maximum size of deployment packages allowed for an account	10 GB
Number of concurrent executions per account	100
Maximum number of reserved instances that an account can create	90 (Number of concurrent executions per account x 90%)
Size of all environment variables of a function	4096 characters

Function Running Resource Constraints

Table 1-5 Function running resource constraints

Resource	Default
Ephemeral disk space (**/tmp** space)	512 MB
Number of file descriptors	1024

Resource	Default
Total number of processes and threads	1024
Maximum execution duration per request	259,200s
Valid payload size of invocation request body (synchronous invocation)	6 MB
Valid payload size of invocation response body (synchronous invocation)	6 MB
Valid payload size of invocation request body (asynchronous invocation)	256 KB
Size of imported resources	≤ 50 MB ZIP file
Image size per function	10 GB
Size of exported resources	≤ 50 MB
Instances per tenant	1000

NOTE

- Valid payload size of invocation response body (synchronous invocation): The returned character string or the JSON character string of the serialized response body is less than or equal to 6 MB by default. The actual data size varies depending on the backend settings of FunctionGraph. The backend determines the size of the serialized data with a byte-level deviation. The actual valid payload size is 6 MB ± 100 bytes.
- You are not advised to invoke a function whose execution time exceeds 90s on the FunctionGraph console. To invoke such a function, use asynchronous invocation.

1.7 Permissions Management

If you need to assign different permissions to employees in your enterprise to access your FunctionGraph resources, IAM is a good choice for fine-grained permissions management. IAM provides identity authentication, permissions management, and access control, helping you secure access to your cloud resources.

With IAM, you can use your account to create IAM users for your employees, and assign permissions to the users to control their access to specific resource types. For example, some software developers in your enterprise need to use FunctionGraph resources but must not delete them or perform any high-risk operations. To achieve this result, you can create IAM users for the software developers and grant them only the permissions required for using FunctionGraph resources.

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

FunctionGraph Permissions

By default, new IAM users do not have any permissions assigned. You need to add a user to one or more groups, and assign permissions policies to these groups. The user then inherits permissions from the groups it is a member of. This process is called authorization. After authorization, the user can perform specified operations on FunctionGraph based on the permissions.

FunctionGraph is a project-level service deployed and accessed in specific physical regions. To assign FunctionGraph permissions to a user group, specify the scope as region-specific projects and select projects in relevant regions for the permissions to take effect. If **All projects** is selected, the permissions will take effect for the user group in all region-specific projects. When accessing FunctionGraph, the users need to switch to a region where they have been authorized to use the FunctionGraph service.

You can grant users permissions by using roles and policies.

- Roles: A type of coarse-grained authorization mechanism that defines permissions related to user responsibilities. This mechanism provides only a limited number of service-level roles for authorization. When using roles to grant permissions, you may also need to assign other roles on which the permissions depend. However, roles are not an ideal choice for fine-grained authorization and secure access control.
- Policies: A type of fine-grained authorization mechanism that defines permissions required to perform operations on specific cloud resources under certain conditions. This mechanism allows for more flexible policy-based authorization, meeting requirements for secure access control.

Table 1-6 lists all the system policies supported by FunctionGraph.

Table 1-6 Permissions description

Role/Policy Name	Description	Category	Dependency
FunctionGraph FullAccess	This policy grants all permissions for FunctionGraph.	System-defined policy	N/A
FunctionGraph ReadOnlyAccess	This policy grants read-only permissions for FunctionGraph.	System-defined policy	N/A
FunctionGraph CommonOperations	This policy grants permissions to query functions and triggers, and invoke functions.	System-defined policy	N/A

Table 1-7 lists the common operations supported by each system-defined policy of FunctionGraph. Please choose proper system-defined policies according to this table.

Table 1-7 Common operations supported by each system-defined policy

Operation	FunctionGraph ReadOnlyAccess	FunctionGraph CommonOperations	FunctionGraph FullAccess
Creating functions	×	×	√
Querying functions	√	√	√
Modifying functions	×	×	√
Deleting functions	×	×	√
Invoking functions	×	√	√
Querying function logs	√	√	√
Viewing function metrics	√	√	√

1.8 Concepts

Function

Functions are code defined to handle events.

Event Source

An event source is a public cloud service or custom application that publishes events.

Synchronous Invocation

Clients wait for explicit responses to their requests from a function. Responses are returned only after the function is invoked.

Asynchronous Invocation

Clients do not care about the function invocation results of their requests. After receiving a request, FunctionGraph puts it in a queue, returns a response, and processes other requests when there are idle resources.

Trigger

A trigger is an event that triggers function execution.

Single-Instance Multi-Concurrency

The number of requests that can be concurrently processed by an instance.

Custom Images

You can directly package and upload container images. The platform then loads and starts these images to create functions.

Custom Function Execution

You can customize scripts and files to execute functions.

Function Logs

Logs generated during function invocation.

Function Monitoring

Monitoring information generated during function execution.

Function Version

FunctionGraph allows you to publish one or more versions throughout the development, testing, 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, modify settings when necessary.

Function Alias

You can create an alias for a specific function version. To roll back to a previous version, use the corresponding alias to represent the version instead of modifying the function code.

Each function alias can be bound to a major version and an additional version for traffic shifting.

Dependency Package

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.

Bootstrap File

The **bootstrap** file is the startup file of an 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.

1.9 Relationships Between FunctionGraph and Other Services

Table 1-8 describes the cloud services that have been interconnected with FunctionGraph.

Table 1-8 Interconnected services

Service	Function
SMN	FunctionGraph functions are constructed to process SMN notifications.
OBS	FunctionGraph functions are created to process OBS bucket events, such as object creation or deletion events. For example, when an image is uploaded to the specified bucket, OBS invokes the function to read the image and create a thumbnail.
Cloud Eye	FunctionGraph is interconnected with Cloud Eye to report monitoring metrics, allowing you to view function metrics and alarm messages through Cloud Eye.
VPC	Functions can be configured to access resources in Virtual Private Clouds (VPCs) or to access the Internet through source network address translation (SNAT) by binding elastic IP addresses.

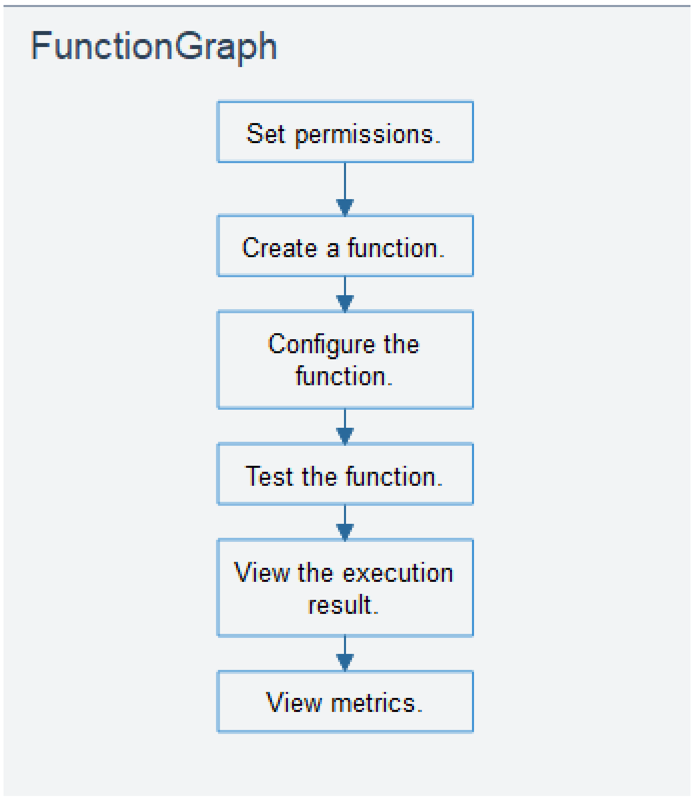
2 Getting Started

2.1 Introduction

General Procedure

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.

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



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

2.2 Creating a Function from Scratch

Introduction

This section describes how to quickly create and test a HelloWorld function on the FunctionGraph console.

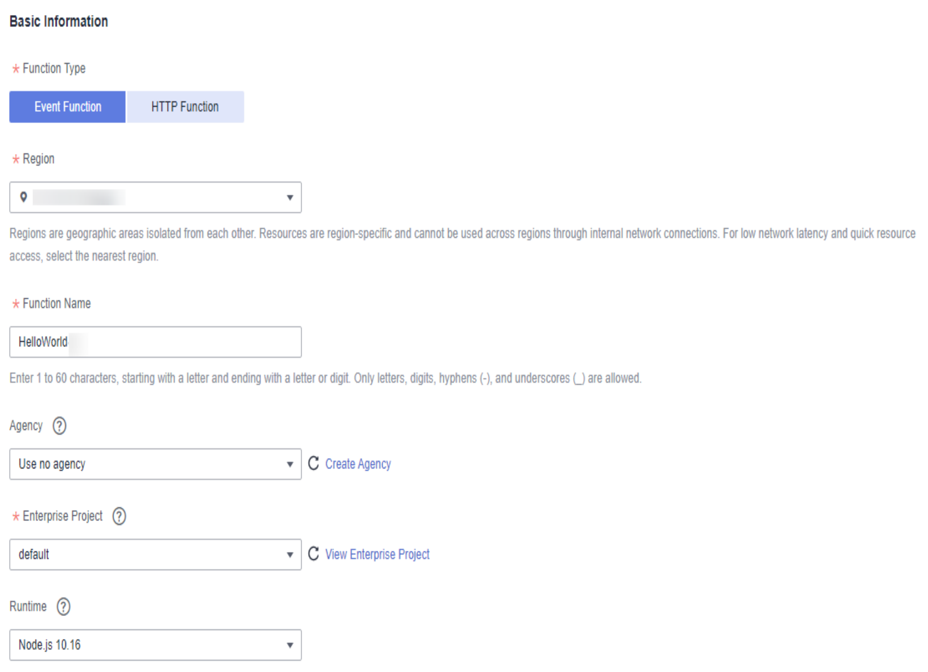
Step 1: Prepare the Environment

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

Step 2: Create a Function

1. Log in to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click **Create Function** in the upper right corner and choose **Create from scratch**.
3. On the displayed page, set **Function Name** to **HelloWorld**, retain the default values for other parameters, and click **Create Function**. For details, see **Figure 2-1**.

Figure 2-1 Configuring basic information


Basic Information

* Function Type

Event Function | HTTP Function

* Region

Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

* Function Name

HelloWorld

Enter 1 to 60 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.

Agency

Use no agency | Create Agency

* Enterprise Project

default | View Enterprise Project

Runtime

Node.js 10.16

4. Configure the code source, copy the following code to the code window, and click **Deploy**.

   The sample code enables you to obtain test events and print test event information.

```
exports.handler = function (event, context, callback) {
    const error = null;
    const output = `Hello message: ${JSON.stringify(event)}`;
    callback(error, output);
}
```

Step 3: Test the Function

1. On the function details page, click **Test**. In the displayed dialog box, create a test event.
2. Select **blank-template**, set **Event Name** to **test**, modify the test event as follows, and click **Create**.

```
{
    "hello": "function"
}
```

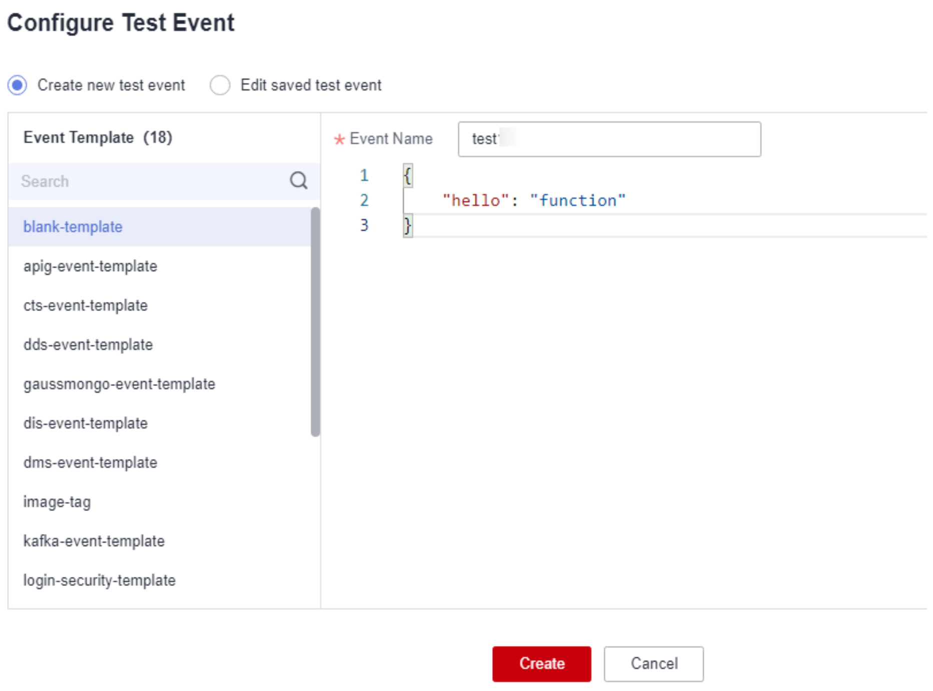


Figure 2-2 Configuring a test event

Configure Test Event

[✓ Create new test event] Edit saved test event

Event Template (18)

Search

- blank-template
- apig-event-template
- cts-event-template
- dds-event-template
- gaussmongo-event-template
- dis-event-template
- dms-event-template
- image-tag
- kafka-event-template
- login-security-template

* Event Name: test

```
{
    "hello": "function"
}
```

Step 4: View the Execution Result

Click **Test** and view the execution result on the right.

- **Function Output**: displays the return result of the function.
- **Log Output**: displays the execution logs of the function.
- **Summary**: displays key information of the logs.

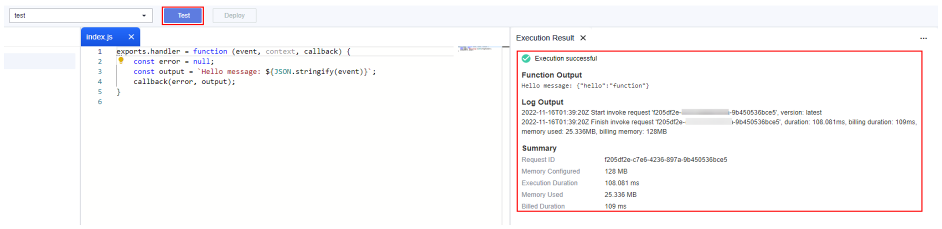


Figure 2-3 Viewing the execution result

```
exports.handler = function (event, context, callback) {
    const error = null;
    const output = `Hello message: ${JSON.stringify(event)}`;
    callback(error, output);
}
```

Execution successful

Function Output

```
Hello message: {"hello":"function"}
```

Log Output

```
2022-11-16T01:39:20Z Start invoke request 'f205df2e-[redacted]-9b450536bce5', version: latest
2022-11-16T01:39:20Z Finish invoke request 'f205df2e-[redacted]-9b450536bce5', duration: 108.081ms, billing duration: 109ms,
memory used: 25.336MB, billing memory: 128MB
```

Summary

Request ID	f205df2e-c7e6-4236-897a-9b450536bce5
Memory Configured	128 MB
Execution Duration	108.081 ms
Memory Used	25.336 MB
Billed Duration	109 ms

NOTE

A maximum of 2 KB logs can be displayed. For more log information, see **Querying Function Logs**.

Step 5: View Monitoring Metrics

On the function details page, click the **Monitoring** tab.

- On the **Monitoring** tab page, choose **Metrics**, and select a time range (such as 5 minutes, 15 minutes, or 1 hour) to query the function.
- The following metrics are displayed: invocations, errors, duration (including the maximum, average, and minimum durations), and throttles.

Step 6: Delete a Function

1. On the function details page, choose **Operation** > **Delete function** in the upper right corner.
2. In the displayed dialog box, click **OK** to release resources.

2.3 Creating a Function Using a Template

Introduction

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

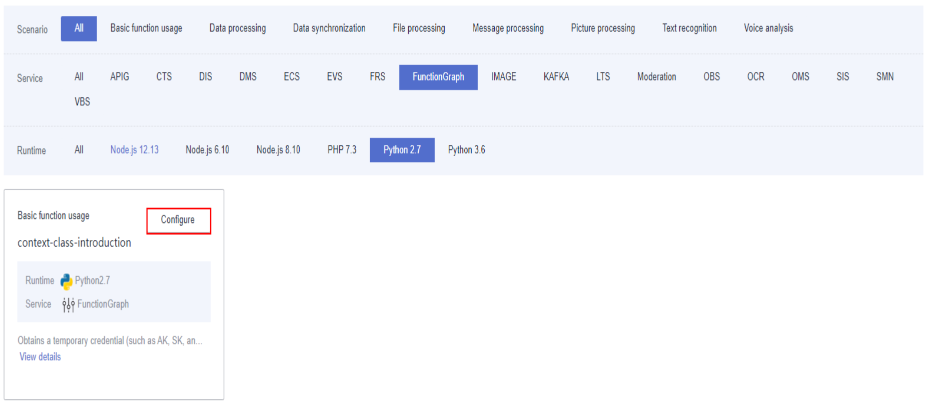
Step 1: Prepare the Environment

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

Step 2: Create a Function

1. Log in to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click **Create Function** in the upper right corner and choose **Select template**.
3. Select the template shown in **Figure 2-4** and click **Configure**.

Figure 2-4 Selecting a template



Obtains a temporary credential (such as AK, SK, an...

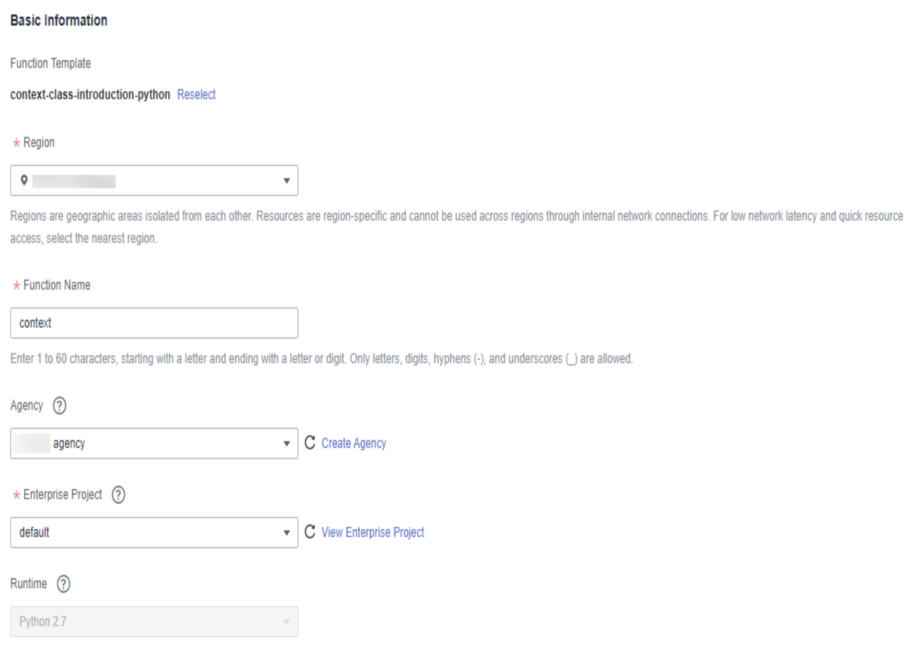
4. Set **Function Name** to **context**, select any agency from the **Agency** drop-down list, retain default values for other parameters, and click **Create Function**.

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.
```

Figure 2-5 Setting basic information



Basic Information

Function Template

context-class-introduction-python Reselect

* Region

Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

* Function Name

context

Enter 1 to 60 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.

Agency

agency

Create Agency

* Enterprise Project

default

View Enterprise Project

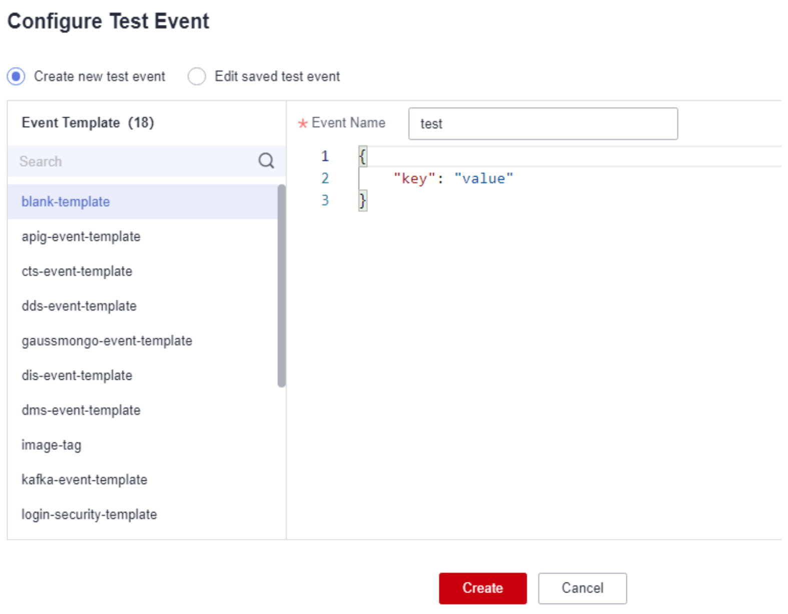
Runtime

Python 2.7

Step 3: Test the Function

1. On the function details page, click **Test**. In the displayed dialog box, create a test event.
2. Select **blank-template**, set **Event Name** to **test**, and click **Create**.

Figure 2-6 Configuring a test event



Configure Test Event

Create new test event   Edit saved test event

Event Template (18)

blank-template
apig-event-template
cts-event-template
dds-event-template
gaussmongo-event-template
dis-event-template
dms-event-template
image-tag
kafka-event-template
login-security-template

* Event Name: test

```
{
    "key": "value"
}
```

Step 4: View the Execution Result

Click **Test** and view the execution result on the right.

- **Function Output**: displays the return result of the function.
- **Log Output**: displays the execution logs of the function.
- **Summary**: displays key information of the logs.

NOTE

A maximum of 2 KB logs can be displayed. For more log information, see **Querying Function Logs**.

Step 5: View Monitoring Metrics

On the function details page, click the **Monitoring** tab.

- On the **Monitoring** tab page, choose **Metrics**, and select a time range (such as 5 minutes, 15 minutes, or 1 hour) to query the function.
- The following metrics are displayed: invocations, errors, duration (including the maximum, average, and minimum durations), and throttles.

Step 6: Delete a Function

1. On the function details page, choose **Operation** > **Delete function** in the upper right corner.
2. In the displayed dialog box, click **OK** to release resources.

2.4 Deploying a Function Using a Container Image

2.4.1 Developing an HTTP Function

Introduction

When developing an HTTP function using a custom image, implement an HTTP server in the image and listen to port 8000 for requests. HTTP functions support only APIG triggers.

Step 1: Prepare the Environment

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

Step 2: Create an Image

Take the Linux x86 64-bit OS as an example.

1. Create a folder.
```
mkdir custom_container_http_example && cd custom_container_http_example
```
2. Implement an HTTP server. Node.js is used as an example. For details about other languages, see **Creating an HTTP Function**.

   Create the **main.js** file to introduce the Express framework, receive POST requests, print the request body as standard output, and return "Hello FunctionGraph, method POST" to the client.

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

const PORT = 8000;

const app = express();
app.use(express.json());

app.post('/*', (req, res) => {
    console.log('receive', req.body);
    res.send('Hello FunctionGraph, method POST');
});

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

3. Create the **package.json** file for npm so that it can identify the project and process project dependencies.

