FunctionGraph

User Guide

Issue	01
Date	2026-01-09

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Huawei Cloud Computing Technologies Co., Ltd.

Address:	Huawei Cloud Data Center Jiaoxinggong Road Qianzhong Avenue Gui'an New District Gui Zhou 550029 People's Republic of China
Website:	https://www.huaweicloud.com/intl/en-us/

Contents

1 Service Overview

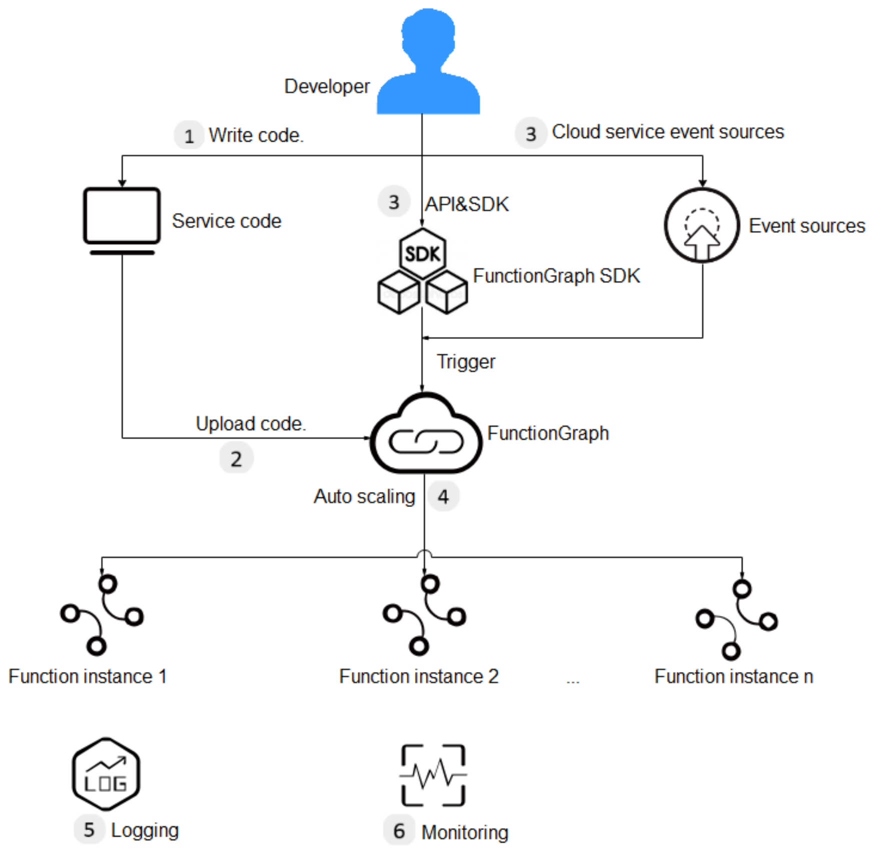
1.1 Overview

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.

Function Service Process

Figure 1-1 shows the process of using functions.

Figure 1-1 Function service process



The process is explained as follows:

1. Write code in Node.js, Python, Java, Go, C#, PHP or Cangjie. For details, see the **FunctionGraph Developer Guide**.
2. Alternatively, edit code inline, directly upload a ZIP file, or upload a ZIP file from Object Storage Service (OBS). For details, see **Table 1-2**.
3. Create an API or set a cloud service event source to trigger the function. For details, see **FunctionGraph Getting Started**.
4. 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 **Function Running Resource Restrictions**.
5. FunctionGraph works with Log Tank Service (LTS), allowing you to query run logs of your function without the need for configurations. For details, see **Querying Logs**.
6. FunctionGraph works with Cloud Eye, allowing you to view graphical monitoring information about your function without the need for configurations. For details, see **Function Monitoring**.

1.2 Product Features

Function Management

FunctionGraph provides console-based function management.

- The Node.js, Java, Python, Go, C#, PHP, Cangjie, 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, 14.18, 16.17, 18.15, 20.15
Python	2.7, 3.6, 3.9, 3.10, 3.12
Java	8, 11, 17, 21
Go	1.x
C#	.NET Core 2.1, .NET Core 3.1, .NET Core 6.0, .NET Core 8.0
PHP	7.3, 8.3
Custom	-
Cangjie	1.0

- 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
C#	Not supported	Supported	Not supported	Supported
PHP	Supported	Supported	Not supported	Supported
Custom	Supported	Supported	Not supported	Supported
Cangjie	Not supported.	Supported	Not supported	Supported

Trigger

FunctionGraph supports multiple trigger types, such as Simple Message Notification (SMN), API Gateway (APIG), and OBS trigger types. **Table 1-3** lists the supported trigger types and the function invocation mode of each trigger type.