```
{
  "name": "custom-container-http-example",
  "version": "1.0.0",
  "description": "An example of a custom container http function",
  "main": "main.js",
  "scripts": {},
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
      "express": "^4.17.1"
  }
}
```

   – **name**: project name
   – **version**: project version
   – **main**: application entry file
   – **dependencies**: all available dependencies of the project in npm

4. Create a Dockerfile.

```
FROM node:12.10.0

ENV HOME=/home/custom_container
ENV GROUP_ID=1003
ENV GROUP_NAME=custom_container
ENV USER_ID=1003
ENV USER_NAME=custom_container

RUN mkdir -m 550 ${HOME} && groupadd -g ${GROUP_ID} ${GROUP_NAME} && useradd -u $
{USER_ID} -g ${GROUP_ID} ${USER_NAME}

COPY --chown=${USER_ID}:${GROUP_ID} main.js ${HOME}
COPY --chown=${USER_ID}:${GROUP_ID} package.json ${HOME}

RUN cd ${HOME} && npm install

RUN chown -R ${USER_ID}:${GROUP_ID} ${HOME}

RUN find ${HOME} -type d | xargs chmod 500
RUN find ${HOME} -type f | xargs chmod 500

USER ${USER_NAME}
WORKDIR ${HOME}

EXPOSE 8000
ENTRYPOINT ["node", "main.js"]
```

   – **FROM**: Specify base image **node:12.10.0**. The base image is mandatory and its value can be changed.

- **ENV**: Set environment variables **HOME** (**/home/custom_container**), **GROUP_NAME** and **USER_NAME** (**custom_container**), **USER_ID** and **GROUP_ID** (**1003**). These environment variables are mandatory and their values can be changed.
- **RUN**: Use the format **RUN** *<Command>*. For example, **RUN mkdir -m 550 ${HOME}**, which means to create the **home** directory for user *${USER_NAME}* during container building.
- **USER**: Switch to user *${USER_NAME}*.
- **WORKDIR**: Switch the working directory to the **home** directory of user *${USER_NAME}*.
- **COPY**: Copy **main.js** and **package.json** to the **home** directory of user *${USER_NAME}* in the container.
- **EXPOSE**: Expose port 8000 of the container. Do not change this parameter.
- **ENTRYPOINT**: Run the **node main.js** command to start the container. Do not change this parameter.

NOTE

1. You can use any base image.
2. In the cloud environment, UID 1003 and GID 1003 are used to start the container by default. The two IDs can be modified by choosing **Configuration** > **Basic Settings** > **Container Image Override** on the function details page. They cannot be **root** or a reserved ID.

5. Build an image.

   In the following example, the image name is **custom_container_http_example**, the tag is **latest**, and the period (.) indicates the directory where the Dockerfile is located. Run the image build command to pack all files in the directory and send the package to a container engine to build an image.

```
docker build -t custom_container_http_example:latest .
```

Step 3: Perform Local Verification

1. Start the Docker container.

```
docker run -u 1003:1003 -p 8000:8000 custom_container_http_example:latest
```

2. Open a new Command Prompt, and send a message through port 8000 to access the **/invoke** directory specified in the template code.

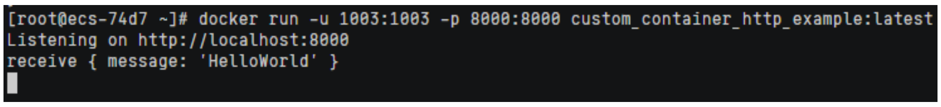
```
curl -XPOST -H 'Content-Type: application/json' -d '{"message":"HelloWorld"}' localhost:8000/
helloworld
```

   The following information is returned based on the module code:

```
Hello FunctionGraph, method POST
```

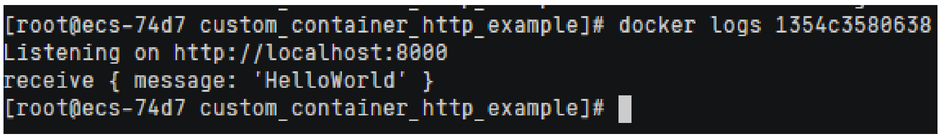
3. Check whether the following information is displayed:

```
receive {"message":"HelloWorld"}
```



```
[root@ecs-74d7 ~]# docker run -u 1003:1003 -p 8000:8000 custom_container_http_example:latest
Listening on http://localhost:8000
receive { message: 'HelloWorld' }
```

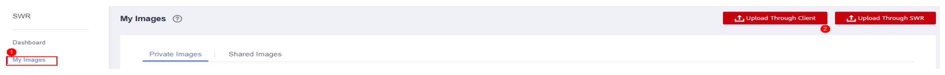
   Alternatively, run the **docker logs** command to obtain container logs.



```
[root@ecs-74d7 custom_container_http_example]# docker logs 1354c3580638
Listening on http://localhost:8000
receive { message: 'HelloWorld' }
[root@ecs-74d7 custom_container_http_example]#
```

Step 4: Upload the Image

1. Log in to the SoftWare Repository for Container (SWR) console. In the navigation pane, choose **My Images**.
2. Click **Upload Through Client** or **Upload Through SWR** in the upper right corner.
3. Upload the image as prompted.



4. View the image on the **My Images** page.

Step 5: Create a Function

1. Log in to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click **Create Function** in the upper right corner and choose **Select template**.
3. Set the basic information.
   - **Function Type**: Select **HTTP Function**.
   - **Function Name**: Enter **custom_container_http**.
   - **Use container image**: Select the image uploaded to SWR.
   - **Agency**: Select an agency with the **SWR Admin** permission. If no agency is available, create one by referring to **Creating an Agency**.
4. After the configuration is complete, click **Create Function**.

Step 6: Test the Function

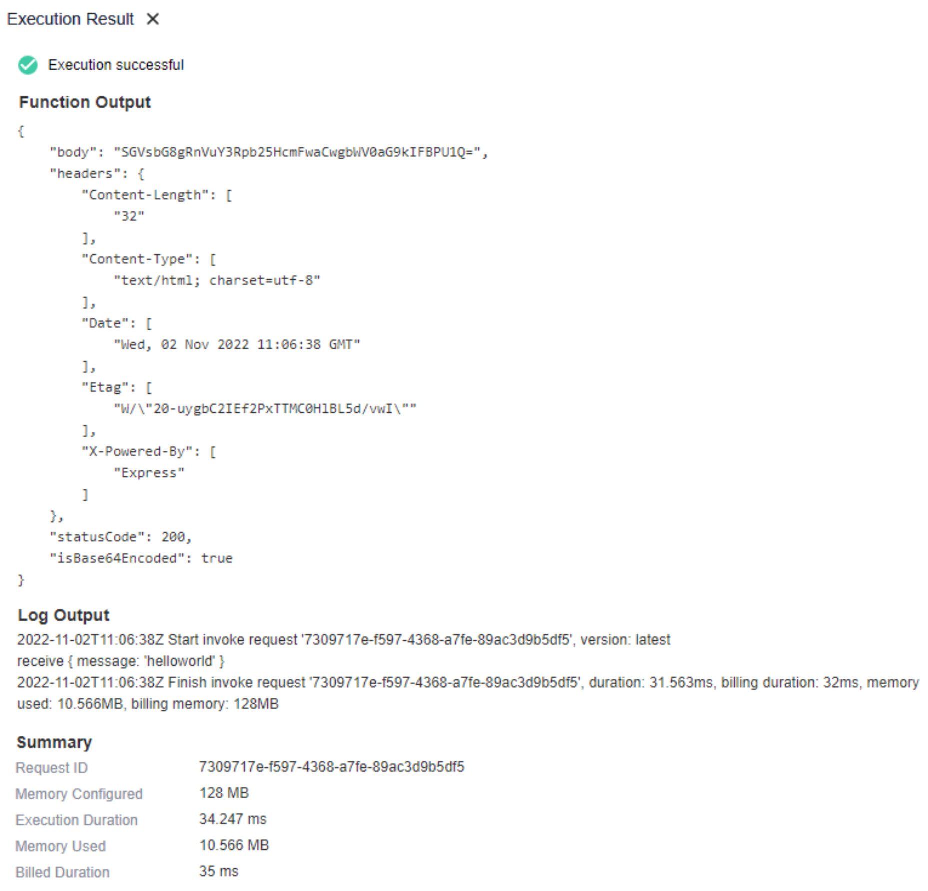
1. On the function details page, click **Test**. In the displayed dialog box, create a test event.
2. Select **apig-event-template**, set **Event Name** to **helloworld**, modify the test event as follows, and click **Create**.

```
{
    "body": "{\"message\": \"helloworld\"}",
    "requestContext": {
        "requestId": "11cdcdcf33949dc6d722640a13091c77",
        "stage": "RELEASE"
    },
    "queryStringParameters": {
        "responseType": "html"
    },
    "httpMethod": "POST",
    "pathParameters": {},
    "headers": {
        "Content-Type": "application/json"
    },
    "path": "/helloworld",
    "isBase64Encoded": false
}
```

Step 7: View the Execution Result

Click **Test** and view the execution result on the right.

Figure 2-7 Execution result



Execution Result ✕

✓ Execution successful

Function Output

```
{
    "body": "SGVsbG8gRnVuY3Rpb25HcmFwaCwgbWV0aG9kIFBPU1Q=",
    "headers": {
        "Content-Length": [
            "32"
        ],
        "Content-Type": [
            "text/html; charset=utf-8"
        ],
        "Date": [
            "Wed, 02 Nov 2022 11:06:38 GMT"
        ],
        "Etag": [
            "W/\"20-uygbC2IEf2PxTTMC0H1BL5d/vwI\""
        ],
        "X-Powered-By": [
            "Express"
        ]
    },
    "statusCode": 200,
    "isBase64Encoded": true
}
```

Log Output

```
2022-11-02T11:06:38Z Start invoke request '7309717e-f597-4368-a7fe-89ac3d9b5df5', version: latest
receive { message: 'helloworld' }
2022-11-02T11:06:38Z Finish invoke request '7309717e-f597-4368-a7fe-89ac3d9b5df5', duration: 31.563ms, billing duration: 32ms, memory
used: 10.566MB, billing memory: 128MB
```

Summary

Request ID	7309717e-f597-4368-a7fe-89ac3d9b5df5
Memory Configured	128 MB
Execution Duration	34.247 ms
Memory Used	10.566 MB
Billed Duration	35 ms

- **Function Output**: displays the return result of the function.
- **Log Output**: displays the execution logs of the function.
- **Summary**: displays key information of the logs.

NOTE

A maximum of 2 KB logs can be displayed. For more log information, see **Querying Function Logs**.

Step 8: View Monitoring Metrics

On the function details page, click the **Monitoring** tab.

- On the **Monitoring** tab page, choose **Metrics**, and select a time range (such as 5 minutes, 15 minutes, or 1 hour) to query the function.
- The following metrics are displayed: invocations, errors, duration (including the maximum, average, and minimum durations), and throttles.

Step 9: Delete the Function

1. On the function details page, choose **Operation** > **Delete function** in the upper right corner.
2. In the displayed dialog box, click **OK** to release resources.

3 Before You Start

3.1 Use of FunctionGraph

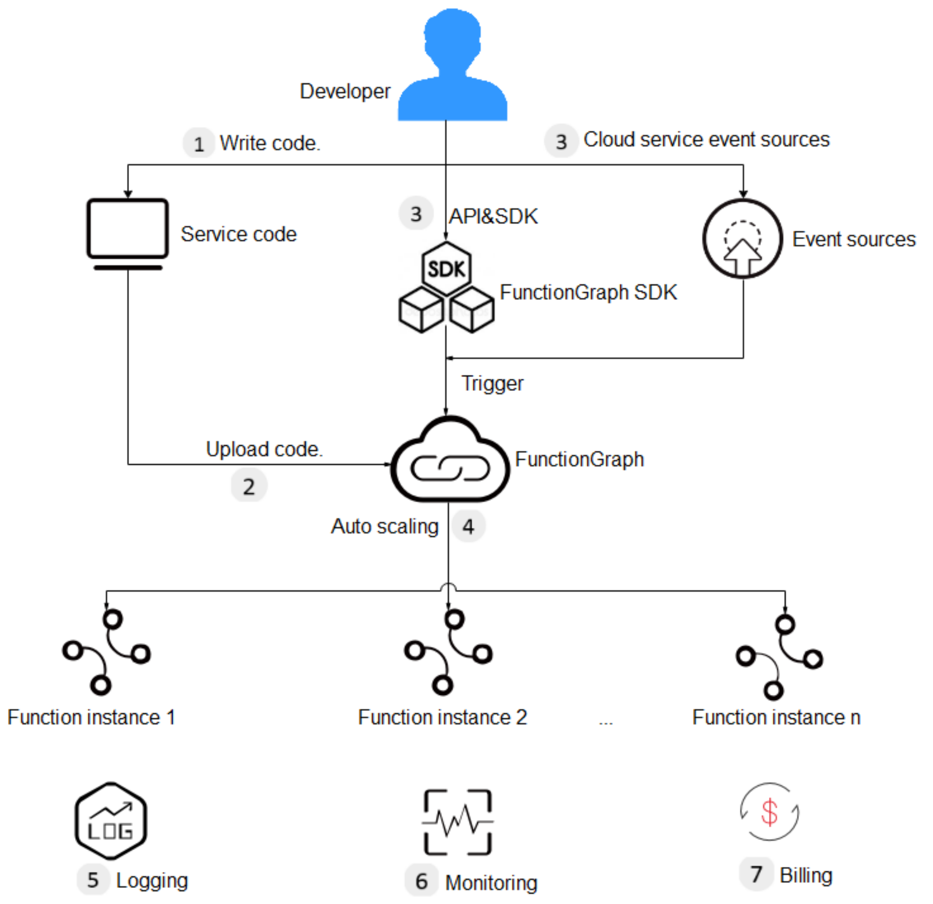
FunctionGraph allows you to run your code without provisioning or managing servers, while ensuring high availability and scalability. All you need to do is upload your code and set execution conditions, and FunctionGraph will take care of the rest.

Process

Figure 3-1 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), Object Storage Service (OBS), 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 3-1 Flowchart



The following shows the details:

1. Write code.

   Write code in Node.js, Python, Java, or Go.
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 as FunctionGraph is interconnected with Log Tank Service (LTS). For details, see **Logs**.
6. View monitoring information.

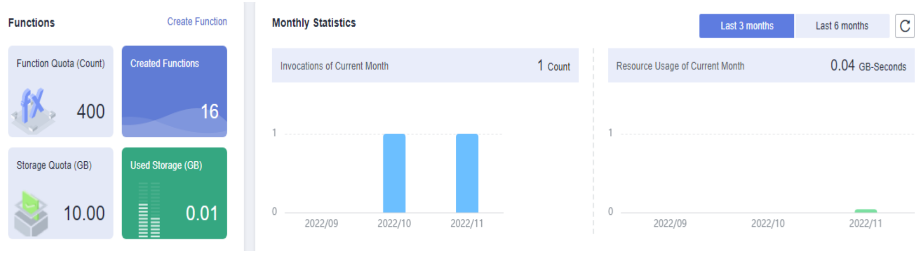
   View graphical monitoring information as FunctionGraph is interconnected with Cloud Eye. For details, see **Metrics**.

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 3-2 Monthly statistics



- View tenant-level metrics, including invocations, errors, duration, and throttles.

  Table 3-1 describes the function metrics.

Table 3-1 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 all functions are executed at a time within a period. **Minimum Duration**: the minimum duration all functions are executed at a time within a period. **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.
Throttles	Count	Number of times that FunctionGraph throttles your functions due to the resource limit.

3.2 Permissions Management

3.2.1 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.

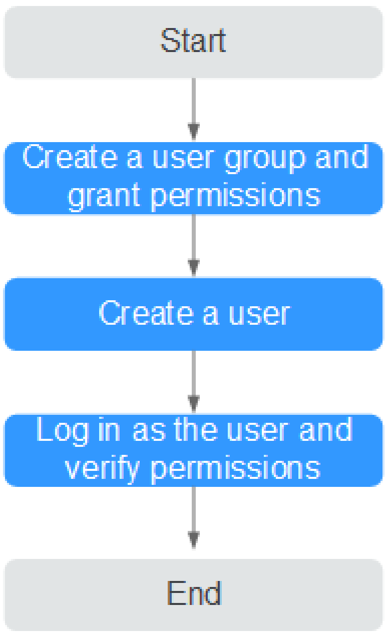
This section describes the procedure for granting permissions. For details, see **Figure 3-3**.

Prerequisites

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

Process

Figure 3-3 Process for granting FunctionGraph permissions


Start
Create a user group and grant permissions
Create a user
Log in as the user and verify permissions
End

1. .

   Create a user group on the IAM console, and assign the **FunctionGraph Invoker** role to the group.
2. .

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

3. 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.

3.2.2 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/*"
            ]
        }
    ]
}
```

3.3 Supported Programming Languages

3.3.1 Node.js

√: Supported. **×**: Not supported.

Runtime	Supported
Node.js 6.10	√

Runtime	Supported
Node.js 8.10	√
Node.js 10.16	√
Node.js 12.13	√
Node.js 14.18	√

3.3.2 Python

√: Supported. ×: Not supported.

Runtime	Supported
Python 2.7	√
Python 3.6	√
Python 3.9	√

3.3.3 Java

√: Supported. ×: Not supported.

Runtime	Supported
Java 8	√
Java 11	√

3.3.4 Go

√: Supported. ×: Not supported.

Runtime	Supported
Go 1.x	√

3.3.5 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**.

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.

- **Next Invocation**

  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 3-2 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.

Parameter	Description
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.
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 posts 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 3-2**.

  $RUNTIME_API_ADDR is a system environment variable. For more information, see **Table 3-3**.

  This API is used to send an error invocation response to FunctionGraph. After the runtime invokes the function handler, it posts 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 3-3 Environment variables

Key	Description
RUNTIME_PROJECT_ID	Project ID
RUNTIME_FUNC_NAME	Function name
RUNTIME_FUNC_VERSION	Function version

Key	Description
RUNTIME_PACKAGE	App to which the function belongs
RUNTIME_HANDLER	Function handler
RUNTIME_TIMEOUT	Function timeout duration
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 3-4 describes the environment variables used in the preceding code.

Table 3-4 Environment variables

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

4 Building a FunctionGraph Function

4.1 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 4-1 lists the code entry modes supported by FunctionGraph for each runtime.

Table 4-1 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
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 4-2 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. ● **File**: Create files and folders, save changes, and close all files. ● **Edit**: Undo/redo typing; cut, copy, and paste code; find and replace content. ● **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**. 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.
Upload file from OBS	1. On the **Code** tab of the function details page, choose **Upload** > **OBS ZIP**. 2. Click **Select File** and upload a local code package to FunctionGraph.

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 specialist.

- 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 specialist.
- 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 specialist.

- 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 specialist.

- 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 specialist.

- 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**.

4.2 Creating a Function from Scratch

4.2.1 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 using **a template** or **container image**.

NOTE

When creating a function from scratch, configure the basic and code information based on **Table 4-3**. 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 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 4-3**. The parameters marked with an asterisk (*) are mandatory.

Table 4-3 Basic information

Parameter	Description
* Function Type	● Event functions: triggered by triggers. ● HTTP functions: triggered once HTTP requests are sent to specific URLs. **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. ● HTTP functions support APIG and APIC triggers only. ● 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: ● Consists of 1 to 60 characters, and can contain letters, digits, hyphens (-), and underscores (_). ● 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**. 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.
Runtime	Select a runtime to compile the function. **NOTICE** CloudIDE supports Node.js and PHP only.

4. Click **Create**. 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.

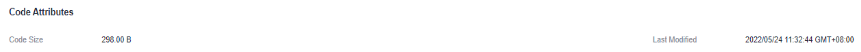
2. You can modify the code and click **Deploy** to deploy the code again.

Viewing Code Information

1. View code attributes.

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

   Figure 4-1 Viewing code attributes

   Code Attributes

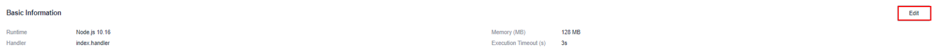
   Code Size 298.00 B

   Last Modified 2022/05/24 11:32:44 GMT+08:00

2. View basic information.

   Figure 4-2 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 **Configuring Basic Settings**.

   Figure 4-2 Editing basic information

   Basic Information

   Runtime Node.js 10.16

   Handler index.handler

   Memory (MB) 128 MB

   Execution Timeout (s) 3s

NOTICE

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

Table 4-4 Default basic information of each runtime

Runtime	Default Basic Information
JAVA	Memory (MB): 512 Handler: com.demo.TriggerTests.apigTest Execution Timeout (s): 15
Node.js	Memory (MB): 128 Handler: index.handler Execution Timeout (s): 3
Custom	Memory (MB): 128 Handler: bootstrap Execution Timeout (s): 3
Python	Memory (MB): 128 Handler: index.handler Execution Timeout (s): 3

Runtime	Default Basic Information
Go 1.x	Memory (MB): 128 Handler: handler Execution Timeout (s): 3

4.2.2 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 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 Java 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 4-5**.

Prerequisites

1. You have prepared a Java JAR package.
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:

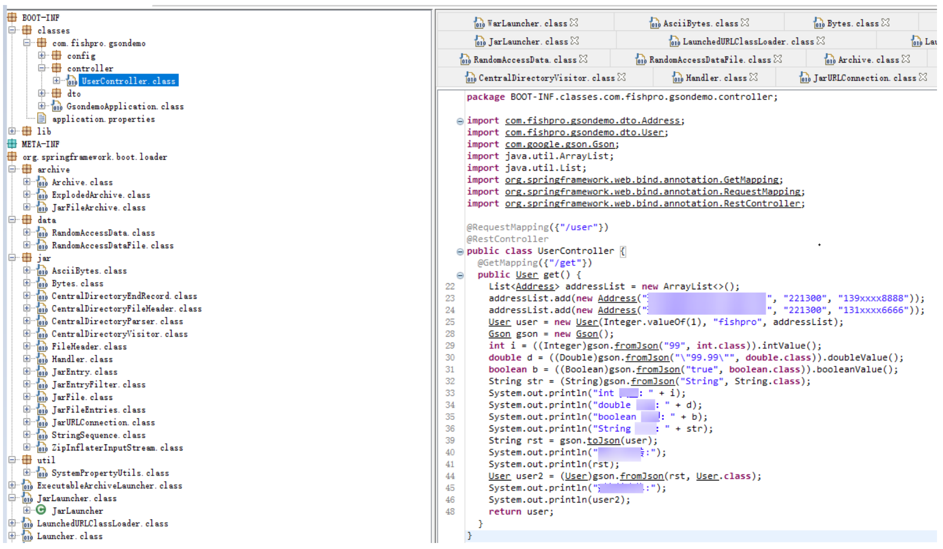
   ```
   /opt/function/runtime/java8/rtsp/jre/bin/java -jar -Dfile.encoding=utf-8 /opt/function/code/
   gsondemo-0.0.1-SNAPSHOT.jar
   ```

   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
   ```

   – **/opt/function/runtime/java8/rtsp/jre/bin/java**: Java path.
   – **/opt/function/code**: path of the function code package.
   – **gsondemo-0.0.1-SNAPSHOT.jar**: example JAR package. The service path is **/user/get**.



```
BOOT-INF
  classes
    com.fishpro.gsondemo
      config
      controller
        UserController.class
      dto
      GsondemoApplication.class
    application.properties
  lib
META-INF
org.springframework.boot.loader
  archive
    Archive.class
    ExplodedArchive.class
    JarFileArchive.class
  data
    RandomAccessData.class
    RandomAccessDataFile.class
  jar
    AsciiBytes.class
    Bytes.class
    CentralDirectoryEndRecord.class
    CentralDirectoryFileHeader.class
    CentralDirectoryParser.class
    CentralDirectoryVisitor.class
    FileHeader.class
    Handler.class
    JarEntry.class
    JarEntryFilter.class
    JarFile.class
    JarFileEntries.class
    JarURLConnection.class
    StringSequence.class
    ZipInflaterInputStream.class
  util
    SystemPropertyUtils.class
  ExecutableArchiveLauncher.class
  JarLauncher.class
    JarLauncher
  LaunchedURLClassLoader.class
  Launcher.class
```

```java
package BOOT-INF.classes.com.fishpro.gsondemo.controller;

import com.fishpro.gsondemo.dto.Address;
import com.fishpro.gsondemo.dto.User;
import com.google.gson.Gson;
import java.util.ArrayList;
import java.util.List;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RequestMapping({"/user"})
@RestController
public class UserController {
  @GetMapping({"/get"})
  public User get() {
    List<Address> addressList = new ArrayList<>();
    addressList.add(new Address("[redacted]", "221300", "139xxxx8888"));
    addressList.add(new Address("[redacted]", "221300", "131xxxx6666"));
    User user = new User(Integer.valueOf(1), "fishpro", addressList);
    Gson gson = new Gson();
    int i = ((Integer)gson.fromJson("99", int.class)).intValue();
    double d = ((Double)gson.fromJson("\"99.99\"", double.class)).doubleValue();
    boolean b = ((Boolean)gson.fromJson("true", boolean.class)).booleanValue();
    String str = (String)gson.fromJson("String", String.class);
    System.out.println("int [redacted]: " + i);
    System.out.println("double [redacted]: " + d);
    System.out.println("boolean [redacted]: " + b);
    System.out.println("String [redacted]: " + str);
    String rst = gson.toJson(user);
    System.out.println("[redacted]:");
    System.out.println(rst);
    User user2 = (User)gson.fromJson(rst, User.class);
    System.out.println("[redacted]:");
    System.out.println(user2);
    return user;
  }
}
```

To use another runtime, change the runtime path by referring to **Table 4-5**. The code package path does not need to be changed.

Table 4-5 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.js16	/opt/function/runtime/nodejs16.17/rtsp/nodejs/bin/node
Node.js18	/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
Python 3.9	/opt/function/runtime/python3.9/rtsp/python/bin/python3

Procedure

1. Create a function.

   a. Create an HTTP function. For details, see **Creating an Event Function**. Pay special attention to the following parameters:

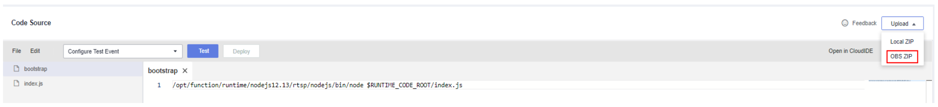
      ▪ **Function Type**: HTTP function

      ▪ **Region**: Select a region where you will deploy your code.

   b. Upload the code. For example, upload a ZIP file from OBS. After the upload is complete, click **Deploy**.

      Zip the JAR package and bootstrap file, and choose **Upload** > **OBS ZIP**.

      Figure 4-3 Uploading a ZIP file from OBS



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

2. Create a trigger.

   NOTE

   HTTP functions support APIG 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 (dedicated) trigger as an example. For more information, see **Using an APIG (Dedicated) 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.

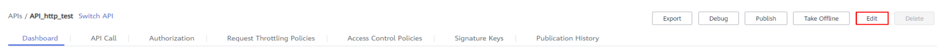
   c. When the configuration is complete, click **OK**. After the trigger is created, **API_test_http** will be generated on the API Gateway console.

3. Publish the API.

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

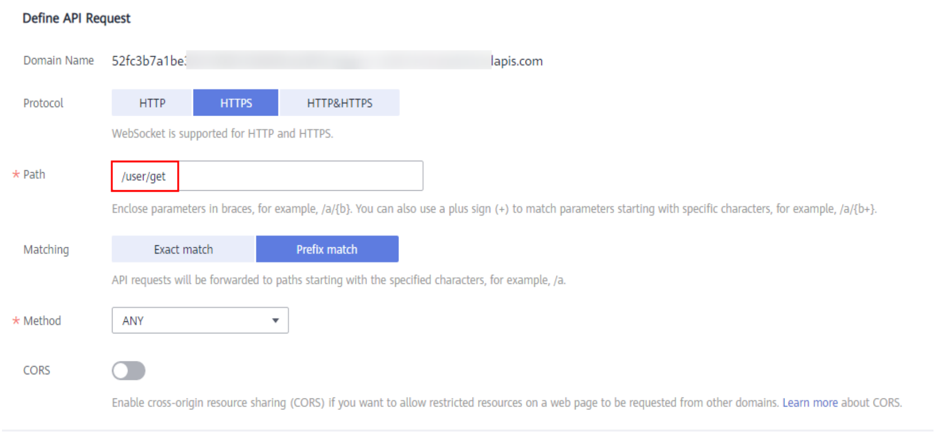
   b. Click **Edit** in the upper right corner. The **Basic Information** page is displayed.

      Figure 4-4 Editing an API



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

Figure 4-5 Defining an API request



Define API Request

Domain Name 52fc3b7a1be[redacted]lapis.com

Protocol HTTP **HTTPS** HTTP&HTTPS

WebSocket is supported for HTTP and HTTPS.

* Path /user/get

Enclose parameters in braces, for example, /a/{b}. You can also use a plus sign (+) to match parameters starting with specific characters, for example, /a/{b+}.

Matching Exact match **Prefix match**

API requests will be forwarded to paths starting with the specified characters, for example, /a.

* Method ANY

CORS

Enable cross-origin resource sharing (CORS) if you want to allow restricted resources on a web page to be requested from other domains. Learn more about CORS.

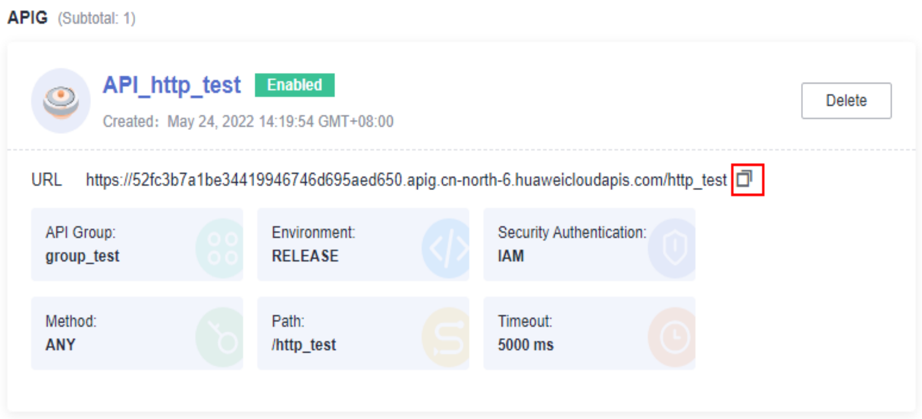
d. Click **Publish API**. On the displayed page, click **Publish**.

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 4-6 Copying the URL



APIG (Subtotal: 1)

API_http_test Enabled

Created: May 24, 2022 14:19:54 GMT+08:00

Delete

URL https://52fc3b7a1be34419946746d695aed650.apig.cn-north-6.huaweicloudapis.com/http_test

API Group: group_test

Environment: RELEASE

Security Authentication: IAM

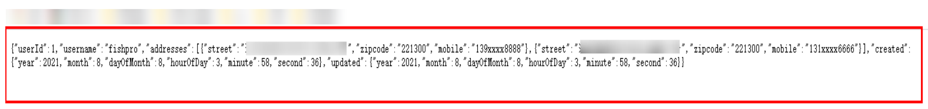
Method: ANY

Path: /http_test

Timeout: 5000 ms

   c. View the request result.

Figure 4-7 Viewing the request result



```
{"userId":1,"username":"fishpro","addresses":[{"street":"[redacted]","zipcode":"221300","mobile":"139xxxx8888"},{"street":"[redacted]","zipcode":"221300","mobile":"131xxxx6666"}],"created":
{"year":2021,"month":8,"dayOfMonth":8,"hourOfDay":3,"minute":58,"second":36},"updated":{"year":2021,"month":8,"dayOfMonth":8,"hourOfDay":3,"minute":58,"second":36}}
```

Common Function Request Headers

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

Table 4-6 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	ProjectID
X-CFF-Package	App to which the function belongs
X-CFF-Region	Current region

4.3 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

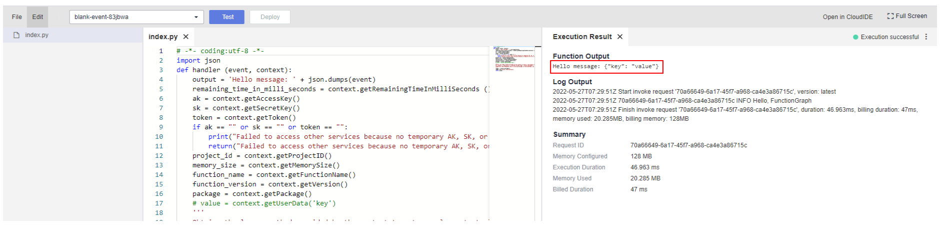
   If no agency is configured, the following message will be displayed when the function is triggered:

   ```
   Failed to access other services because no temporary AK, SK, or token has been obtained. Please set an agency.
   ```
5. Click **Save**.

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 4-8 Successful execution result



```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    output = 'Hello message: ' + json.dumps(event)
    remaining_time_in_milli_seconds = context.getRemainingTimeInMilliSeconds (
    ak = context.getAccessKey()
    sk = context.getSecretKey()
    token = context.getToken()
    if ak == "" or sk == "" or token == "":
        print("Failed to access other services because no temporary AK, SK, or
        return("Failed to access other services because no temporary AK, SK, or
    project_id = context.getProjectID()
    memory_size = context.getMemorySize()
    function_name = context.getFunctionName()
    function_version = context.getVersion()
    package = context.getPackage()
    # value = context.getUserData('key')
    '''
```

Function Output

```
Hello message: {"key": "value"}
```

Log Output

```
2022-05-27T07:29:51Z Start invoke request '70a66649-6a17-45f7-a968-ca4e3a86715c', version: latest
2022-05-27T07:29:51Z 70a66649-6a17-45f7-a968-ca4e3a86715c INFO Hello, FunctionGraph
2022-05-27T07:29:51Z Finish invoke request '70a66649-6a17-45f7-a968-ca4e3a86715c', duration: 46.963ms, billing duration: 47ms,
memory used: 20.285MB, billing memory: 128MB
```

Summary

Request ID	70a66649-6a17-45f7-a968-ca4e3a86715c
Memory Configured	128 MB
Execution Duration	46.963 ms
Memory Used	20.285 MB
Billed Duration	47 ms

4.4 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 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**.

The following features are supported:

- **Downloading images**

  Images are stored in 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**.

- **Reserved instances**

  For details, see the description about reserved instances.

NOTE

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

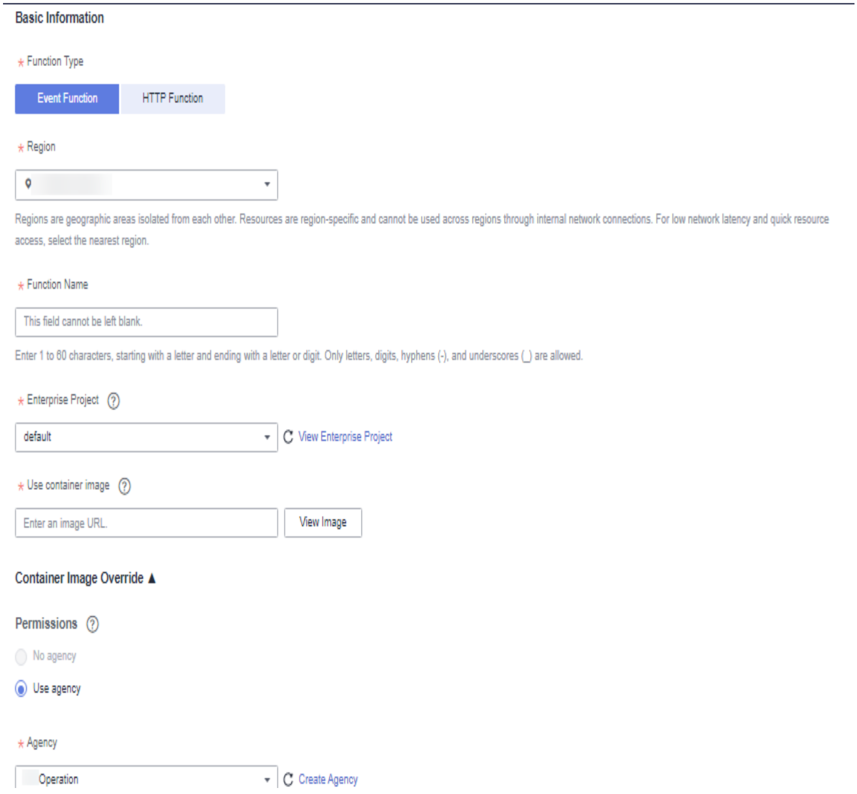
Prerequisites

You have created an agency with the **SWR Admin** permissions 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 **Use container image**. For details, see **Table 4-7**.

Figure 4-9 Creating a function using a container image



Basic Information

* Function Type

Event Function | HTTP Function

* Region

Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

* Function Name

This field cannot be left blank.

Enter 1 to 60 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.

* Enterprise Project

default | View Enterprise Project

* Use container image

Enter an image URL. | View Image

Container Image Override ▲

Permissions

No agency

Use agency

* Agency

Operation | Create Agency

Table 4-7 Parameter description

Parameter	Description
*Function Type	Select a function type. HTTP functions: triggered once HTTP requests are sent to specific URLs. **NOTE** ● The custom container image must contain an HTTP server with listening port 8000. ● HTTP functions support APIG and APIC triggers only.

Paramete r	Description
*Region	Select a region where you will deploy your code.
*Function Name	Name of the function, which must meet the following requirements: ● Consists of 1 to 60 characters, and can contain letters, digits, hyphens (-), and underscores (_). ● 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.
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.
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 (,). ● **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 (,). ● **Working Dir**: working directory that a container runs. If no directory is specified, the directory in the image configuration will be used. The directory must start with a slash (/). ● **User ID**: user ID for running the image. If no user ID is specified, the default value **1003** will be used. ● **Group ID**: user group ID. If no user group ID is specified, the default value **1003** will be used.
Agency	Select an agency with **SWR Admin** permissions. 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.
- The image must be public.
- 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.

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}`);
});
```

5 Configuring the Function

5.1 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.

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.

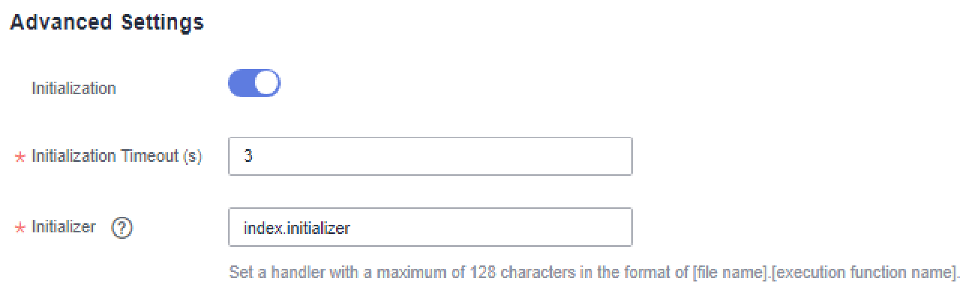
Initializing a Function

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

Figure 5-1 Enabling initialization



Initialization

* Initialization Timeout (s): 3

* Initializer: index.initializer

Set a handler with a maximum of 128 characters in the format of [file name].[execution function name].

Table 5-1 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. 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]*. **NOTE** This parameter is not required if function initialization is disabled.

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

5.2 Configuring Basic Settings

Introduction

After a function is created, **Memory**, **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 5-2**. Parameters marked with an asterisk (*) are mandatory.

Table 5-2 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. **NOTICE** 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 (.). Example: **myfunction.handler** ● For a Java function, the handler must be named in the format of *[package name].[class name].[execution function name]*. Example: **com.xxxxx.exp.Myfunction.myHandler** ● For a Go function, the handler name must be the same as the executable file name in the uploaded code package. Example: If the executable file is **handler**, set this parameter to **handler**.
*Enterprise Project	Select a created enterprise project and add the function to it. By default, **default** is selected.
*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. The value ranges from 3s to 259,200s.
Memory (MB)	Memory of a function instance. Options: 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2560, 3072, 3584, 4096, 8192, 10,240.
Ephemeral Storage (MB)	Options: 512 and 10,240. Default: 512. By default, the **/tmp** directory of each function is 512 MB. You can increase the size to 10,240 MB (10 GB) if necessary. Before doing so, contact technical support to grant you the required permissions.

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

4. Click **Save**.

5.3 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, **create an agency** and select the required action by referring to **Table 5-3**.

Table 5-3 Common actions

Scenario	Action	Description
Using a custom image	SWR Admin	SWR Admin: administrator who has all permissions for the SoftWare Repository for Container (SWR) service. For details about how to create a custom image, see **Deploying a Function Using a Container Image**.
Mounting an SFS Turbo file system	SFS Administrator or Tenant administrator	SFS Administrator: administrator who has all permissions for the Scalable File Service (SFS) service. Tenant administrator: administrator for all cloud services except IAM. This user can perform any operations on all cloud resources of the enterprise. For details about how to mount an SFS Turbo file system, see **Mounting an SFS Turbo File System**.

Scenario	Action	Description
Mounting an ECS shared directory	Tenant Guest and VPC Administrator	Tenant Guest: user with read-only permissions for all cloud services (except IAM) VPC Administrator: network administrator The agency must have at least the Tenant Guest and VPC Administrator permissions. For details about how to mount an ECS shared directory, see **Mounting an ECS Shared Directory**.
Configuring cross-domain VPC access	VPC Administrator	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. For details about how to configure cross-domain VPC access, see **Configuring the Network**.
Creating an OBS bucket and trigger	Tenant Administrator	Tenant administrator: administrator for all cloud services except IAM. This user can perform any operations on all cloud resources of the enterprise. For details about how to create an OBS trigger, see **Using an OBS Trigger**.

Creating an Agency

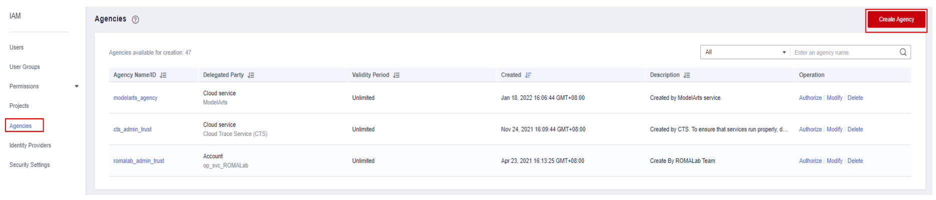
NOTE

In the following example, the **Tenant 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.

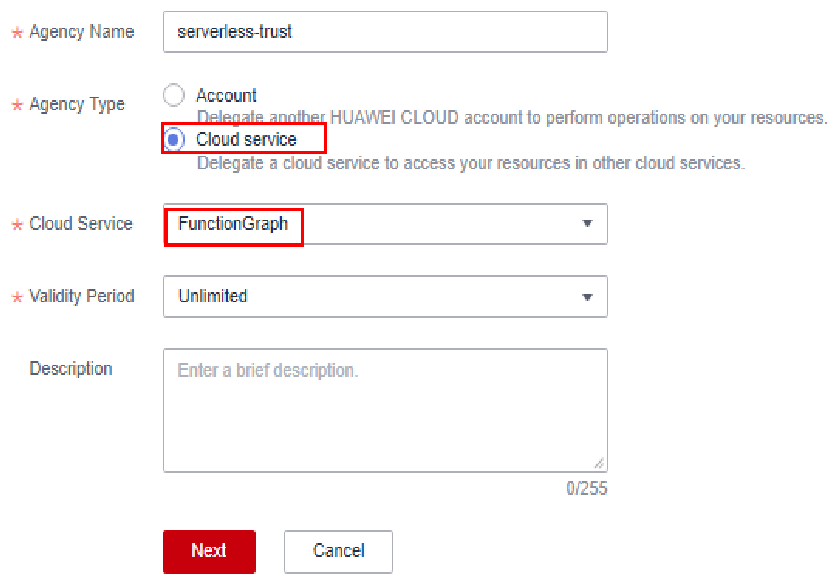
Figure 5-2 Creating an agency



Agencies available for creation: 47

Agency Name/ID	Delegated Party	Validity Period	Created	Description	Operation
modelarts_agency	Cloud service ModelArts	Unlimited	Jan 18, 2022 16:06:44 GMT+08:00	Created by ModelArts service.	Authorize | Modify | Delete
cts_admin_trust	Cloud service Cloud Trace Service (CTS)	Unlimited	Nov 24, 2021 16:09:44 GMT+08:00	Created by CTS. To ensure that services run properly, d...	Authorize | Modify | Delete
romalab_admin_trust	Account op_svc_ROMALab	Unlimited	Apr 23, 2021 16:13:25 GMT+08:00	Create By ROMALab Team	Authorize | Modify | Delete

3. Configure the agency.



Agency Name: serverless-trust

Agency Type: Account — Delegate another HUAWEI CLOUD account to perform operations on your resources. [✓ Cloud service — Delegate a cloud service to access your resources in other cloud services.]

Cloud Service: FunctionGraph

Validity Period: Unlimited

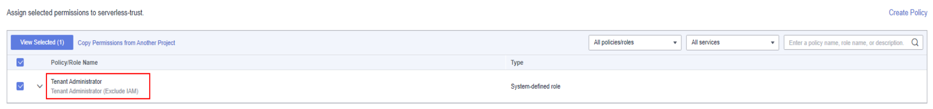
Description: Enter a brief description.

0/255

Figure 5-3 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 **Tenant Administrator** permission is used as an example.



Assign selected permissions to serverless-trust.

✓	Policy/Role Name	Type
✓	Tenant Administrator Tenant Administrator (Exclude IAM)	System-defined role

Create Policy
View Selected (1)
Copy Permissions from Another Project
All policies/roles
All services
Enter a policy name, role name, or description.

Figure 5-4 Selecting policies

Table 5-4 Example of agency permissions

Policy Name	Scenario
Tenant Administrator	Administrator for all cloud services except IAM. This user can perform any operations on all cloud resources of the enterprise.

5. Click **Next** and select the scope.

Configuring an Agency

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** > **Permissions**, click **Create Agency**, and set an agency based on site requirements by referring to 2–5.

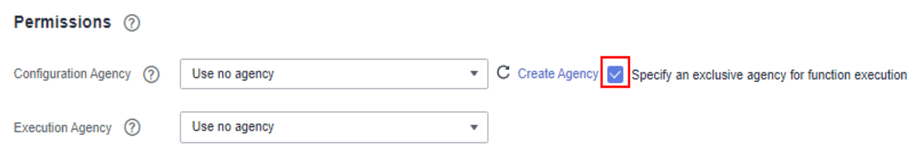
Table 5-5 Agency configuration parameters

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 5-5** shows the agency options.

Figure 5-5 Setting agencies



Permissions

Configuration Agency: Use no agency | Create Agency | ☑ Specify an exclusive agency for function execution

Execution Agency: Use no agency

- **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.

5.4 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 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 NAT gateway for the VPC and bind an EIP to the gateway. For details, see **Configuring a Fixed Public IP Address**.

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 5-6**

Table 5-6 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.

   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. You can bind functions in a project to up to four different subnets in any VPCs. (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. 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 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. Log in to the NAT Gateway console, and click **Create 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 **Creating a Public NAT Gateway**.
3. Click the NAT gateway name. On the details page that is displayed, click **Add SNAT Rule**, set the rule, and click **OK**.