Table 1-3 Function invocation modes

Trigger	Function Invocation Mode
SMN trigger	Asynchronous invocation
APIG trigger	Synchronous invocation
OBS trigger	Asynchronous invocation
Data Ingestion Service (DIS) 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.

For details on how to query logs, see **Querying Logs**.

For details on how to view function metrics, see **Viewing Function Metrics**.

For details about tenant-level monitoring information, see **Dashboard**.

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 API Gateway (APIG) 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 intelligently 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 artificial intelligence (AI) application.

Real-time File Processing

Uploading files from a client to OBS triggers functions that create image thumbnails in real time, convert video formats, aggregate and filter data files, or implement other file operations.

Advantages:

- FunctionGraph automatically scales out resources to run more function instances as the number of requests increases.
- Files are uploaded to OBS to trigger file processing functions.

Web & Mobile Backends

Interconnect FunctionGraph with other cloud services or your VMs to quickly build highly available and scalable web & mobile backends.

Advantages:

- FunctionGraph ensures high reliability of website data using OBS and CloudTable, and high-availability of website logic using API Gateway.
- FunctionGraph automatically scales out resources to run more function instances as the number of requests increases.

Artificial Intelligence

Integrate FunctionGraph with EI services for text recognition and illicit image identification.

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.

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. 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 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 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

- Only APIG (shared) triggers can be created for HTTP functions.
- 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 Related Services

Table 1-4 describes the cloud services that have been interconnected with FunctionGraph.

Table 1-4 Interconnected services

Service	Function
SMN	FunctionGraph functions are constructed to process SMN notifications. For details, see .
API Gateway	FunctionGraph functions are invoked over HTTPS by defining REST APIs with specified backend services.
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.

Service	Function
LTS	FunctionGraph functions are built to process logs subscribed to in LTS. When LTS collects subscribed logs, the function is triggered to process or analyze the logs or to load the logs to other systems.
CTS	FunctionGraph functions are defined to analyze and process key information in logs according to the event notifications of specified service type and operations configured in CTS. ● With CTS, you can record operations associated with FunctionGraph for later query, audit, and backtrack operations. **Table 1-5** lists the FunctionGraph operations that can be recorded by CTS. ● CTS starts recording operations on cloud resources once being enabled. View traces of the last seven days on the CTS console.
DDS	DDS triggers trigger FunctionGraph functions upon a table change in the database.
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.

Table 1-5 FunctionGraph operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Creating a function	Function	CreateFunction
Deleting a function	Function	DeleteFunction
Modifying function information	Function	ModifyFunctionMetadata
Publishing a function version	Function version	PublishFunctionVersion
Deleting a function alias	Function version alias	DeleteVersionAlias
Deleting a function trigger	Trigger	DeleteTrigger
Creating a function trigger	Trigger	CreateTrigger
Disabling a function trigger	Trigger	DisabledTrigger

Operation	Resource Type	Trace Name
Enabling a function trigger	Trigger	enabledTrigger

1.7 Quotas and Usage Restrictions

Account Resource Restrictions

The following table provides the quotas for account resources. For details on how to query and modify quotas, see **Quotas**.

Table 1-6 Account resource restrictions

Resource	Restriction	Adjustable
Maximum number of functions that can be created under an account	400	No
Maximum number of versions allowed for a function	20	No
Maximum number of aliases allowed for a function	10	No
Maximum number of DIS, LTS, Kafka, and Timer triggers allowed for a function version	10	No
Size of a code deployment package (in ZIP or JAR format) that can be uploaded to the FunctionGraph console	40 MB	No
Size of a code deployment package (in ZIP or JAR format) that can be edited inline during function API invocation	50 MB	No
Size of an original code deployment package allowed during function API invocation	• ZIP: 1500 MB (after decompression) • OBS bucket: 300 MB (after compression)	No
Maximum size of deployment packages allowed for an account	10 GB	No
Number of concurrent executions per account	100	Yes

Resource	Restriction	Adjustable
Maximum number of reserved instances that an account can create	90 (Number of concurrent executions per account x 90%)	Yes
Size of all environment variables of a function	4,096 characters	No
Maximum size of code that can be displayed on the console	20 MB	No