5.5 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

FunctionGraph supports the following types of file systems:

- 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 512 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**.

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.

Configuring VPC Access

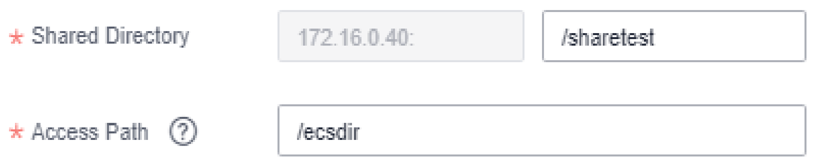
Before adding an ECS shared directory, specify the VPC where the ECS is deployed. View the VPC information on the details page of the ECS. Click the VPC name to go to the VPC details page, and view the subnet.

Set the acquired VPC and subnet for the function.

Mounting an ECS Directory

Enter a shared directory and function access path.

Figure 5-6 Setting the path


* Shared Directory: 172.16.0.40: /sharetest
* Access Path: /ecsdir

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.

Creating an NFS Shared Directory on ECS

1. **Linux**
   - CentOS, SUSE, EulerOS, Fedora, or openSUSE
     1. 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.cloud.com/euler/2.3/os/x86_64/
        enabled=1
        gpgcheck=1
        gpgkey=http://repo.cloud.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.cloud.com/euler/2.5/os/x86_64/
        enabled=1
        gpgcheck=1
        gpgkey=http://repo.cloud.com/euler/2.5/os/RPM-GPG-KEY-EulerOS
        ```

NOTE

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.

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.

iii. Start the NFS service.

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

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.

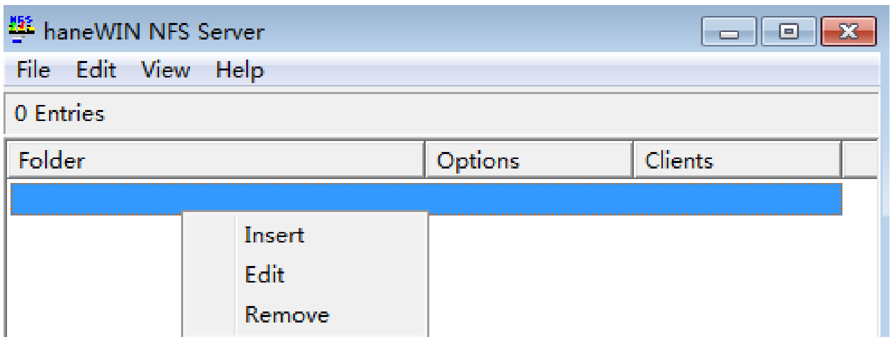
   – In the case of WinNFSd, see **WinNFSd configuration**.

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

   i. Run **nfsctl.exe** as the Windows administrator.

   ii. Right-click in the blank area and choose **Insert** from the shortcut menu.

Figure 5-7 Insert



5.6 Configuring Environment Variables

Overview

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

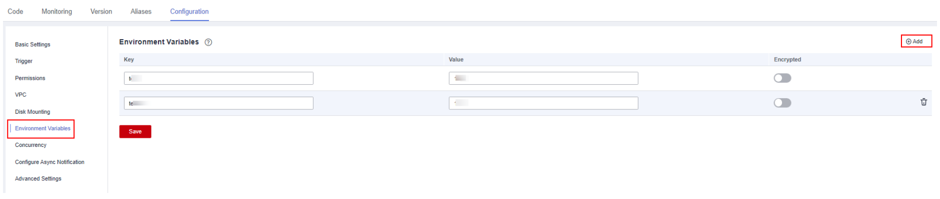
Scenario

- 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 5-8 Adding environment variables



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

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.
- When you define environment variables, FunctionGraph displays all your input information in plain text. To ensure security, 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 5-7 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.
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. Unit: MB
RUNTIME_MAX_RESP_BODY_SIZE	Maximum size of a response body	Obtain the value from a system environment variable. Default value: 6,291,456 bytes

Environment Variable	Description	Obtaining Method and Default Value
RUNTIME_INITIALIZER_HANDLER	Initializer	Obtain the value from a system environment variable.
RUNTIME_INITIALIZER_TIMEOUT	Initialization timeout of a function	Obtain the value from a system environment variable.
RUNTIME_ROOT	Runtime package path	Obtain the value from a system environment variable. Default value: **/home/snuser/runtime**
RUNTIME_CODE_ROOT	Path for storing code in a container	Obtain the value from a system environment variable. Default value: **/opt/function/code**
RUNTIME_LOG_DIR	Path for storing system logs in a container	Obtain the value from a system environment variable. 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.

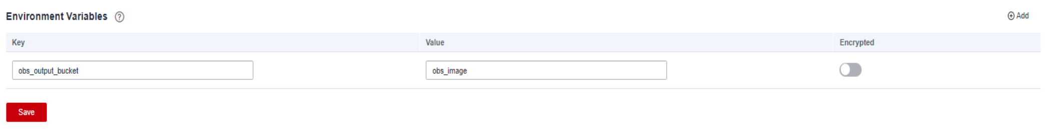
In the following code snippet, **obs_output_bucket** is the bucket used for storing processed images.

```
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'
```

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

Figure 5-9 Environment variables


Environment Variables

Key	Value	Encrypted
obs_output_bucket	obs_image	

5.7 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.

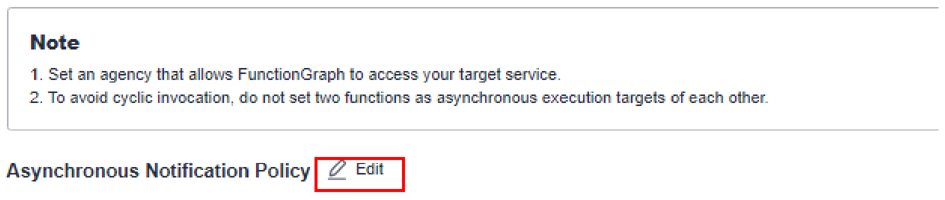
Procedure

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**.

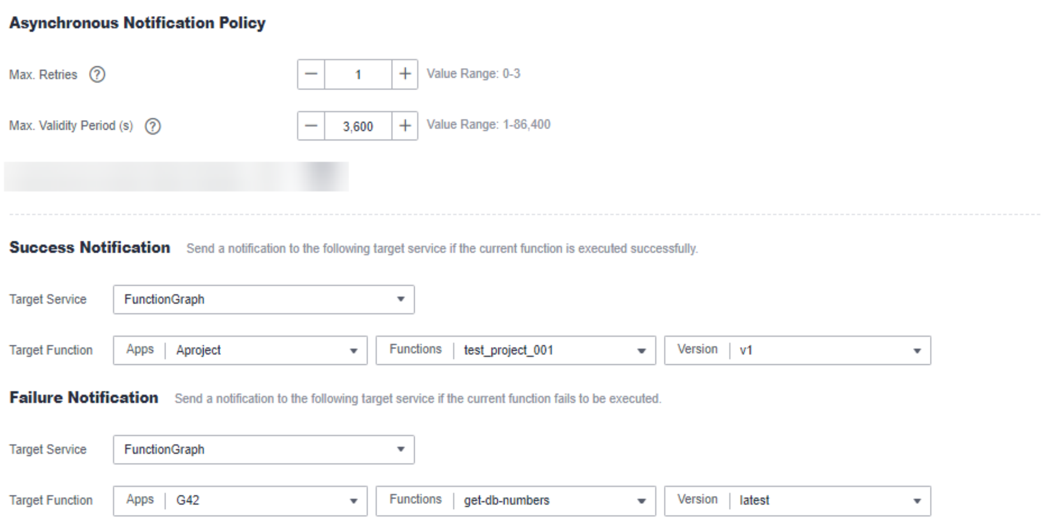
Figure 5-10 Configuring an asynchronous notification policy


Note

1. Set an agency that allows FunctionGraph to access your target service.
2. To avoid cyclic invocation, do not set two functions as asynchronous execution targets of each other.

Asynchronous Notification Policy

Step 4 Set parameters by referring to **Table 5-8**. For example, specify **FunctionGraph** for **Target Service**.


Asynchronous Notification Policy

Max. Retries: 1 (Value Range: 0-3)

Max. Validity Period (s): 3,600 (Value Range: 1-86,400)

Success Notification Send a notification to the following target service if the current function is executed successfully.

Target Service: FunctionGraph

Target Function: Apps | Aproject; Functions | test_project_001; Version | v1

Failure Notification Send a notification to the following target service if the current function fails to be executed.

Target Service: FunctionGraph

Target Function: Apps | G42; Functions | get-db-numbers; Version | latest

Figure 5-11 Setting parameters

Table 5-8 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**. • **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. 1. FunctionGraph 2. OBS
Failure Notification	**Target Service**: to which a notification will be sent if a function fails to be executed. 1. FunctionGraph 2. OBS

Step 5 Click **OK**.

NOTE

1. Set an agency that allows FunctionGraph to access the target service.
2. 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 5-9**. 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 5-9 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.

5.8 Configuring Single-Instance Multi-Concurrency

NOTE

This feature is supported only by FunctionGraph v2.

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.

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 5-10 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.
		log = context.getLogger()
		log.info("test")

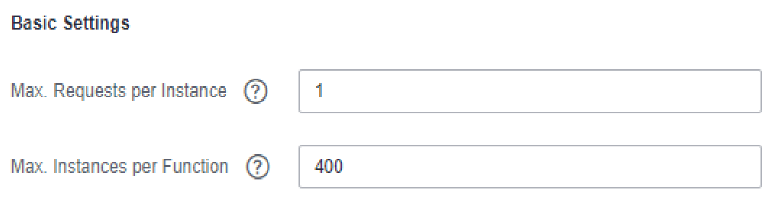
Shared variables	Not involved.	Modifying shared variables will cause errors. Mutual exclusion protection is required when you modify non-thread-safe variables during function writing.
Monitoring metrics	Perform monitoring based on the actual situation.	Under the same load, the number of function instances 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 5-11** and click **Save**.

Figure 5-12 Concurrency configuration


Basic Settings

Max. Requests per Instance: 1

Max. Instances per Function: 400

Table 5-11 Description

Parameter	Description
Max. Requests per Instance	Number of concurrent requests supported by a single instance. Value range: 1–1000.
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. **0**: The function is disabled. **NOTE** Requests that exceed the processing capability of instances will be discarded. 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.

5.9 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.

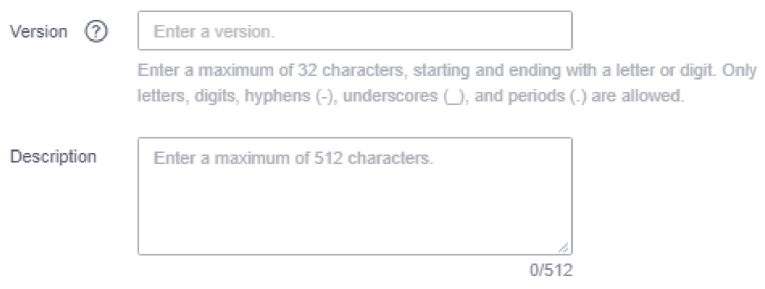
NOTE

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 5-13 Parameters for publishing a new version

   Version ?
   Enter a version.
   Enter a maximum of 32 characters, starting and ending with a letter or digit. Only letters, digits, hyphens (-), underscores (_), and periods (.) are allowed.
   Description
   Enter a maximum of 512 characters.
   0/512

   – **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

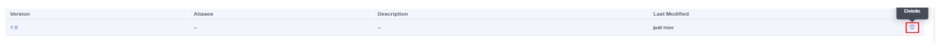
   - You can publish up to 10 versions for a function.
   - For a function whose latest version has been configured with reserved instances, the function configuration can be modified. By default, non-latest versions do not have reserved instances.
   - No disk is attached to a new version created based on latest. Environment variables cannot be set if no trigger has been bound to the version.

Deleting a Version

1. 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 of the latest version, select the version to delete.

Figure 5-14 Deleting a version



Version	Aliases	Description	Last Modified	
1.0	--	--	just now	

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.

5.10 Managing Aliases

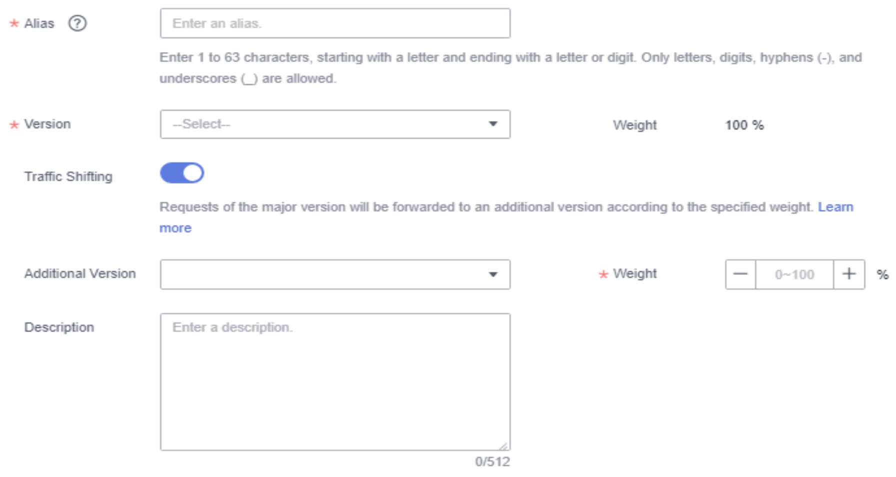
Overview

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.

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 5-15 Creating an alias


* Alias: Enter an alias.
Enter 1 to 63 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.
* Version: --Select-- Weight 100 %
Traffic Shifting
Requests of the major version will be forwarded to an additional version according to the specified weight. Learn more
Additional Version * Weight 0~100 %
Description: Enter a description.
0/512

– **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.
– **Weight**: Enter an integer from 0 to 100.
– **Description**: Enter a description for the alias.

4. Click **OK**.

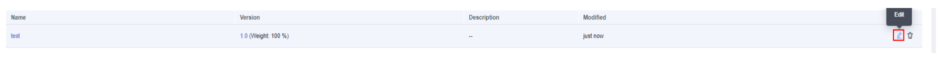
NOTE

You can create up to 10 aliases for a function.

Modifying 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 of the latest version, select the alias to modify.

Figure 5-16 Modifying an alias


Name	Version	Description	Modified
test	1.0 (Weight: 100 %)	--	just now

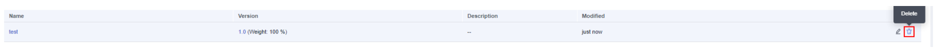
4. Modify the alias information, and click **OK**.

Deleting 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 of the latest version, select the alias to delete.

Figure 5-17 Deleting an alias



Name	Version	Description	Modified
test	1.0 (Weight: 100 %)	--	just now

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

5.11 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.

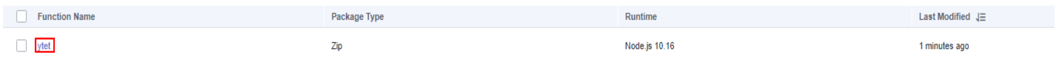
Prerequisites

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 5-18 Selecting a created function



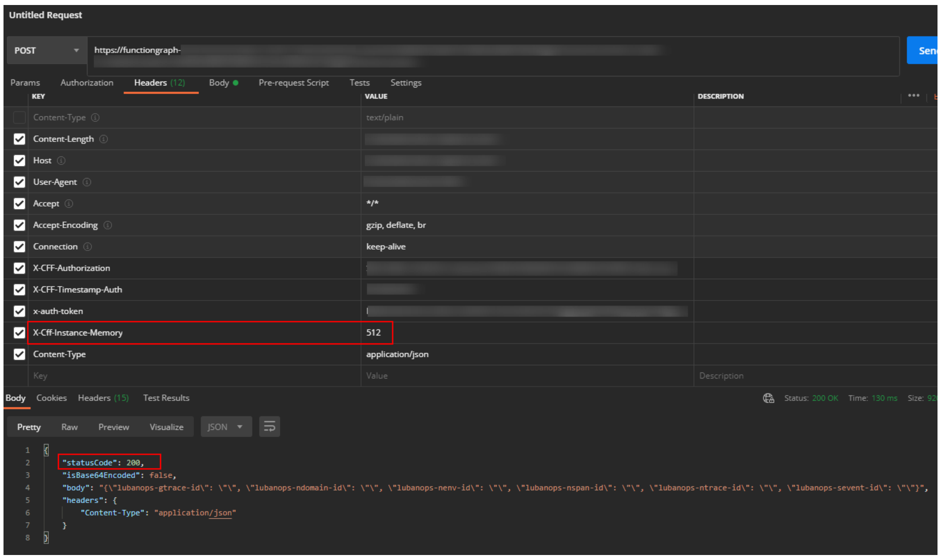
Function Name	Package Type	Runtime	Last Modified
ytet	Zip	Node.js 10.16	1 minutes ago

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**, or **4096**

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 5-19 Adding a request header and calling the function



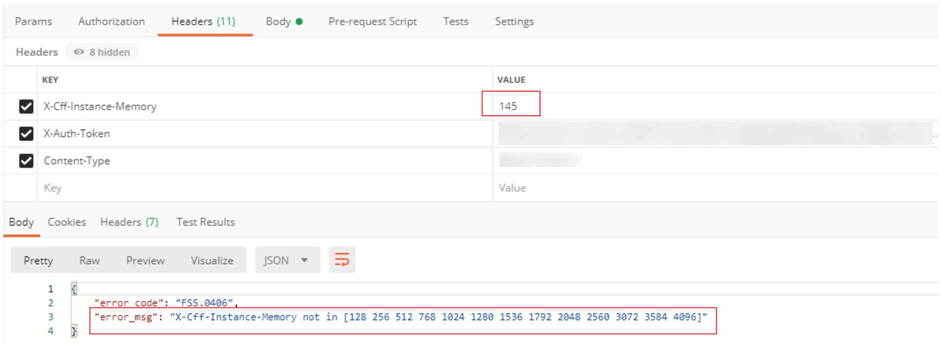
KEY	VALUE	DESCRIPTION
Content-Type	text/plain	
Content-Length		
Host		
User-Agent		
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
X-CFF-Authorization		
X-CFF-Timestamp-Auth		
x-auth-token		
X-Cff-Instance-Memory	512	
Content-Type	application/json	
Key	Value	Description

```
{
    "statusCode": 200,
    "isBase64Encoded": false,
    "body": "{\"lubanops-gtrace-id\": \"\", \"lubanops-ndomain-id\": \"\", \"lubanops-nenv-id\": \"\", \"lubanops-nspan-id\": \"\", \"lubanops-ntrace-id\": \"\", \"lubanops-sevent-id\": \"\"}",
    "headers": {
        "Content-Type": "application/json"
    }
}
```

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**, or **4096**, error code FSS.0406 will be returned when the API is called. You only need to reset the memory value.

Figure 5-20 Invocation failure



KEY	VALUE
X-Cff-Instance-Memory	145
X-Auth-Token	
Content-Type	
Key	Value

```
{
    "error_code": "FSS.0406",
    "error_msg": "X-Cff-Instance-Memory not in [128 256 512 768 1024 1280 1536 1792 2048 2560 3072 3584 4096]"
}
```

----End

6 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 6-1**. The parameter marked with an asterisk (*) is mandatory.

Table 6-1 Test event information

Parameter	Description
Configure Test Event	You can choose to create a test event or edit an existing one. Use the default option **Create new test event**.
Event Template	If you select **blank-template**, you can create a test event from scratch. If you select a template, the corresponding test event in the template is automatically loaded. For details about event templates, see **Table 6-2**.
*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 6-2 Event template description

Template Name	Description
blank-template	The template event is **{"key": "value"}**, which can be changed based on requirements.
apig-event-template	Simulates an API Gateway event to trigger your function.
dis-event-template	Simulates a DIS event to trigger your function.
smn-event-template	Simulates an SMN event to trigger your function.
obs-event-template	Simulates an OBS event to trigger your function.
timer-event-template	Simulates a timer event to trigger your function.
lts-event-template	Simulates an LTS event to trigger your function.
cts-event-template	Simulates a CTS event to trigger your function.
dds-event-template	Simulates a DDS event to trigger your function.
kafka-event-template	Simulates a Kafka event to trigger your function.
rabbitmq-event-template	Simulates a RabbitMQ event to trigger your function.
gaussmongo-event-template	Simulates a GaussDB(for Mongo) event to trigger your function.
login-security-template	Serves as an input for the **loginSecurity-realtime-analysis-python** function template.
porn-image-analysis	Serves as an input for the **porn-image-analysis** function template.
voice-analyse	Serves as an input for the **voice-analysis** function template.
image-tag	Serves as an input for the **image-tag** function template.

Step 5 Click **Save**.

----End

Testing a Function

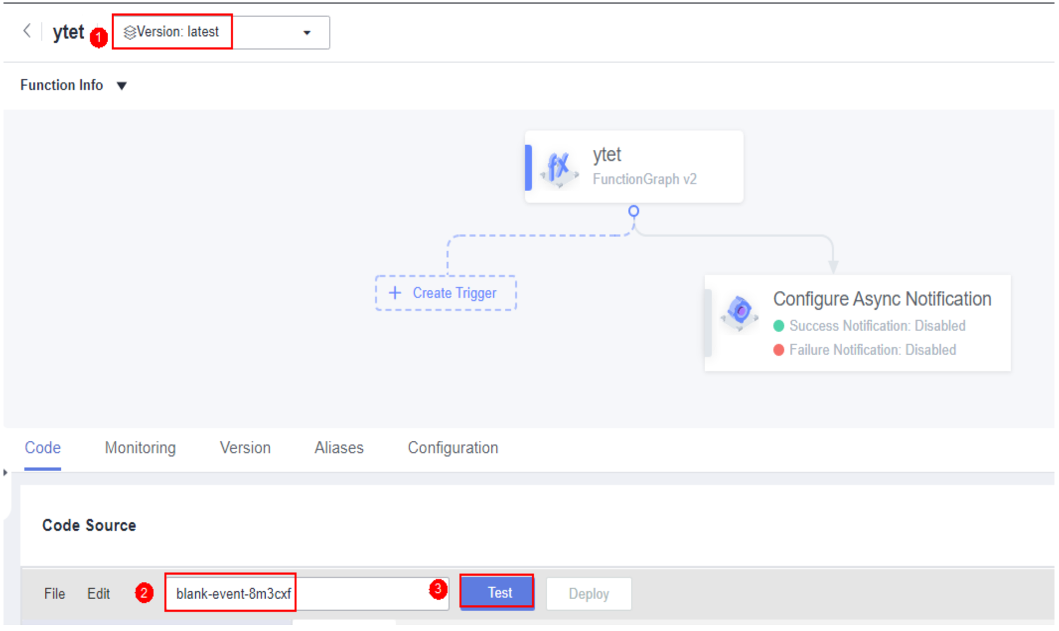
After creating a function, you can test it online to check whether it can run properly as expected.

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 version and test event, and click **Test**.

Figure 6-1 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 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, 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 6-3**.

Table 6-3 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 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, select a version and choose **Select test event** > **Configure test event**.

Step 4 In the **Configure Test Event** dialog box, select the test event you want to delete according to **Table 6-4**.

Table 6-4 Test event information

Parameter	Description
Configure Test Event	Select **Edit saved test event**.
Saved Test Event	Select the test event you want to delete.

Step 5 Click **Delete**.

----End

7 Creating Triggers

7.1 Managing Triggers

Enabling or Disabling a Trigger

You can enable or disable triggers as required. **Note that OBS and 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** > **Trigger**. 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 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** > **Trigger**. On the displayed page, locate the row that contains the target trigger and click **Delete**.