Function Running Resource Restrictions

Table 1-7 Function running resource restrictions

Resource	Restriction	Adjustable
Ephemeral disk space (**/tmp** space)	512 MB	No
Number of file descriptors	1024	No
Total number of processes and threads	1024	No
Maximum execution duration per request	259,200s	Yes
Valid payload size of invocation request body (synchronous invocation)	6 MB	No
Valid payload size of invocation response body (synchronous invocation)	6 MB	No
Valid payload size of invocation request body (asynchronous invocation)	256 KB	No
Size of imported resources	≤ 50 MB ZIP file	No
Size of exported resources	≤ 50 MB	No
Instances per tenant	1000	Yes
Max. memory per function	10 GB	No
Bandwidth	Unlimited	-
Size of a log	Unlimited	-
Maximum execution duration of initializer	259,200s	Yes

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. If the function execution time is longer than 90s, use asynchronous invocation.
- The valid payload size of a request body is 6 MB when a Kafka or DIS trigger is used and is 4 MB when an APIG trigger is used.

2 User Guide

2.1 Overview

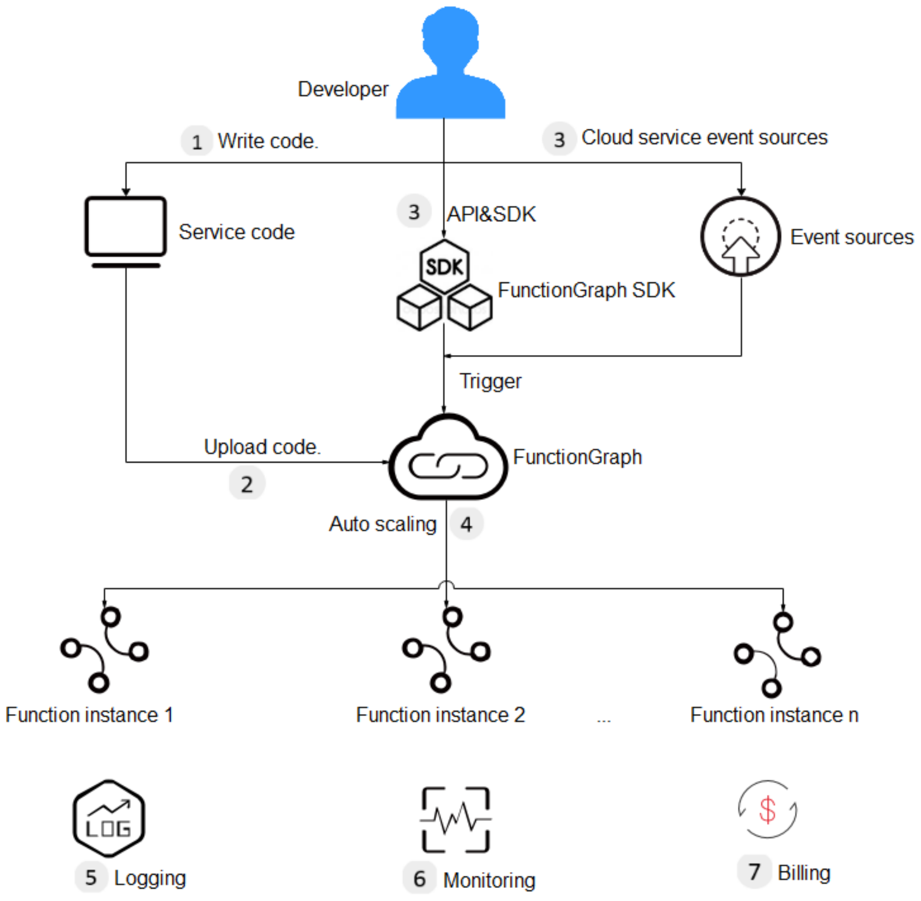
2.1.1 How Do I Use FunctionGraph?

Process

Figure 2-1 shows the process of using functions.

1. Write code, package and upload it to FunctionGraph, and add event sources such as Simple Message Notification (SMN), Object Storage Service (OBS), and API Gateway (APIG) 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 run logs and monitoring metrics of a function. You will be billed for the code execution, but will not be charged when the code is not running.

Figure 2-1 Flowchart



The following shows the details:

1. Write code in Node.js, Python, Java, Go, C#, or PHP. For details, see the FunctionGraph Developer Guide.
2. Edit code inline, upload a local ZIP or JAR file, or upload a ZIP file from OBS. For details, see **Table 2-30**.
3. Functions are triggered by API calls or event sources. For details, see **Using an SMN Trigger**, **Using an APIG (Dedicated) Trigger**, **Using an OBS Trigger**, **Using a DIS Trigger**, **Using a Timer Trigger**, **Using a CTS Trigger**, **Using an LTS Trigger**, and **Using a Kafka Trigger**.
4. FunctionGraph implements auto scaling based on the number of requests. For details about the maximum number of function instances that can be run concurrently, see **Quotas and Usage Restrictions**.
5. View run logs of function as FunctionGraph is interconnected with Log Tank Service (LTS). For details, see **Querying Logs**.
6. View graphical monitoring information as FunctionGraph is interconnected with Cloud Eye. For details, see **Viewing Function Metrics**.