----End

7.2 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.

Prerequisites

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

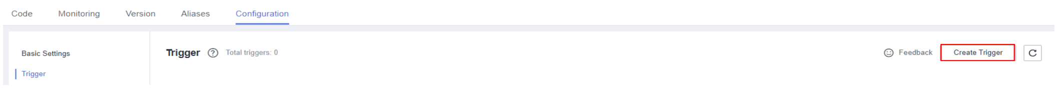
Creating a Timer Trigger

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 7-1 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 **Appendix: 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.

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 Log in 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

7.3 Using an APIG (Dedicated) Trigger

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

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**.

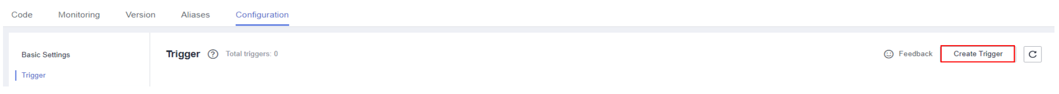
Step 4 Click **Create**.

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 7-2 Creating a trigger



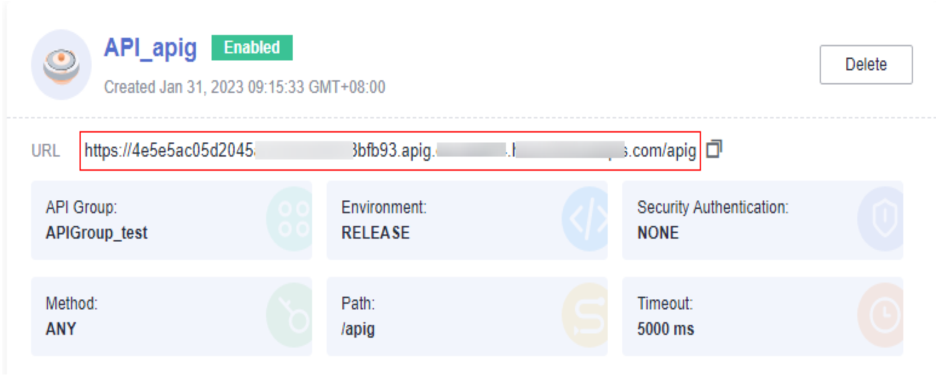
Step 7 Configure the trigger information.

Table 7-1 Trigger information

Parameter	Description
Trigger Type	Select **API Gateway (Dedicated Gateway)**.
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. Select **APIGroup_test**.
Environment	An API can be called in different environments, such as production, test, and development environments. API Gateway supports environment management, which allows you to define different request paths for an API in different environments. To ensure that the API can be called, select **RELEASE**.
Security Authentication	There are three authentication modes: ● **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. Select **None**.
Protocol	There are two types of protocols: ● HTTP ● HTTPS Select **HTTPS**.
Timeout (ms)	Enter **5000**.

Step 8 Click **OK**.

Figure 7-3 Creating a trigger


API_apig Enabled
Created Jan 31, 2023 09:15:33 GMT+08:00
Delete
URL https://4e5e5ac05d2045 3bfb93.apig. .com/apig
API Group: APIGroup_test
Environment: RELEASE
Security Authentication: NONE
Method: ANY
Path: /apig
Timeout: 5000 ms

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 API Gateway console. You can click the API name in the trigger list to go to the API Gateway console.

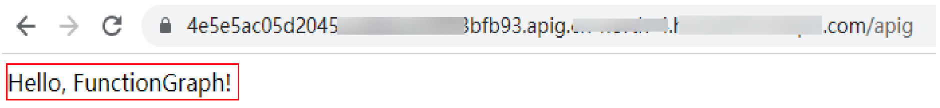
----End

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 7-4**.

Figure 7-4 Returned result


4e5e5ac05d2045 3bfb93.apig. .com/apig
Hello, FunctionGraph!

NOTE

1. The input for API Gateway invocation comes from an event template provided by the function. For details, see **Table 6-2**.
2. The function response for API Gateway invocation is encapsulated and must contain **body(String)**, **statusCode(int)**, **headers(Map)**, and **isBase64Encoded(boolean)**.

----End

Viewing the Execution Result

Step 1 Log in 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

7.4 Using an OBS Trigger

This section describes how to create an OBS trigger and upload an image package to a specified OBS bucket to trigger a function.

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 an OBS bucket, for example, **obs_cff**.

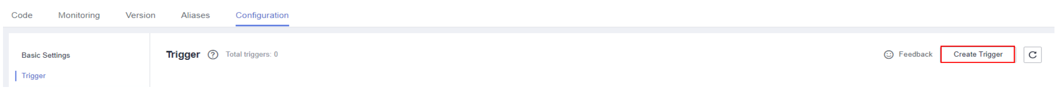
Creating an OBS Trigger

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

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

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

Figure 7-5 Creating a trigger



Step 4 Set the following parameters:

- **Trigger Type**: Select **Object Storage Service (OBS)**.
- **Bucket Name**: Specify the OBS bucket to be used as an event source, for example, **obs-cff**.
- **Events**: Select events that will trigger the function. In this example, select **Put**, **Post**, and **Delete**. When files in the **obs_cff** bucket are updated, uploaded, or deleted, the function is triggered.
- **Event Notification Name**: Specify the name of the event notification to be sent by SMN when an event occurs.
- **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.

Step 5 Click **OK**.

----End

Triggering a Function

On the OBS console, upload an image ZIP package to the **obs-cff** bucket.

NOTE

After the ZIP package is uploaded to the **obs-cff** bucket, the **HelloWorld** function is triggered.

Viewing the Execution Result

Step 1 Log in 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

7.5 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.

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 "Creating an Instance" in the *Distributed Message Service for Kafka User Guide*.
- You have created a topic under a Kafka instance. For details, see section "Creating a Topic" in the *Distributed Message Service for Kafka User Guide*.

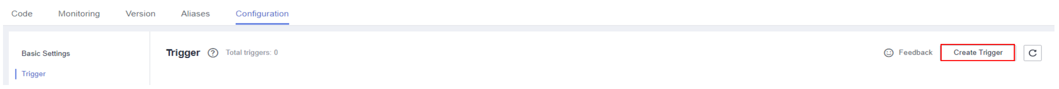
Creating a Kafka Trigger

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 7-6 Creating a trigger



Step 4 Set the following parameters:

- **Trigger Type**: Select **Distributed Message Service for Kafka (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**.

----End

Configuring a Kafka Event to Trigger the 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 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 7-2** and click **Save**.

Table 7-2 Test event information

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

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

----End

7.6 Using an LTS Trigger

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

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**.

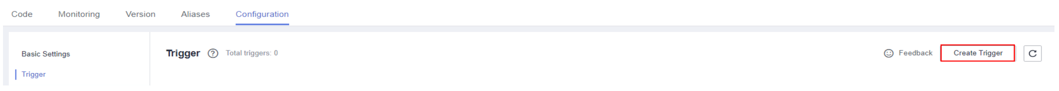
Creating an LTS Trigger

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 7-7 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 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 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 7-3** and click **Save**.

Table 7-3 Test event information

Parameter	Description
Configure Test Event	You can choose to create a test event or edit an existing one. Use the default option **Create new test event**.
Event Template	Select **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

7.7 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.

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**.

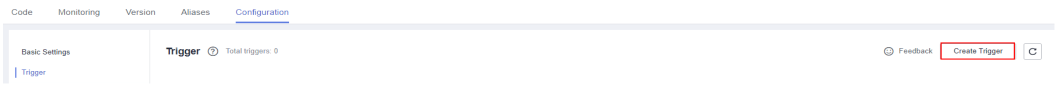
Step 4 Click **Create Function**.

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 7-8 Creating a trigger



Step 7 Configure the trigger information.

Table 7-4 Trigger information

Parameter	Description
Trigger Type	Select **Cloud Trace Service (CTS)**.
Notification Name	Enter a notification name, for example, **Test**.
Service Type	Select **FunctionGraph**.
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 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 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 7-5** and click **Save**.

Table 7-5 Test event information

Parameter	Description
Configure Test Event	You can choose to create a test event or edit an existing one. Use the default option **Create new test event**.
Event Template	Select **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

7.8 Appendix: 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 7-6**.

Table 7-6 Example expressions

Expression	Meaning
@every 30m	Triggers a function every 30 minutes.

Expression	Meaning
@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 7-7** describes the fields in a standard cron expression.

Table 7-7 Parameter description

Parameter	Description	Value Range	Special Characters Allowed
Seconds	Yes	0-59	, - * /
Minutes	Yes	0-59	, - * /
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 7-8**.	, - * /
Day-of-week	No	0–6 or Sun–Sat. The value is case-insensitive, as shown in **Table 7-9**. **0** means Sunday.	, - * ? /

Table 7-8 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

Month	Digit	Abbreviation
July	7	Jul
August	8	Aug
September	9	Sep
October	10	Oct
November	11	Nov
December	12	Dec

Table 7-9 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 7-10 describes the special characters that can be used in a cron expression.

Table 7-10 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.

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

Table 7-11 describes several example cron expressions.

Table 7-11 Example cron expressions

Cron Expression	Example
0 15 2 * * ?	Executes a function at 02:15:00 every day.
0 30 8 ? * Mon-Fri	Executes a function at 08:30:00 every Monday through Friday.
0 45 7 1-3 * ?	Executes a function at 07:45:00 on the first three days of every month.
0 0/3 * ? * Mon,Wed,Fri,Sun	Executes a function every 3 minutes on every Monday, Wednesday, Friday, and Sunday.
0 0/3 9-18 ? * Mon-Fri	Executes a function every 3 minutes during 09:00–18:00 every Monday through Friday.
0 0/30 * * * ?	Executes a function every 30 minutes.

8 Invoking the Function

8.1 Synchronous Invocation

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

8.2 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 8-1 Invocation mode

Event Source	Invocation Mode
SMN	Asynchronous
OBS	Asynchronous
DIS	Asynchronous
Timer	Asynchronous
LTS	Asynchronous
CTS	Asynchronous
DMS for Kafka	Asynchronous

- APIG and APIG (dedicated) 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

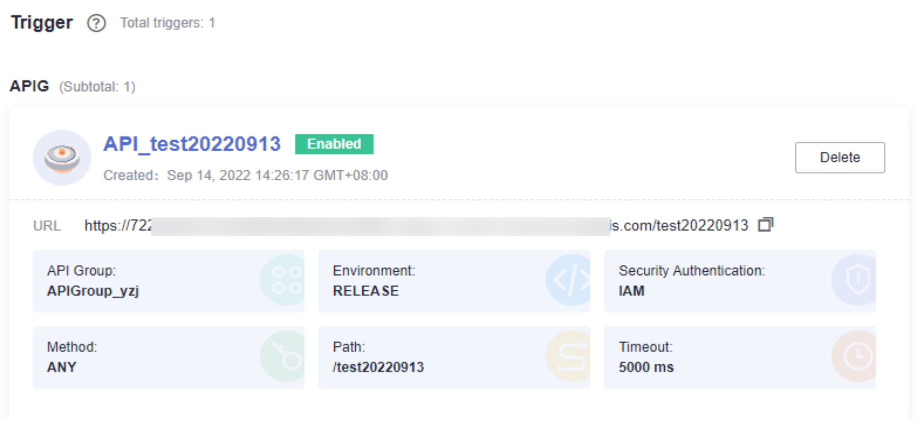
  If the E2E function execution latency exceeds 90s, asynchronous invocation is recommended. If synchronous invocation is used, no responses can be received after 90s due to gateway restrictions.

Example

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

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

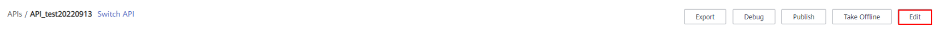
Figure 8-1 Clicking a trigger name



API_test20220913 Enabled
Created: Sep 14, 2022 14:26:17 GMT+08:00
URL https://72 [redacted] s.com/test20220913
API Group: APIGroup_yzj
Environment: RELEASE
Security Authentication: IAM
Method: ANY
Path: /test20220913
Timeout: 5000 ms

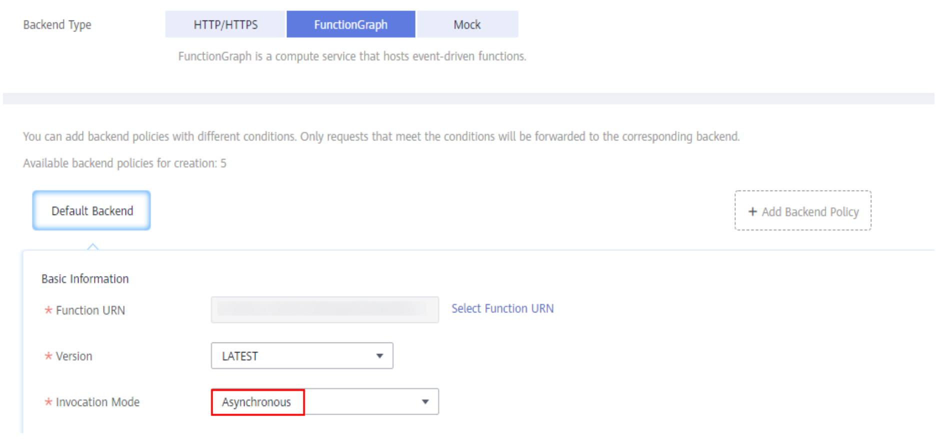
3. Click **Edit** in the upper right.

Figure 8-2 Clicking Edit



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

Figure 8-3 Changing the invocation mode



FunctionGraph is a compute service that hosts event-driven functions.

You can add backend policies with different conditions. Only requests that meet the conditions will be forwarded to the corresponding backend.

Available backend policies for creation: 5

* Function URN: [redacted]
* Version: LATEST
* Invocation Mode: Asynchronous

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

8.3 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.

9 Monitoring

9.1 Metrics

9.1.1 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 9-1 describes the function metrics.

Table 9-1 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. **Minimum Duration**: the minimum duration a function is executed within a period. **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.

9.1.2 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 9-2 Function metrics

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period of Raw Data (Minute)
count	Invocations	Number of function invocations Unit: Count	≥ 0 counts	Functions	1
failcount	Errors	Number of invocation errors The following errors are included: ● Function request error (causing an execution failure and returning error code 200) ● Function syntax or execution error Unit: Count	≥ 0 counts	Functions	1
rejectcount	Throttles	Number of function throttles That is, the number of times that FunctionGraph throttles your functions due to the resource limit. Unit: Count	≥ 0 counts	Functions	1
concurrency	Number of concurrent requests	Maximum number of concurrent requests during function invocation. Unit: Count	≥ 0 counts	Functions	1

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period of Raw Data (Minute)
reservedinstancenum	Number of reserved instances	Number of reserved instances Unit: Count	≥ 0 counts	Functions	1
duration	Average duration	Average duration of function invocation Unit: ms	≥ 0 ms	Functions	1
maxDuration	Maximum duration	Maximum duration of function invocation Unit: ms	≥ 0 ms	Functions	1
minDuration	Minimum duration	Minimum duration of function invocation Unit: ms	≥ 0 ms	Functions	1

Dimensions

Key	Value
package-functionname	*App name-Function name* Example: default-myfunction_Python

9.1.3 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**.

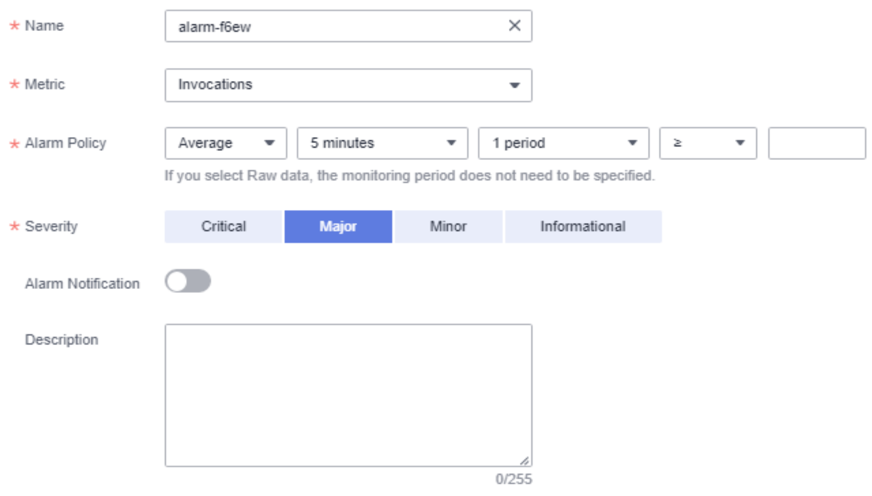
Step 2 Click the name of the desired function.

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 9-1**.

Figure 9-1 Creating an alarm rule



* Name: alarm-f6ew

* Metric: Invocations

* Alarm Policy: Average | 5 minutes | 1 period | ≥ |

If you select Raw data, the monitoring period does not need to be specified.

* Severity: Critical [✓ Major] Minor Informational

Alarm Notification

Description

0/255

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

----End

Function Metrics

Table 9-3 lists the function metrics that can be monitored by Cloud Eye.

Table 9-3 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

Metric	Display Name	Description	Unit	Upper Limit	Lower Limit	Recommended Threshold	Value Type	Dimension
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	-	int	package-functionname
maxDuration	Maximum Duration	Maximum duration of function invocation	ms	-	0	-	int	package-functionname
minDuration	Minimum Duration	Minimum duration of function invocation	ms	-	0	-	int	package-functionname

9.2 Logs

9.2.1 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**.

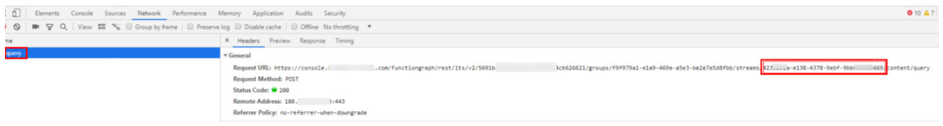
9.2.2 Managing Function Logs

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.

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.



▾ General
Request URL: https://console.[redacted].com/functiongraph/rest/lts/v2/5691b[redacted]cb626621/groups/f9f979a1-e1a9-469e-a5e3-be2e7e5d8fbb/streams/423[redacted]a-e138-4370-9ebf-9bb[redacted]669/content/query
Request Method: POST
Status Code: 200
Remote Address: 100.[redacted]:443
Referrer Policy: no-referrer-when-downgrade

- 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.

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 9-4 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. 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 9-5 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

10 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.

----End

Disabling a Function

Disabled functions can no longer be executed.

Step 1 Log in 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** 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 Log in 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** in the upper right corner.

----End

Deleting a function

You can delete unused functions to release resources.

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

Step 2 Click the name of the function you want to delete.

Step 3 Select a search criterion (function name, runtime, or app name) in the upper right corner, enter a keyword, and click 🔍 to search for the function you want to delete.

Step 4 Click **Delete** in the same row as the function.

Step 5 In the **Delete Function** dialog box, click **Yes** to confirm the deletion.

----End

11 Dependency Management

Introduction

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 11-1 Dependency configuration parameters

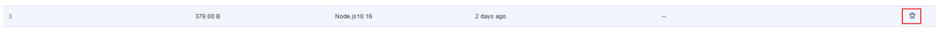
Parameter	Description
Name	Dependency name.

Parameter	Description
Upload Mode	Upload a ZIP file directly or upload a file from OBS. ● **Upload ZIP file**: Click **Select File** to upload one. ● **Upload file from OBS**: Specify an OBS link 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.



3	379.00 B	Node.js10.16	2 days ago	--

----End

Configuring Dependencies for a Function

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

Step 2 Click the name of the desired function.

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

Step 4 On the displayed **Select Dependency** dialog box, select dependencies and click **OK**.

Table 11-2 Dependency configuration

Parameter	Description
Runtime	Runtime of this function. It cannot be changed.
Type	Add a **Public** or **Private** dependency.
Name	Select a dependency.
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

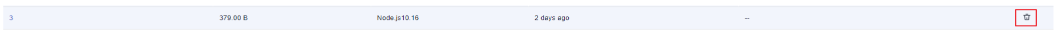
To delete a dependency, just delete all of its versions.

Step 1 Log in 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 11-1 Deleting a dependency version



3 379.00 B Node.js10.16 2 days ago --

NOTE

Dependencies referenced by functions cannot be deleted.

----End

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 11-3** and **Table 11-4**. You can import these libraries to your inline code in the same way as you import standard libraries.

Table 11-3 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

Name	Usage	Version
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 API Gateway 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 11-4 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	HTTP client	0.10.3
numpy	Mathematical computing	1.13.1
redis	Redis client	2.10.5
obsclient	OBS client	-
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. For details, see **Creating a Dependency**. These libraries will then be used in your function code.

Importing Dependent Libraries

The code for processing images is as follows:

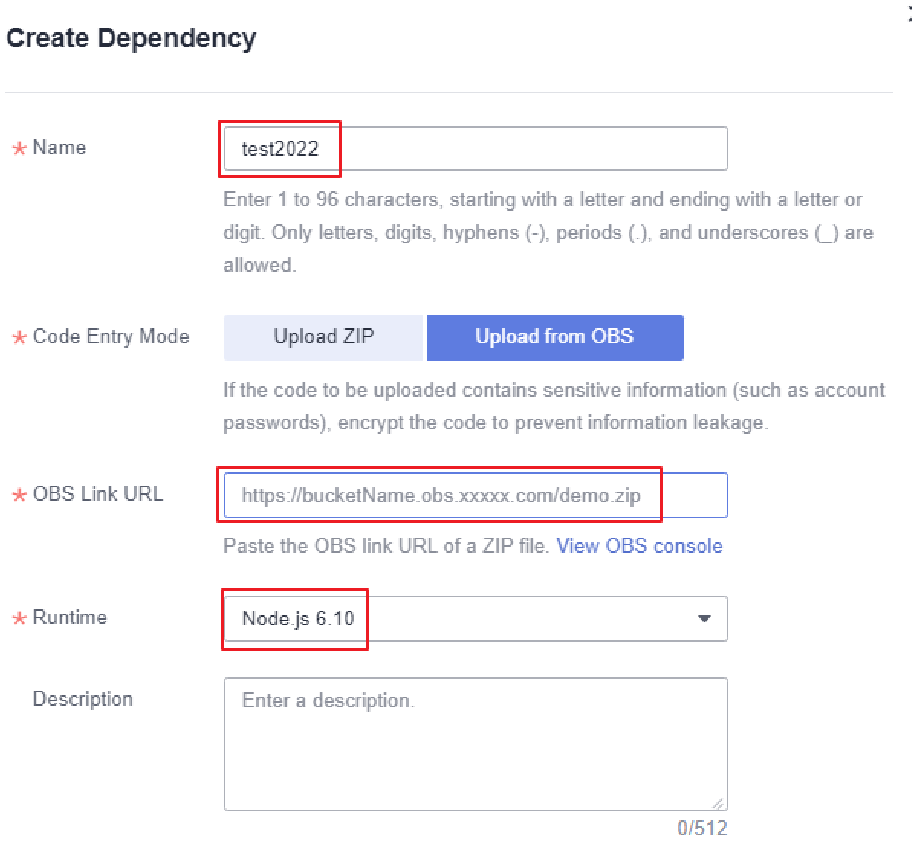
```
# -*- coding: utf-8 -*-
from PIL import Image, ImageEnhance

from com.obs.client.obs_client import ObsClient

import sys
import os

current_file_path = os.path.dirname(os.path.realpath(__file__))
# append current path to search paths, so that we can import some third party libraries.
sys.path.append(current_file_path)
region = 'your region'
obs_server = 'obs.xxxxxxcloud.com'
def newObsClient(context):
    ak = context.getAccessKey()
    sk = context.getSecretKey()
    return ObsClient(access_key_id=ak, secret_access_key=sk, server=obs_server,
                     path_style=True, region=region, ssl_verify=False, max_retry_count=5, timeout=20)
def downloadFile(obsClient, bucket, objName, localFile):
    resp = obsClient.getObject(bucket, objName, localFile)
    if resp.status < 300:
        print 'download file', file, 'succeed'
    else:
        print('download failed, errorCode: %s, errorMessage: %s, requestId: %s' % resp.errorCode,
resp.errorMessage,
              resp.requestId)
def uploadFileToObs(client, bucket, objName, file):
    resp = client.putFile(bucket, objName, file)
    if resp.status < 300:
        print 'upload file', file, 'succeed'
    else:
        print('upload failed, errorCode: %s, errorMessage: %s, requestId: %s' % resp.errorCode,
resp.errorMessage,
              resp.requestId)
def getObjInfoFromObsEvent(event):
    s3 = event['Records'][0]['s3']
    eventName = event['Records'][0]['eventName']
    bucket = s3['bucket']['name']
    objName = s3['object']['key']
    print "*** obsEventName: %s, srcBucketName: %s, objName: %s", eventName, bucket,
objName
    return bucket, objName
def set_opacity(im, opacity):
    """Set the transparency."""
    if im.mode != "RGBA":
        im = im.convert('RGBA')
    else:
        im = im.copy()
    alpha = im.split()[3]
    alpha = ImageEnhance.Brightness(alpha).enhance(opacity)
    im.putalpha(alpha)
    return im
def watermark(im, mark, opacity=0.6):
    """Add a watermark."""
    try:
        if opacity < 1:
            mark = set_opacity(mark, opacity)
        if im.mode != 'RGBA':
            im = im.convert('RGBA')
        if im.size[0] < mark.size[0] or im.size[1] < mark.size[1]:
            print "The mark image size is larger size than original image file."
            return False
        x = (im.size[0] - mark.size[0]) / 2
        y = (im.size[1] - mark.size[1]) / 2
        layer = Image.new('RGBA', im.size, )
        layer.paste(mark, (x, y))
        return Image.composite(layer, im, layer)
    except Exception as e:
        print ">>>>>>>>>>> WaterMark EXCEPTION:  " + str(e)
        return False
def watermark_image(localFile, fileName):
    im = Image.open(localFile)
    watermark_image_path = os.path.join(current_file_path, "watermark.png")
    mark = Image.open(watermark_image_path)
    out = watermark(im, mark)
    print "**********finish water mark"
    name = fileName.split('.')
    outFileName = name[0] + '-watermark.' + name[1]
    outFilePath = "/tmp/" + outFileName
    if out:
        out = out.convert('RGB')
        out.save(outFilePath)
    else:
        print "Sorry, Save watermarked file Failed."
    return outFileName, outFilePath
def handler(event, context):
    srcBucket, srcObjName = getObjInfoFromObsEvent(event)
    outputBucket = context.getUserData('obs_output_bucket')
    client = newObsClient(context)
    # download file uploaded by user from obs
    localFile = "/tmp/" + srcObjName
    downloadFile(client, srcBucket, srcObjName, localFile)
    outFileName, outFile = watermark_image(localFile, srcObjName)
    # Upload converted files to a new OBS bucket.
    uploadFileToObs(client, outputBucket, outFileName, outFile)
    return 'OK'
```

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**.

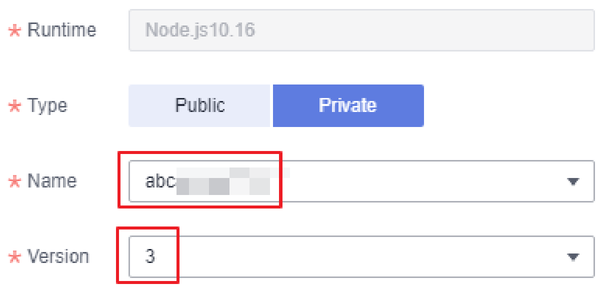
Figure 11-2 Setting the dependency


Create Dependency

* Name	test2022
	Enter 1 to 96 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), periods (.), and underscores (_) are allowed.
* Code Entry Mode	Upload ZIP / Upload from OBS
	If the code to be uploaded contains sensitive information (such as account passwords), encrypt the code to prevent information leakage.
* OBS Link URL	https://bucketName.obs.xxxxx.com/demo.zip
	Paste the OBS link URL of a ZIP file. View OBS console
* Runtime	Node.js 6.10
Description	Enter a description.
	0/512

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 11-3 Selecting a dependency



* Runtime	Node.js10.16
* Type	Public / Private
* Name	abc
* Version	3

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.

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** and intelligent recommendation policies.

  NOTE

  By default, you do not have the permissions to use the reserved instance function. To use this function, submit a service ticket to add your account to the whitelist.

Configuring a Fixed Number of Reserved Instances

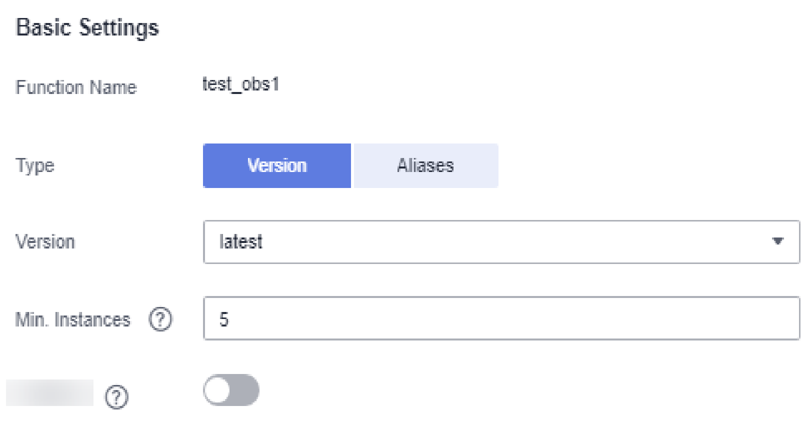
Ensure that the function for which you want to create reserved instances already exists on the FunctionGraph console.

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.



Basic Settings

Function Name	test_obs1
Type	Version / Aliases
Version	latest
Min. Instances	5

Figure 12-2 Basic settings

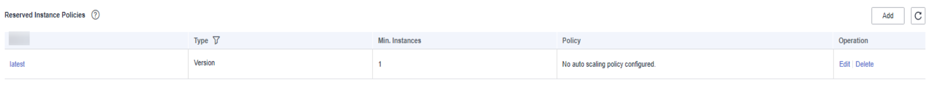
Table 12-1 Basic settings

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**.
Min. 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.

5. Click **OK**. The new policy is displayed in the reserved instance policy list.



Reserved Instance Policies

	Type	Min. Instances	Policy	Operation
latest	Version	1	No auto scaling policy configured.	Edit | Delete

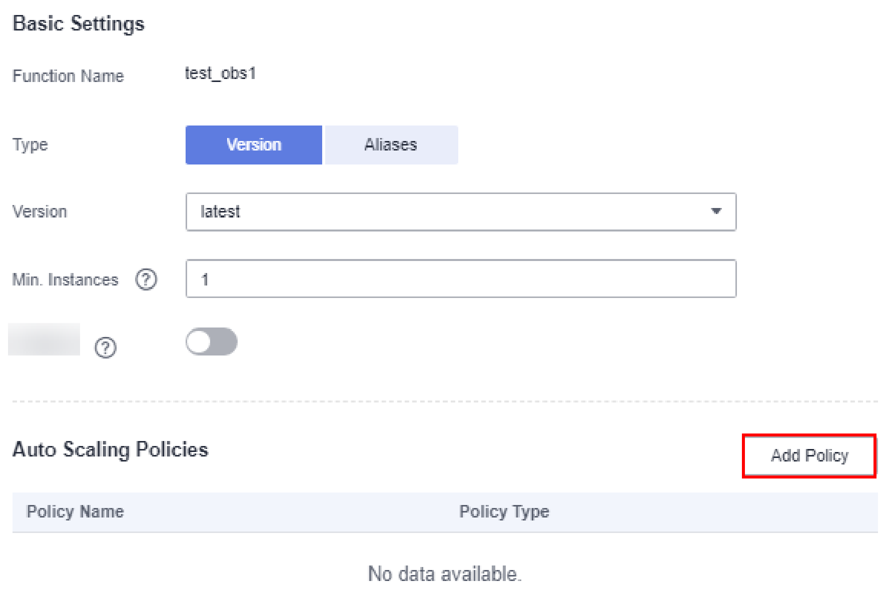
Figure 12-3 Policy list

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



Basic Settings

Function Name: test_obs1

Type: Version | Aliases

Version: latest

Min. Instances: 1

Auto Scaling Policies

Add Policy

Policy Name	Policy Type

No data available.

2. Set parameters by referring to **Table 12-2**.

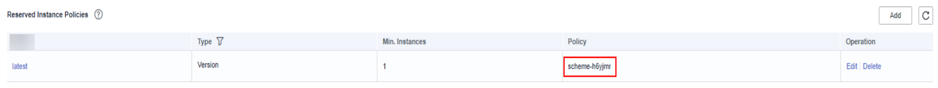
Table 12-2 Scheduled scheduling policy parameters

Parameter	Description
Policy Name	Policy name.
Cron Expression (UTC)	Set this parameter by referring to **Appendix: Cron Expressions for a Function Timer Trigger**.
Validity	Local time when the cron expression is effective. The scheduled scaling policy is effective only during this validity period. In other time, the **Min. Instances** in the basic settings is used.

Parameter	Description
Min. Instances	The number of reserved instances to be created when the policy is effective. Set a number that meets your service requirements. **NOTE** 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 12-5 Policy list



Reserved Instance Policies

	Type	Min. Instances	Policy	Operation
latest	Version	1	scheme-h6yjmr	Edit | Delete

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.

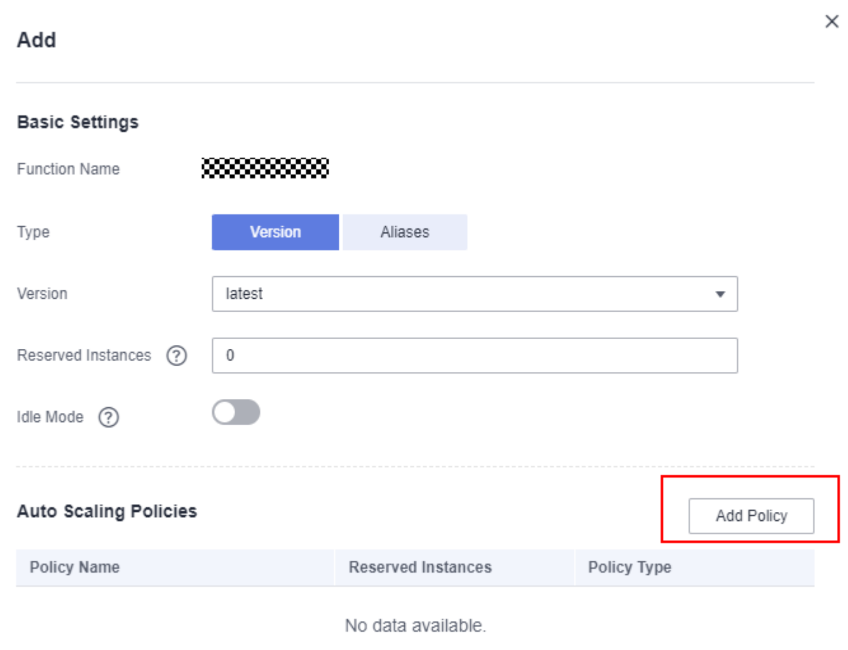
Configuring an Intelligent Recommendation Policy

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 12-6 Clicking Add Policy


Add

Basic Settings

Function Name [redacted]

Type: Version | Aliases

Version: latest

Reserved Instances: 0

Idle Mode

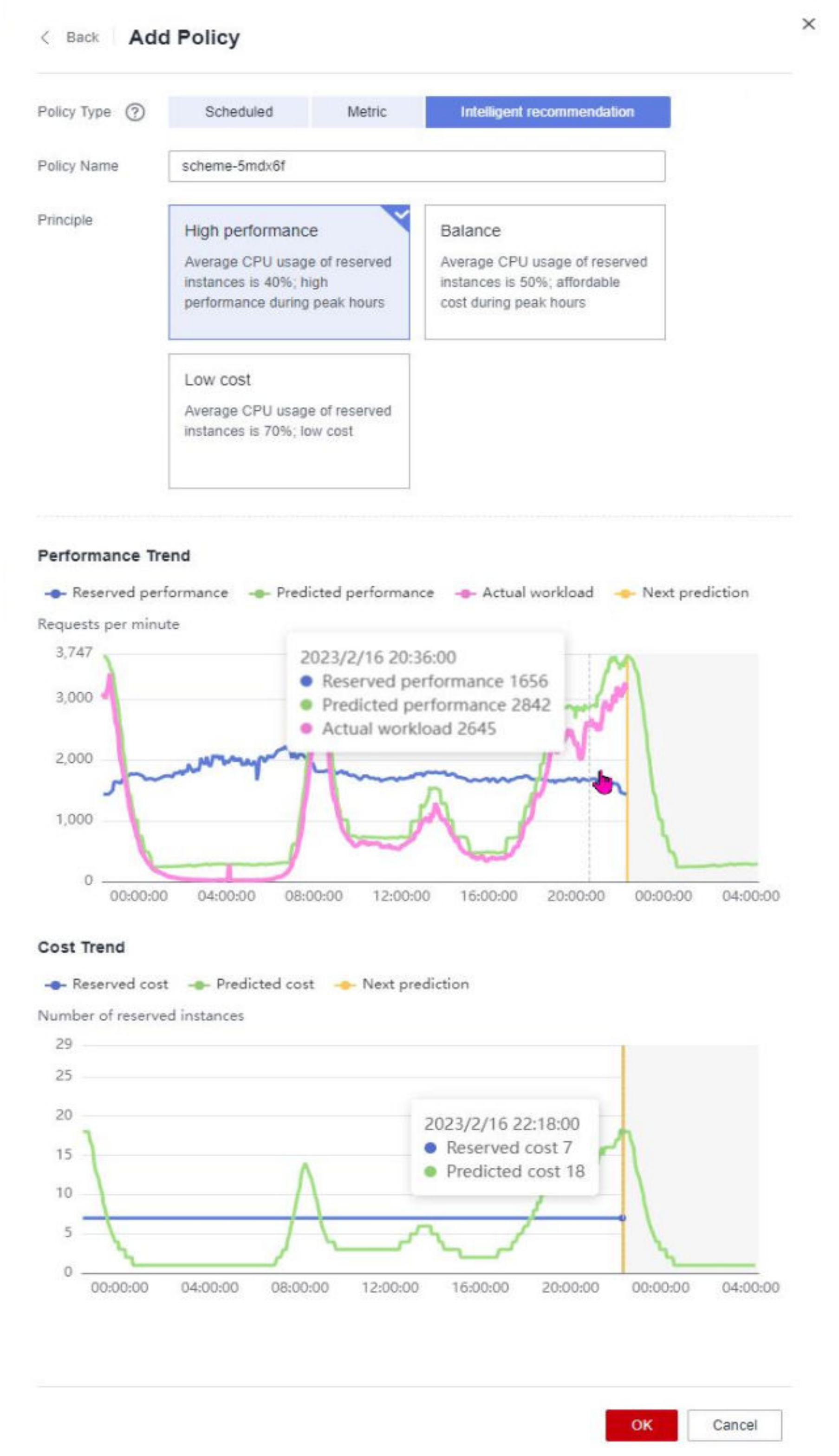
Auto Scaling Policies

Policy Name	Reserved Instances	Policy Type

No data available.

2. Select **Intelligent recommendation**, and select any of **High performance**, **Balance**, and **Low cost** while referring to the performance and cost trends.

Figure 12-7 Intelligent recommendation



Add Policy

Policy Type: Scheduled | Metric | Intelligent recommendation

Policy Name: scheme-5mdx6f

Principle:

- High performance: Average CPU usage of reserved instances is 40%; high performance during peak hours
- Balance: Average CPU usage of reserved instances is 50%; affordable cost during peak hours
- Low cost: Average CPU usage of reserved instances is 70%; low cost

Performance Trend

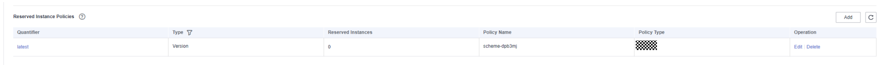
2023/2/16 20:36:00
Reserved performance 1656
Predicted performance 2842
Actual workload 2645

Cost Trend

2023/2/16 22:18:00
Reserved cost 7
Predicted cost 18

3. Click **OK**. The new policy is displayed in the reserved instance policy list.

Figure 12-8 Reserved Instance Policies



Reserved Instance Policies

Add

Quantifier	Type	Reserved Instances	Policy Name	Policy Type	Operation
latest	Version	0	scheme-dpb3mj	[redacted]	Edit | Delete

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 Audit

13.1 Operations Logged by CTS

Table 13-1 lists the FunctionGraph operations that can be logged by CTS.

Table 13-1 Operations logged by CTS

Operation	Resource Type	Event Name
Creating a function	Function	createFunction
Deleting a function	Function	deleteFunction
Modifying function information	Function	updateFunctionConfig
Publishing a function version	Function version	publishFunctionVersion
Deleting a function version alias	Function version alias	deleteVersionAlias
Deleting a function trigger	Trigger	deleteTrigger
Creating a function trigger	Trigger	createTrigger
Disabling a function trigger	Trigger	disabledTrigger
Enabling a function trigger	Trigger	enabledTrigger

13.2 Viewing Audit Logs

Scenario

After Cloud Trace Service (CTS) is enabled, the tracker starts recording operations on cloud resources and data in OBS buckets. CTS keeps operation records of the latest 7 days.

This section describes how to query and export operation records of the last seven days on the CTS console.

Procedure

1. Log in to the management console.
2. Click **Service List** and choose **Management & Governance** > **Cloud Trace Service**.
3. In the navigation pane, click **Trace List**.
4. Specify the search criteria as needed.
   - Time range: Select a time range in the upper right.
   - **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 **Data** for **Trace Type**, filter traces by tracker.
   - **Operator**: Select one or more operators from the drop-down list.
   - **Trace Status**: Select **All trace statuses**, **Normal**, **Warning**, or **Incident**.
5. Click **Query**.
6. Click **Export** on the right to export all traces in the query result as a CSV file. The file can contain up to 5,000 records.
7. Click ⌄ on the left of a trace to expand its details.
8. Click **View Trace** in the **Operation** column. The trace details are displayed.

   For details about key fields in the trace structure, see section "Trace References" in the *Cloud Trace Service User Guide*.

14 FAQs

14.1 General FAQs

14.1.1 What Is FunctionGraph?

FunctionGraph allows you to run you 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.

14.1.2 Do I Need to Apply for Any Compute, Storage, or Network Services When Using FunctionGraph?

When using FunctionGraph, you do not need to apply for or pre-configure any computing, storage, or network services, but need to upload and run code in supported runtimes. FunctionGraph provides and manages underlying compute resources, including server CPUs, memory, and networks. It performs configuration and resource maintenance, code deployment, automatic scaling, load balancing, secure upgrade, and resource monitoring.

14.1.3 Do I Need to Deploy My Code After Programming?

After programming, you only need to 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 be 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 plugin name of the handler. For example, if the name of the dynamic library file is **testplugin.so**, set the handler to **testplugin.Handler**.

14.1.4 What Runtimes Does FunctionGraph Support?

Table 14-1 lists the runtimes supported by FunctionGraph.

Table 14-1 Supported runtimes and versions

Runtime	Version
Python	2.7, 3.6, 3.9
Node.js	6.10, 8.10, 10.16, 12.13, 14.18, 16.17, 18.15
Java	8 and 11
Go	1.x

14.1.5 How Much Disk Space Is Allocated to Each FunctionGraph Function?

Each FunctionGraph function is allocated 512 MB ephemeral disk space. You can upload deployment packages up to 10 GB in size. For more information, see **Notes and Constraints**.

14.1.6 Does FunctionGraph Support Function Versioning?

Yes. For details, see **Managing Versions**.

14.1.7 How Does a Function Read or Write Files?

Background

A function can read files in the code directory. The working directory of a function is the upper-level directory of the handler file. Assume that you have uploaded a folder named **backend**. To read its **test.conf** file in the same level of directory as the handler file, use relative path **code/backend/test.conf** or use a full path (that is, the value of the **RUNTIME_CODE_ROOT** environment variable). To write a file (for example, to create or download a file), go to the **/tmp** directory or use the file system mounting feature provided by FunctionGraph.

NOTE

- If containers are reclaimed, file read/written content will become invalid.
- Currently, FunctionGraph does not support instance persistence.

Typical Scenarios

- Download files stored in Object Storage Service (OBS) to the **/tmp** directory for processing.
- To store function execution data in OBS, create a file in the **/tmp** directory, write the data into the file, and then upload the file to OBS.

14.1.8 Does FunctionGraph Support Function Extension?

FunctionGraph has integrated non-standard libraries such as redis, http, and obs_client. You can directly use these libraries when developing functions.

Alternatively, use your own dependencies. For more information, see **Dependency Management**.

14.1.9 Which Permissions Are Required for an IAM User to Use FunctionGraph?

When logging in to FunctionGraph and adding, deleting, modifying, and querying a function and its triggers as an IAM user, grant permissions to the user group to which the IAM user belongs as required. For example, to create an OBS bucket and trigger, grant the **Tenant Administrator** permission for OBS. Manage permissions on a principle of least permissions (PoLP) basis.

14.1.10 How Can I Create an ODBC Drive-based Python Dependency Package for Database Query?

For OS-dependent packages (for example, unixODBC), download the source code to compile dependency packages.

1. Log in to your ECS on the ECS console (ensure that the GCC and Make tools have been installed), and run the following command to download the source code package:
   ```
   wget source code path
   ```
   If you downloaded a **.zip** file, run the following command to decompress it:
   ```
   unzip xxx/xx.zip
   ```
   If you downloaded a **tar.gz** file, run the following command to decompress it:
   ```
   tar -zxvf xxx/xx.tar.gz
   ```
2. Run the following command to create the **/opt/function/code** directory:
   ```
   mkdir /opt/function/code
   ```
3. Go to the destination directory and run the following command:
   ```
   ./configure --prefix=/opt/function/code --sysconfdir=/opt/function/code;make;make install
   ```
4. Go to **/opt/function/code/lib/pkgconfig** and check whether the prefix directory is **/opt/function/code**.
   ```
   cd /opt/function/code/lib/pkgconfig
   ```
5. Copy all files in **/opt/function/code/lib** to **/opt/function/code**.
   ```
   cp -r /opt/function/code/lib/* /opt/function/code
   ```
6. Switch to **/opt/function/code** and compress all files in it to a **.zip** package.
   ```
   cd /opt/function/code
   zip -r xxx.zip *
   ```

14.1.11 What Is the Quota of FunctionGraph?

For details about the resource quota of FunctionGraph, see **Notes and Constraints**.

14.1.12 How Does a Container Image–based Function Resolve a Private DNS Domain Name?

FunctionGraph functions created with a container image cannot directly parse private Domain Name Service (DNS) domain names. However, you can call DNS APIs to achieve this purpose.

Resolving a Private DNS Domain Name

1. Obtain a private domain name and zone ID.

   This procedure uses a domain name with a record set as an example.

   a. Log in to the DNS console.

   b. Obtain a zone ID.

      Click ⚙, and select **Domain Name** in the search box to obtain a zone ID.

   c. Obtain the private domain name corresponding to a recording set.

      Click the domain name to go to the record set list, and select a record set.

2. Compile the resolution logic.

   Debug the API used to **query record sets in a zone**.

   – Set **zone_id** to the zone ID obtained in the preceding step, and click **Debug**. The IP address of the private domain name is displayed in the response body.

   – Switch to the **Sample Code** tab to obtain the complete code. For details about the dependencies, click **View SDK Details**.

14.1.13 How Do I Use a Domain Name to Access an API Registered with API Gateway (Dedicated)?

The domain name **www.test.com** is used as an example. The procedure is as follows:

Step 1 Log in to the API Gateway console, choose **Dedicated Gateways** in the navigation pane, and click the target gateway name. On the **Gateway Information** page, view **EIP** in the **Inbound Access** area to obtain the IP address of the API gateway.

Step 2 On the DNS console, configure an IPv4 rule for mapping **www.test.com** to an API gateway address.

Step 3 Configure domain name resolution. In this way, you can access the API registered with the API gateway by using domain name **www.test.com**.

----End

14.1.14 What Are the Common Application Scenarios of FunctionGraph?

1. Web applications: mini programs, web pages/apps, chatbots, and Backends for Frontends (BFF).
2. Event-driven applications: file processing, image processing, live video streaming/transcoding, real-time data stream processing, and IoT rule/event processing.
3. AI applications: third-party service integration, AI inference, and license plate recognition.

14.1.15 Why Can't the API Gateway Domain Name Bound to a Service Be Resolved During Function Invocation?

Currently, FunctionGraph resolves only DNS domain names and POD domain names.

14.1.16 Does FunctionGraph Support Synchronous Transmission at the Maximum Intranet Bandwidth?

Not currently.

14.1.17 What If the VPC Quota Is Used Up?

A tenant can create up to 4 VPCs. To create more VPCs, submit a service ticket.

14.1.18 How Can I Print Info, Error, or Warn Logs?

Take Java as an example. You can use this demo to print logs.

14.1.19 Can I Set the Domain Name of an API to My Own Domain Name?

Yes. The procedure is as follows:

Step 1 Log in to the APIG console and bind a domain name by referring to section "Binding a Domain Name" in the *API Gateway User Guide*.

Step 2 On the **Domain Names** tab page of the created API group, click **Bind Independent Domain Name**. For example, set **xxxx.apig.x** to **test.com/user/get**.

----End

14.1.20 Can I Change the Runtime?

No. Once a function is created, its runtime cannot be changed.

14.1.21 Can I Change a Function's Name?

No. A function's name cannot be changed once the function is created.

14.1.22 Why Is Message "failed to mount exist system path" Displayed?

When you see this message, mount the file to a new path.

User ID/user group ID: Can be any number except 1000. The value **–1** will be automatically converted to **1003**. The two IDs control the directory permissions for accessing a remote file system.

File system/ECS name: Name of the file system or ECS to create. Ensure that you have specified a VPC and agency that you have been authorized to access.

Shared directory: To configure a remote shared directory for the mounted ECS, see **Creating an NFS Shared Directory on an ECS**.

Access path: Location where the file system is to be mounted in the function. Set a new two-level directory that starts with **/mnt**. For example, **/mnt/test**.

14.1.23 How Do I Obtain Uploaded Files?

Take Python as an example. If you use **os.getcwd()** to query the current directory, the directory will be **/opt/function**. However, code has actually been uploaded to **/opt/function/code**.

You can use either of the following methods to obtain uploaded files:

1. Run the **cd** command to switch to **/opt/function/code**.
2. Access the full path (value of the **RUNTIME_CODE_ROOT** environment variable).

NOTE

You can obtain uploaded files by referring to the preceding methods when other languages are used.

14.1.24 Why Can't I Receive Responses for Synchronous Invocation?

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.

14.1.25 What Should I Do If the os.system("command &") Execution Logs Are Not Collected?

Do not use **os.system("command &")**. The background command output will not be collected. To obtain the command output, use **subprocess.Popen** instead.

14.1.26 Which Directories Can Be Accessed When a Custom Runtime Is Used?

By default, only the **/tmp** directory can be accessed, for example, for creating or downloading files.

14.1.27 Which Minor Versions of Python 3.6 and 3.9 Are Supported?

3.6.8 and 3.9.2.

14.1.28 Which Actions Can Be Used Instead of a VPC Administrator Agency for VPC Access?

The actions listed in **Table 14-2** can be used.

Table 14-2 Actions

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

14.1.29 What Are the Possible Causes for Function Timeout?

- The code logic timed out. In this case, optimize the code or increase the timeout.
- The network timed out. To fix this issue, increase the timeout.
- It took a long time to load Java classes during cold start. In this case, increase the timeout or memory.

14.1.30 How Do I Obtain the Code of a Function?

1. Log in to the FunctionGraph console, and click the name of the target function to go to the details page. Choose **Operation** > **Export function** in the upper right, and click **Export Code**.
2. Alternatively, call the function export API.

14.1.31 Do You Have Sample Code for Initializers?

Yes. See the following examples:

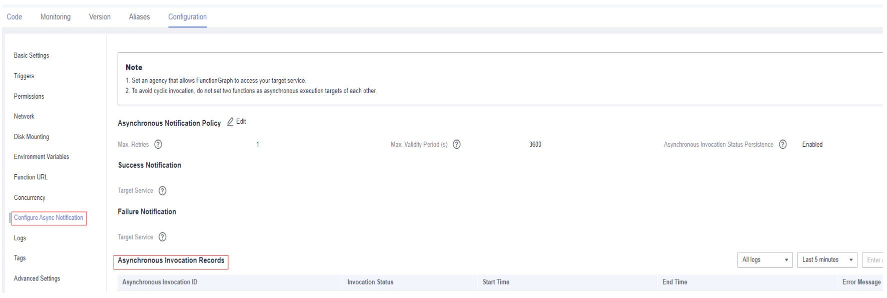
- Node.js
```
exports.initializer = function(context, callback) {
    callback(null, '');
    };
```
- Python
```
def my_initializer(context):
    print("hello world!")
```
- Java
```
public void my_initializer(Context context)
{
RuntimeLogger log = context.getLogger();
log.log(String.format("ak:%s", context.getAccessKey()));
}
```
- PHP
```
<?php
Function my_initializer($context) {
    echo 'hello world' . PHP_EOL;
    }
?>
```

14.1.32 How Do I Enable Structured Log Query?

Scenario

To check the status of asynchronous invocation requests, view the records by choosing **Configuration** > **Configure Async Notification** on the function details page, as shown in **Figure 14-1**.

Figure 14-1 Asynchronous invocation records



Note
1. Set an agency that allows FunctionGraph to access your target service.
2. To avoid cyclic invocation, do not set two functions as asynchronous execution targets of each other.

Asynchronous Notification Policy

Max. Retries 1 Max. Validity Period (s) 3600 Asynchronous Invocation Status Persistence Enabled

Success Notification

Target Service

Failure Notification

Target Service

Asynchronous Invocation Records

Asynchronous Invocation ID	Invocation Status	Start Time	End Time	Error Message

Prerequisites

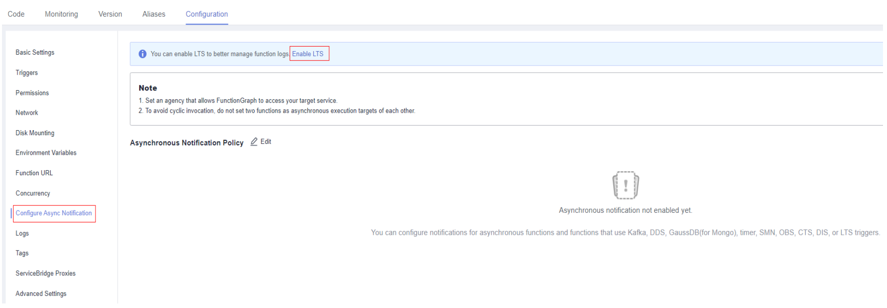
You have enabled asynchronous invocation status persistence.

Procedure

Step 1 Contact customer service to add your account to the whitelist of this feature.

Step 2 On the **Configure Async Notification** page, click **Enable LTS**, as shown in **Figure 14-2**.

Figure 14-2 Enabling LTS



You can enable LTS to better manage function logs. Enable LTS

Note
1. Set an agency that allows FunctionGraph to access your target service.
2. To avoid cyclic invocation, do not set two functions as asynchronous execution targets of each other.

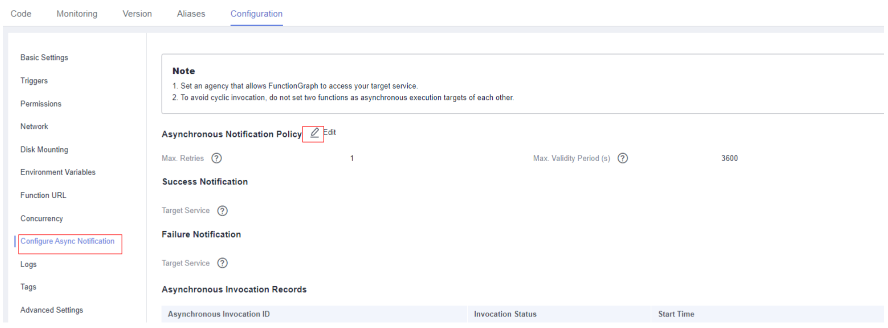
Asynchronous Notification Policy

Asynchronous notification not enabled yet.

You can configure notifications for asynchronous functions and functions that use Kafka, DDS, GaussDB(for Mongo), timer, SMN, OBS, CTS, DIS, or LTS triggers.

Step 3 Click **Edit** next to **Asynchronous Notification Policy**, and enable **Asynchronous Invocation Status Persistence**, as shown in **Figure 14-3** and **Figure 14-4**.

Figure 14-3 Configuring asynchronous notification policy



Note
1. Set an agency that allows FunctionGraph to access your target service.
2. To avoid cyclic invocation, do not set two functions as asynchronous execution targets of each other.

Asynchronous Notification Policy

Max. Retries 1 Max. Validity Period (s) 3600

Success Notification

Target Service

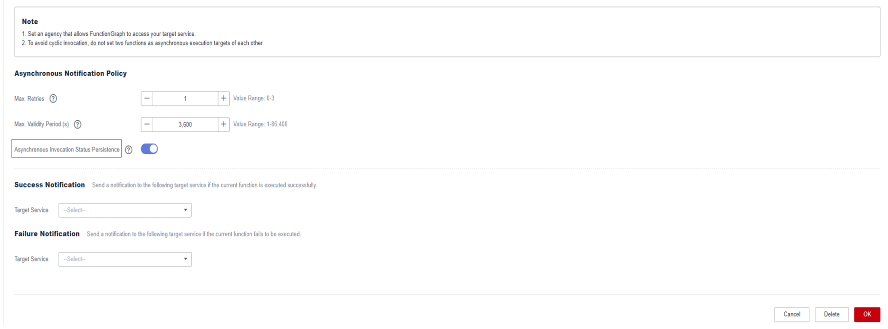
Failure Notification

Target Service

Asynchronous Invocation Records

Asynchronous Invocation ID	Invocation Status	Start Time

Figure 14-4 Enabling asynchronous invocation status persistence



Note
1. Set an agency that allows FunctionGraph to access your target service.
2. To avoid cyclic invocation, do not set two functions as asynchronous execution targets of each other.

Asynchronous Notification Policy

Max. Retries 1 Value Range: 0-3

Max. Validity Period (s) 3,600 Value Range: 1-86,400

Asynchronous Invocation Status Persistence

Success Notification Send a notification to the following target service if the current function is executed successfully

Target Service --Select--

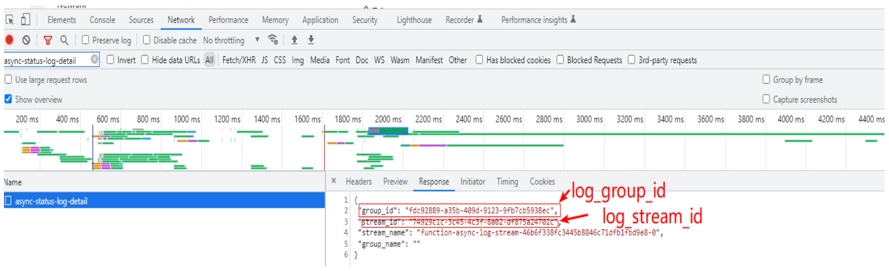
Failure Notification Send a notification to the following target service if the current function fails to be executed

Target Service --Select--

Step 4 Configure structured query on the LTS console.

1. On the function details page, view the log group and log stream. Press **F12**, choose **Network**, enter filter **async-status-log-detail**, and obtain the log group ID and log stream ID, as shown in **Figure 14-5**.

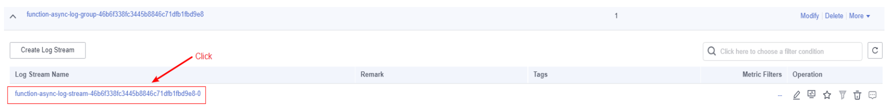
   Figure 14-5 Obtaining log group ID and log stream ID

   ```
   {
    "group_id": "fdc92889-a35b-409d-9123-9fb7cb5938ec",
    "stream_id": "74929c1c-3c45-4c3f-8a02-df875a24702c",
    "stream_name": "function-async-log-stream-46b6f338fc3445b8846c71dfb1fbd9e8-0",
    "group_name": ""
   }
   ```

2. On the LTS console, locate the log group and log stream by their IDs, as shown in **Figure 14-6**.

   Figure 14-6 Viewing log stream

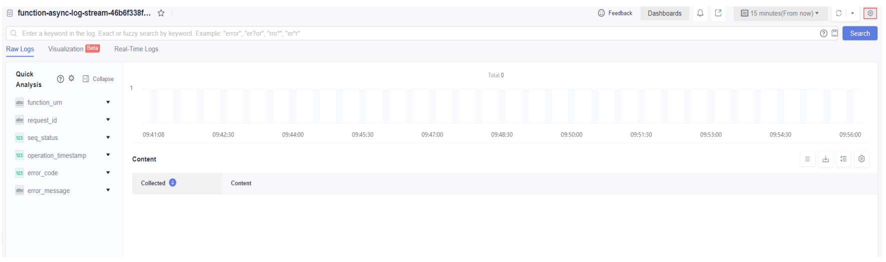
   

   function-async-log-group-46b6f338fc3445b8846c71dfb1fbd9e8 1

Log Stream Name	Remark	Tags	Metric Filters	Operation
function-async-log-stream-46b6f338fc3445b8846c71dfb1fbd9e8-0			--	

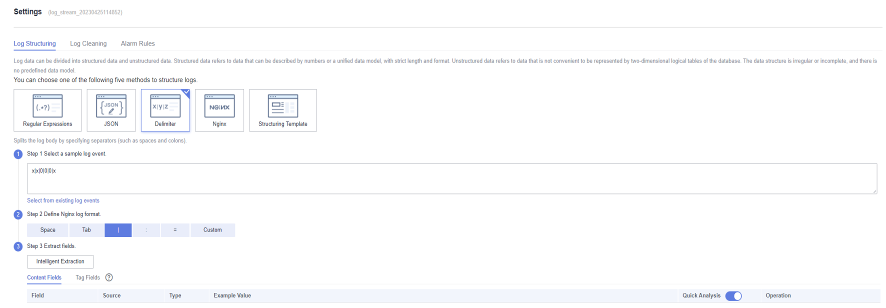
3. On the log stream details page, click the gear icon in the upper right, as shown in **Figure 14-7**.

Figure 14-7 Clicking the gear icon



4. Configure log structuring, as shown in **Figure 14-8**.

Figure 14-8 Configuring log structuring



Log data can be divided into structured data and unstructured data. Structured data refers to data that can be described by numbers or a unified data model, with strict length and format. Unstructured data refers to data that is not convenient to be represented by two-dimensional logical tables of the database. The data structure is irregular or incomplete, and there is no predefined data model.

You can choose one of the following five methods to structure logs.

Splits the log body by specifying separators (such as spaces and colons).

1. Step 1 Select a sample log event.

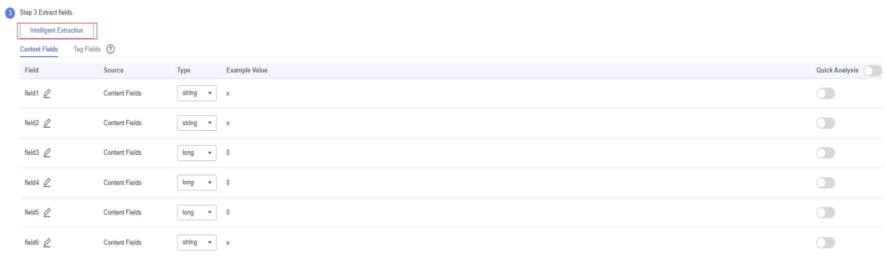
```
x|x|0|0|0|x
```

Select from existing log events

2. Step 2 Define Nginx log format.
3. Step 3 Extract fields.

5. Click **Intelligent Extraction**, as shown in **Figure 14-9**.

Figure 14-9 Intelligent Extraction

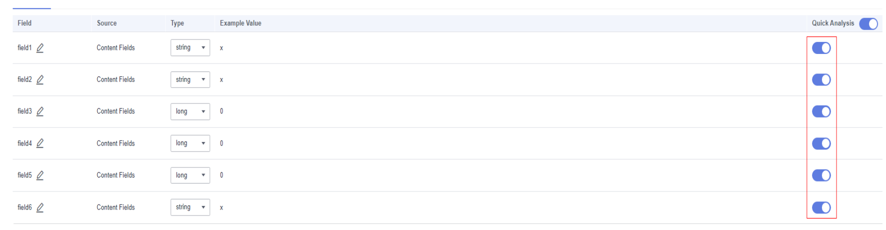


Field	Source	Type	Example Value	Quick Analysis
field1	Content Fields	string	x	
field2	Content Fields	string	x	
field3	Content Fields	long	0	
field4	Content Fields	long	0	
field5	Content Fields	long	0	
field6	Content Fields	string	x	

6. Click ✎ to modify the field definition as follows:
   a. Change **field1** to **function_urn** and its type to **string**.
   b. Change **field2** to **request_id** and its type to **string**.
   c. Change **field3** to **seq_status** and its type to **long**.
   d. Change **field4** to **operation_timestamp** and its type to **long**.
   e. Change **field5** to **error_code** and its type to **long**.
   f. Change **field6** to **error_message** and its type to **string**.

   Enable **Quick Analysis**, as shown in **Figure 14-10**.

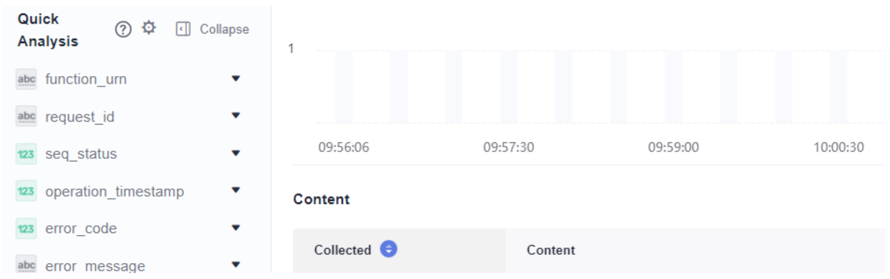
Figure 14-10 Enabling quick analysis



Field	Source	Type	Example Value	Quick Analysis
field1	Content Fields	string	x	●
field2	Content Fields	string	x	●
field3	Content Fields	long	0	●
field4	Content Fields	long	0	●
field5	Content Fields	long	0	●
field6	Content Fields	string	x	●

7. Click **Save**. **Figure 14-11** shows the configuration.

Figure 14-11 Saved configuration



----End

14.2 Function Creation FAQs

14.2.1 Can I Add Threads and Processes in Function Code?

You can create additional threads and processes in your function by using runtime and OS features.

14.2.2 What Are the Rules for Packaging a Function Project?

In addition to inline code editing, you can create a function by uploading a ZIP or JAR file, or uploading a ZIP file from OBS. For details, see **Packaging Rules** and **Example ZIP Project Packages**.

Packaging Rules

In addition to inline code editing, you can create a function by uploading a local ZIP file or JAR file, or uploading a ZIP file from Object Storage Service (OBS). **Table 14-3** describes the rules for packaging a function project.

Table 14-3 Function project packaging rules

Runtime	JAR File	ZIP File	ZIP File on OBS
Node.js	Not supported.	● If the function project files are saved under the **~/Code/** directory, select and package all files under this directory to ensure that the function handler is under the root directory after the ZIP file is decompressed. ● If the function project uses third-party dependencies, package the dependencies into a ZIP file, and import the ZIP file on the function code page. Alternatively, package the third-party dependencies and the function project files together.	Compress project files into a ZIP file and upload it to an OBS bucket.

Runtime	JAR File	ZIP File	ZIP File on OBS
Python 2.7	Not supported.	● If the function project files are saved under the **~/Code/** directory, select and package all files under this directory to ensure that the function handler is under the root directory after the ZIP file is decompressed. ● If the function project uses third-party dependencies, package the dependencies into a ZIP file, and import the ZIP file on the function code page. Alternatively, package the third-party dependencies and the function project files together.	Compress project files into a ZIP file and upload it to an OBS bucket.

Runtime	JAR File	ZIP File	ZIP File on OBS
Python 3.6	Not supported.	● If the function project files are saved under the **~/Code/** directory, select and package all files under this directory to ensure that the function handler is under the root directory after the ZIP file is decompressed. ● If the function project uses third-party dependencies, package the dependencies into a ZIP file, and import the ZIP file on the function code page. Alternatively, package the third-party dependencies and the function project files together.	Compress project files into a ZIP file and upload it to an OBS bucket.
Java 8	If the function does not reference third-party components, compile only the function project files into a JAR file.	If the function references third-party components, compile the function project files into a JAR file, and compress all third-party components and the function JAR file into a ZIP file.	Compress project files into a ZIP file and upload it to an OBS bucket.

Runtime	JAR File	ZIP File	ZIP File on OBS
Go 1.x	Not supported.	Zip the compiled file and ensure that the name of the binary file is consistent with that of the handler. For example, if the name of the binary file is **Handler**, set the name of the handler to **Handler**.	Compress project files into a ZIP file and upload it to an OBS bucket.

Example ZIP Project Packages

- Example directory of a Nods.js project package

```
Example.zip                          Example project package
|--- lib                         Service file directory
|--- node_modules                        NPM third-party component directory
|--- index.js                      .js handler file (mandatory)
|--- package.json                    NPM project management file
```

- Example directory of a Python project package

```
Example.zip                          Example project package
|--- com                           Service file directory
|--- PLI                          Third-party dependency PLI directory
|--- index.py                       .py handler file (mandatory)
|--- watermark.py                     .py file for image watermarking
|--- watermark.png                    Watermarked image
```

- Example directory of a Java project package

```
Example.zip                          Example project package
|--- obstest.jar                    Service function JAR file
|--- esdk-obs-java-3.20.2.jar         Third-party dependency JAR file
|--- jackson-core-2.10.0.jar          Third-party dependency JAR file
|--- jackson-databind-2.10.0.jar       Third-party dependency JAR file
|--- log4j-api-2.12.0.jar           Third-party dependency JAR file
|--- log4j-core-2.12.0.jar           Third-party dependency JAR file
|--- okhttp-3.14.2.jar             Third-party dependency JAR file
|--- okio-1.17.2.jar              Third-party dependency JAR file
```

- Example directory of a Go project package

```
Example.zip                          Example project package
|--- testplugin.so                   Service function package
```

14.2.3 How Does FunctionGraph Isolate Code?

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

14.2.4 How Do I Create the Bootstrap File for an HTTP Function?

To create an HTTP function, create a bootstrap file. For details, see **Creating a Bootstrap File**.

14.3 Trigger Management FAQs

14.3.1 What If Error Code 500 Is Reported When Functions that Use APIG Triggers Return Strings?

Ensure that the function response for an invocation by API Gateway has been encapsulated and contains **body(String)**, **statusCode(int)**, **headers(Map)**, and **isBase64Encoded(boolean)**.

The following is an example response returned by a Node.js function that uses an APIG trigger:

```
exports.handler = function (event, context, callback) {
    const response = {
        'statusCode': 200,
        'isBase64Encoded': false,
        'headers': {
            "Content-type": "application/json"
        },
        'body': 'Hello, FunctionGraph with APIG',
    }
    callback(null, response);
}
```

The following is an example response returned by a Java function that uses an APIG trigger:

```
import java.util.Map;

public HttpTriggerResponse index(String event, Context context){
        String body = "<html><title>FunctionStage</title>"
                + "<h1>This is a simple APIG trigger test</h1><br>"
                + "<h2>This is a simple APIG trigger test</h2><br>"
                + "<h3>This is a simple APIG trigger test</h3>"
                + "</html>";
        int code = 200;
        boolean isBase64 = false;
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "text/html; charset=utf-8");
        return new HttpTriggerResponse(body, headers, code, isBase64);
    }


class HttpTriggerResponse {
    private String body;
    private Map<String, String> headers;
    private int statusCode;
    private boolean isBase64Encoded;
    public HttpTriggerResponse(String body, Map<String,String> headers, int statusCode,
boolean isBase64Encoded){
        this.body = body;
        this.headers = headers;
        this.statusCode = statusCode;
        this.isBase64Encoded = isBase64Encoded;
    }
}
```

14.3.2 What Do LATEST and TRIM_HORIZON Mean in DIS Trigger Configuration?

Cursors **LATEST** and **TRIM_HORIZON** specify the start points for reading data in Data Ingestion Service (DIS) streams.

- **TRIM_HORIZON**: Data is read from the earliest valid record stored in the partition.

  For example, a tenant used a DIS stream to upload three pieces of data A1, A2, and A3. Assuming that A1 expires but A2 and A3 are still valid after a period of time, if the tenant specifies **TRIM_HORIZON** for downloading data, only A2 and A3 can be downloaded.
- **LATEST**: Data is read from the latest record in the partition. This option ensures that the most recent data in the partition is read.

14.3.3 Why Can't I Enable or Disable OBS Triggers by Calling APIs?

OBS does not support pull triggers. Therefore, OBS triggers cannot be enabled or disabled.

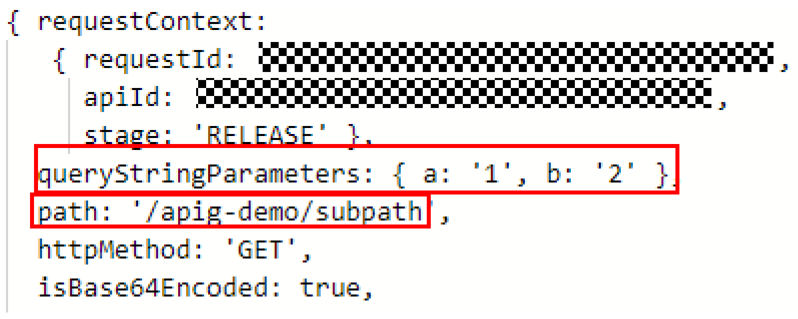
14.3.4 How Do I Use an APIG Trigger to Invoke a Function?

For details, see **Using an APIG Trigger**.

14.3.5 How Does a Function Obtain the Request Path or Parameters When Using an APIG Trigger?

By default, the request path or parameters are included in **event**. A function invokes APIG using its event template. You can obtain the request path or parameters from the function execution result.

Example:



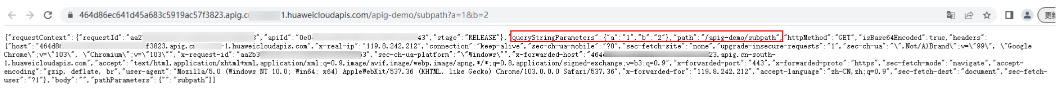
```
{ requestContext:
   { requestId: [redacted],
     apiId: [redacted],
     stage: 'RELEASE' },
  queryStringParameters: { a: '1', b: '2' },
  path: '/apig-demo/subpath',
  httpMethod: 'GET',
  isBase64Encoded: true,
```

- **queryStringParameters**: parameters following the question mark (?), separated with ampersands (&). These parameters are added in the URL of a GET request, and will be transferred in the URL string format when a GET request is initiated.
- **path**: API URL.

You can call an API using its request path. Example: **https://464d86ec641d45a683c5919ac57f3823.apig.projectID.huaweicloudapis.com/apig-demo/subpath**

Alternatively, you can call an API by adding request parameters. Example:

https://464d86ec641d45a683c5919ac57f3823.apig.projectID.huaweicloudapis.com/apig-demo**/subpath?a=1&b=2**



464d86ec641d45a683c5919ac57f3823.apig.c[redacted]1.huaweicloudapis.com/apig-demo/subpath?a=1&b=2

```
{"requestContext":{"requestId":"aa2[redacted]3","apiId":"0e0[redacted]43","stage":"RELEASE"},"queryStringParameters":{"a":"1","b":"2"},"path":"/apig-demo/subpath","httpMethod":"GET","isBase64Encoded":true,"headers":
{"host":"464d8[redacted]f3823.apig.c[redacted]-1.huaweicloudapis.com","x-real-ip":"119.8.242.212","connection":"keep-alive","sec-ch-ua-mobile":"?0","sec-fetch-site":"none","upgrade-insecure-requests":"1","sec-ch-ua":"\".Not/A)Brand\";v=\"99\", \"Google
Chrome\";v=\"103\", \"Chromium\";v=\"103\"","x-request-id":"aa2b3[redacted]53","sec-ch-ua-platform":"\"Windows\"","x-forwarded-host":"464[redacted]23.apig.cn-south-
1.huaweicloudapis.com","accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9","x-forwarded-port":"443","x-forwarded-proto":"https","sec-fetch-mode":"navigate","accept-
encoding":"gzip, deflate, br","user-agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36","x-forwarded-for":"119.8.242.212","accept-language":"zh-CN,zh;q=0.9","sec-fetch-dest":"document","sec-fetch-
user":"?1"},"body":"","pathParameters":{"":"subpath"}}
```

14.3.6 Can I Create an OBS Trigger with an Existing Bucket?

Yes. If a message is displayed indicating that the configuration of the current trigger conflicts with that of another one, the two triggers have the same bucket, prefix, and suffix. If you still want to use this bucket for the current trigger, modify the prefix or suffix.

14.4 Dependency Management FAQs

14.4.1 What Is a Dependency?

A dependency is a program package and also an environment required for running a software package. The software package relies on and can only run in the environment.

14.4.2 When Do I Need a Dependency?

When you install a program or develop code that relies on an environment to run, you need to introduce the dependency.

14.4.3 What Are the Precautions for Using a Dependency?

- The name of each file in a dependency cannot end with a tilde (~).
- There should be no more than 30,000 files in a dependency.
- You can upload a ZIP dependency file within 10 MB on the function details page. For a larger dependency (max. 300 MB), upload it using OBS.
- If your function uses a large private dependency, increase the timeout by choosing **Configuration** > **Basic Settings** on the function details page.

14.4.4 What Dependencies Does FunctionGraph Support?

Supported Dependencies

FunctionGraph supports standard libraries and third-party dependencies.

- 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, as described in **Table 14-4** and **Table 14-5**. You can import these components to your inline code in the same way as you import standard libraries.

Table 14-4 Third-party components integrated with the Node.js runtime

Name	Description	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 14-5 Non-standard libraries supported by the Python runtime

Module	Description	Version
dateutil	Date and time processing	2.6.0
requests	HTTP library	2.7.0
httplib2	httpclient	0.10.3
numpy	Mathematical computation	1.13.1
redis	Redis client	2.10.5
obsclient	OBS client	-
smnsdk	SMN access	1.0.1

- Other third-party libraries

  For other third-party libraries not listed in the preceding tables, package and upload them to an OBS bucket or on the function details page. For details, see **How Do I Create a Dependency on the FunctionGraph Console?** These libraries will then be used in your function code.

14.4.5 Does FunctionGraph Support Class Libraries?

Yes. FunctionGraph supports both standard libraries and non-standard third-party libraries. For details, see **What Dependencies Does FunctionGraph Support?**

14.4.6 How Do I Use Third-Party Dependencies on FunctionGraph?

1. Package third-party libraries into a ZIP package by referring to **How Do I Create Function Dependencies?**
2. Create a dependency on the FunctionGraph console by referring to **How Do I Create a Dependency on the FunctionGraph Console?**
3. On the function details page, click the **Code** tab, and add the dependency by referring to **How Do I Add a Dependency to a Function?** Then you can use the dependency in the function code.

14.4.7 How Do I Create Function Dependencies?

You are advised to create function dependencies in EulerOS. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

NOTE

If the modules to be installed need dependencies such as .dll, .so, and .a, archive them to a .zip package.

Creating a Dependency for a Python Function

Ensure that the Python version of the packaging environment is the same as that of the function. For Python 2.7, Python 2.7.12 or later is recommended. For Python 3.6, Python 3.6.3 or later is recommended.

To install the PyMySQL dependency for a Python 2.7 function in the local **/tmp/pymysql** directory, run the following command:

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

After the command is successfully executed, go to the **/tmp/pymysql** directory:

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

Go to the **site-packages** directory (generally, **usr/lib64/python2.7/site-packages/**) and then run the following command:

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

The required dependency is generated.

NOTE

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

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

Creating a Dependency for a Node.js Function

Ensure that the corresponding Node.js version has been installed in the environment.

To install the MySQL dependency for a Node.js 8.10 function, run the following command:

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

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

- Linux OS

  Run the following command to generate a ZIP package.

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

  The required dependency is generated.

- Windows OS

  Compress **node_modules** into a ZIP file.

To install multiple dependencies, create a **package.json** file first. For example, enter the following content into the **package.json** file and then run the following command:

```
{
    "name": "test",
    "version": "1.0.0",
    "dependencies": {
        "redis": "~2.8.0",
        "mysql": "~2.17.1"
    }
}
npm install --save
```

NOTE

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

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

For other Node.js versions, you can create dependencies in the way stated above.

Creating a Dependency for a Java Function

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

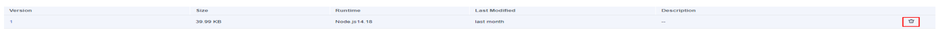
14.4.8 How Do I Create a Dependency on the FunctionGraph Console?

1. Log in to the FunctionGraph console, and choose **Functions** > **Dependencies** in the navigation pane.
2. Click **Create Dependency**.
3. Set the following parameters.

Table 14-6 Dependency configuration parameters

Parameter	Description
Name	Dependency name.
Code Entry Mode	Upload a ZIP file directly or through OBS. ● **Upload ZIP file**: Click **Select File** to upload a ZIP file. ● **Upload from OBS**: Specify an OBS link URL.
Runtime	Select a runtime.
Description	Description of the dependency. This parameter is optional.

4. Click **OK**. By default, a new dependency is version **1**.
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.



Version	Size	Runtime	Last Modified	Description	
1	39.99 KB	Node.js14.18	last month	--	🗑

14.4.9 How Do I Add a Dependency to a Function?

1. On the function details page, click the **Code** tab, and click **Add** in the **Dependencies** area.
   – **Public**: Public dependencies are provided by FunctionGraph and can be directly added.
   – **Private**: Private dependencies are those you created and uploaded.
2. Click **OK**.

14.5 Function Execution FAQs

14.5.1 How Long Does It Take to Execute a FunctionGraph Function?

Within 900s for synchronous execution and 72 hours for asynchronous execution.

The default execution timeout is 3s. You can set the timeout (unit: s) to an integer from 3 to 259,200. If you set the timeout of a function to 3s, it will be terminated after 3s.

14.5.2 Which Steps Are Included in Function Execution?

Function execution includes two steps:

1. Select an idle instance with required memory.
2. Run specified code.

14.5.3 How Does FunctionGraph Process Concurrent Requests?

FunctionGraph automatically scales in or out function instances based on the number of requests. If the number of concurrent requests increases, FunctionGraph allocates more function instances to process the requests. If that number decreases, FunctionGraph allocates fewer function instances accordingly.

Number of function instances = Function concurrency/Concurrency per instance

- Function concurrency: the number of requests concurrently executed by a function at a certain time point.
- Concurrency per instance: the maximum number of concurrent requests allowed by a single instance. This is equivalent to the **Max. Requests per Instance** parameter on the **Concurrency** page.

14.5.4 What If Function Instances Have Not Been Executed for a Long Time?

If a function has not been executed for a period of time, all instances related to the function will be released.

14.5.5 How Can I Speed Up Initial Access to a Function?

C# and Go support a lower startup speed than other languages due to mechanism issues. You can use the following methods to speed up initial access to a function:

- Allocate more memory to the function.
- Simplify function code, for example, delete unnecessary dependency packages.
- When using C# in non-concurrent scenarios, you can also:

  Create a one-minute timer trigger to ensure that there is at least one active instance.

14.5.6 How Do I Know the Actual Memory Used for Function Execution?

The returned information about a function contains the maximum memory consumed. Alternatively, check the memory usage in the execution result.

14.5.7 Why Is My First Request Slow?

Functions are cold-started. If initialization or a lengthy operation is performed during the first function execution, the first request will be delayed. However, subsequent requests before container deletion will be faster. If there is no request within one minute, the container will be deleted.

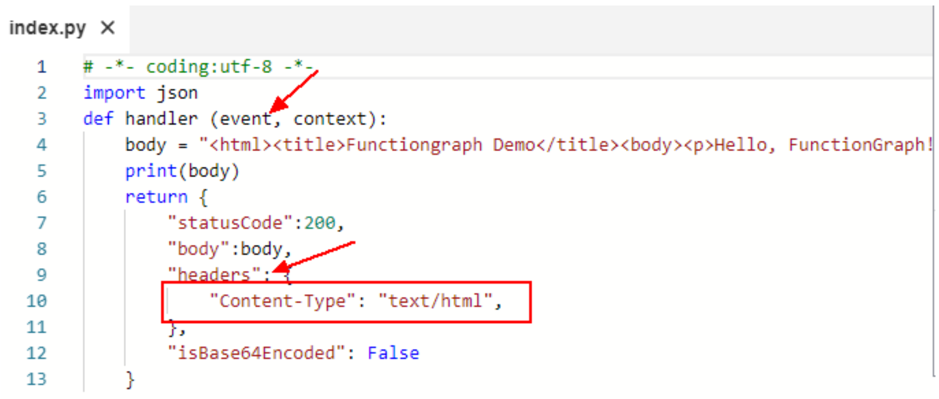
14.5.8 What Do I Do If an Error Occurs When Calling an API?

Rectify the fault by referring to **Error Codes**. If the fault persists, contact technical support.

14.5.9 How Do I Read the Request Header of a Function?

The first parameter in the function handler contains the request header. You can print the function execution result to obtain required fields.

As shown in the following figure, **event** is the first parameter in the function handler, and **headers** is the request header.



```
index.py ×
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    body = "<html><title>Functiongraph Demo</title><body><p>Hello, FunctionGraph!
    print(body)
    return {
        "statusCode":200,
        "body":body,
        "headers": {
            "Content-Type": "text/html",
        },
        "isBase64Encoded": False
    }
```

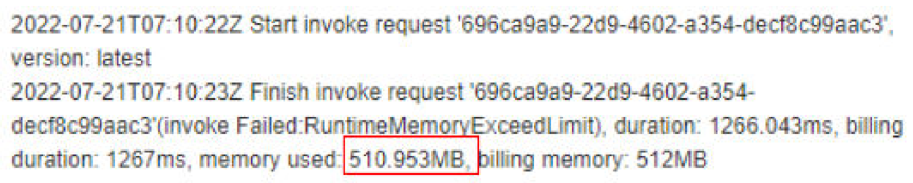
14.5.10 Why Does a Function Use More Memory Than Estimated and Even Trigger the Out of Memory Alarm?

1. Event parsing and cache consume extra memory during function invocation.
2. After the invocation is complete, reclaimed memory is often put in the internal pool instead of back to the OS, resulting in high memory usage. This is more obvious in the case of high concurrency.

14.5.11 How Do I Check the Memory Usage When Seeing "runtime memory limit exceeded"?

Check the used memory in the response.

Figure 14-12 Checking the used memory



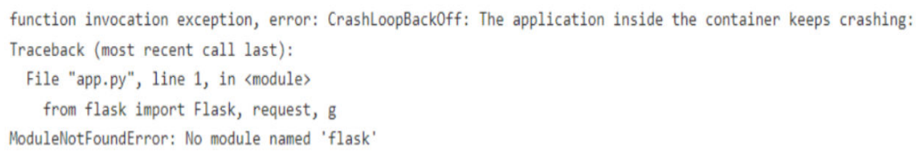
```
2022-07-21T07:10:22Z Start invoke request '696ca9a9-22d9-4602-a354-decf8c99aac3',
version: latest
2022-07-21T07:10:23Z Finish invoke request '696ca9a9-22d9-4602-a354-
decf8c99aac3'(invoke Failed:RuntimeMemoryExceedLimit), duration: 1266.043ms, billing
duration: 1267ms, memory used: 510.953MB, billing memory: 512MB
```

14.5.12 How Do I Troubleshoot "CrashLoopBackOff"?

The message "CrashLoopBackOff: The application inside the container keeps crashing" is displayed when a custom image execution failure occurs. In this case, perform the following operations:

1. Analyze the causes.

   Figure 14-13 Viewing the execution result

   ```
   function invocation exception, error: CrashLoopBackOff: The application inside the container keeps crashing:
   Traceback (most recent call last):
     File "app.py", line 1, in <module>
       from flask import Flask, request, g
   ModuleNotFoundError: No module named 'flask'
   ```

2. Verify the container image by referring to **Deploying a Function Using a Container Image**.
3. Check whether the image uses the Linux x86 architecture. Currently, only Linux x86 images are supported.

14.5.13 After I Updated an Image with the Same Name, Reserved Instances Still Use the Old Image. What Can I Do?

Use a non-latest tag to manage image updates, and do not use the same image name.

14.6 Function Configuration FAQs

14.6.1 Can I Set Environment Variables When Creating Functions?

Yes. Set variables to dynamically pass settings to your function code and libraries without changing your code. For more information, see section "Configuring Environment Variables" in the *FunctionGraph User Guide*.

14.6.2 Can I Enter Sensitive Information in Environment Variables?

FunctionGraph displays all the information you enter in plain text. Therefore, do not enter insensitive information such as passwords when you define environment variables.

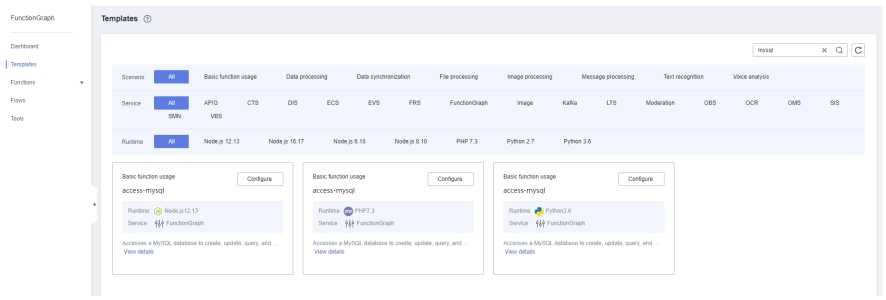
14.7 External Resource Access FAQs

14.7.1 How Does a Function Access the MySQL Database?

Perform the following operations:

1. Check whether the MySQL database is deployed in a VPC.
   - Yes: Configure the same VPC and subnet as the MySQL database for the function by referring to **Configuring VPC Access**.
   - No: See **How Do I Configure External Network Access?**
2. Search for MySQL templates and select the one with the desired runtime, as shown in **Figure 14-14**. Set the parameters as required and click **Create Function**.

Figure 14-14 Selecting a function template



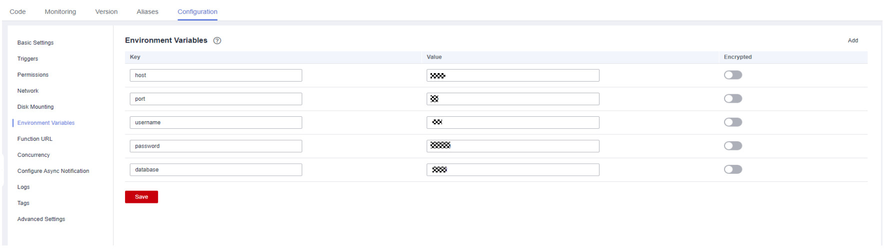
Basic function usage
Configure
access-mysql
Runtime Node.js12.13
Service FunctionGraph
Accesses a MySQL database to create, update, query, and ...
View details

Basic function usage
Configure
access-mysql
Runtime PHP7.3
Service FunctionGraph
Accesses a MySQL database to create, update, query, and ...
View details

Basic function usage
Configure
access-mysql
Runtime Python3.6
Service FunctionGraph
Accesses a MySQL database to create, update, query, and ...
View details

3. After the MySQL function is created, choose **Configuration** > **Environment Variables**, enable encryption as required (see **Figure 14-15**), and click **Save**.

Figure 14-15 Enabling encryption



Environment Variables

Key	Value	Encrypted
host	[redacted]	○
port	[redacted]	○
username	[redacted]	○
password	[redacted]	○
database	[redacted]	○

NOTE

If the function needs to access RDS APIs, **create an agency** and grant required permissions.

14.7.2 How Does a Function Access Redis?

Perform the following operations:

1. Check whether the Redis instance is deployed in a VPC.
   - If the Redis instance is deployed in a VPC, configure the same VPC and subnet as the Redis instance for the function by referring to **Configuring VPC Access**.
   - If the Redis instance is built on a public network, obtain its public IP address.
2. Compile code for connecting a function to the Redis instance.

   FunctionGraph has integrated third-party library **redis-py** in its Python 2.7 and Python 3.6 runtimes. Therefore, you do not need to download any other Redis libraries.

```
# -*- coding:utf-8 -*-
import redis
def handler (event, context):
    r = redis.StrictRedis(host="host_ip",password="passwd",port=6379)
    print(str(r.get("hostname")))
    return "^_^"
```

NOTE

- If the function fails to access to the Redis instance on a public network, perform the following operations:
  - Modify the **redis.conf** file to allow access from any IP addresses.
  - Set a password for accessing the Redis instance in the **redis.conf** file.
  - Disable the firewall.
- If the function needs to access DCS APIs, **create an agency** and grant required permissions.

14.7.3 How Do I Configure External Network Access?

By default, functions deployed in a VPC are isolated from the Internet. If a function needs to access both internal and external networks, add a NAT gateway for the VPC.

Prerequisites

1. You have created a VPC and subnet according to **Creating a VPC**.
2. You have obtained an elastic IP address according to **Assigning an EIP**.

Procedure of Creating a NAT Gateway

Step 1 Log in to the NAT Gateway console, and click **Create NAT Gateway**.

Step 2 On the displayed page, enter gateway information, select a VPC and subnet (for example, **vpc-01**), and confirm and submit the settings to create a NAT gateway. For details, see **Creating a NAT Gateway**.

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

----End

14.8 Other FAQs

14.8.1 How Do I View the Alarm Rules Configured for a Function?

Log in to the Cloud Eye console and view alarm rules.

14.8.2 Does FunctionGraph Support ZIP Decompiling During Video Transcoding?

No. Please decompile your files before uploading them.

15 Change History

Table 15-1 Change history

Date	Description
2023-5-30	This issue is the first official release.