2.1.2 Permissions Management

2.1.2.1 Creating a User and Granting Permissions

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

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

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

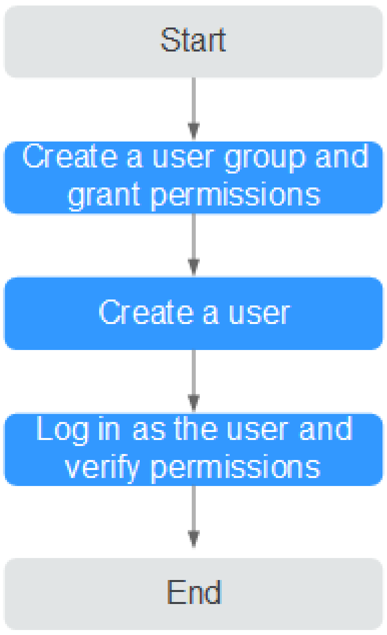
This section describes the procedure for granting permissions. For details, see **Figure 2-2**.

Prerequisites

You have understood which FunctionGraph permissions will be granted to the target user group.

Process

Figure 2-2 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. Log in to the management console as the created user and check whether this user only has read permissions for FunctionGraph:

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

2.1.2.2 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.

This section provides examples of common custom FunctionGraph policies.

Example Custom Policies

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

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

- Example 2: Denying function deletion

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

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

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

- Example 3: Configuring permissions for specific resources

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

  NOTE

  Specify function resources:

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

  For function resources, IAM automatically generates the resource path prefix **FUNCTIONGRAPH:*:*:function:**. You can specify a resource path by adding the application or function name next to the path prefix. Wildcards (*) are supported. For example, **FUNCTIONGRAPH:*:*:function:Default/*** indicates any function in the **Default** application.

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "functiongraph:function:list"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "functiongraph:function:listAlias",
                "functiongraph:function:listVersion",
                "functiongraph:function:getConfig",
                "functiongraph:function:getCode",
                "functiongraph:function:updateCode",
                "functiongraph:function:invoke",
                "functiongraph:function:updateConfig",
                "functiongraph:function:createVersion",
                "functiongraph:function:updateAlias",
                "functiongraph:function:createAlias"
            ],
            "Resource": [
                "FUNCTIONGRAPH:*:*:function:Default/*"
            ]
        }
    ]
}
```

2.1.3 Building a FunctionGraph Function

2.1.3.1 Programming Models

Node.js

FunctionGraph currently supports Node.js 6.10, Node.js 8.10, Node.js 10.16, Node.js 12.13, Node.js 14.18, Node.js 16.17, Node.js 18.15, and Node.js 20.15. For details about programming, SDK APIs, and function development, see **Developing Functions in Node.js**.

Python

FunctionGraph supports Python 2.7, Python 3.6, Python 3.9, Python 3.10, and Python 3.12. For details about Python function APIs, SDK APIs, and function development, see **Developing Functions in Python**.

Java

FunctionGraph currently supports Java 8, Java 11, Java 17, and Java 21. For details about Java function APIs, SDK APIs, and function development, see **Developing Functions in Java**.

Go

FunctionGraph supports only Go 1.x. For details about Go function APIs, SDK APIs, and function development, see **Developing Functions in Go**.

C#

FunctionGraph supports C# (.NET Core 2.1), C# (.NET Core 3.1), C# (.NET Core 6.0), and C# (.NET Core 8.0). For details about C# function APIs, SDK APIs, and function development, see **Developing Functions in C#**.

PHP

FunctionGraph supports PHP 7.3 and 8.3. For details about PHP function syntax, SDK APIs, and function development, see **Developing Functions in PHP**.

Custom Runtime

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. For more information about custom runtimes, see **Using a Custom Runtime**.

Cangjie

FunctionGraph supports Cangjie 1.0. For details about how to develop Cangjie functions, see **Developing Functions in Cangjie**.

2.1.3.2 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.

Table 2-1 lists the code entry modes supported by FunctionGraph for each runtime.

Table 2-1 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
Python	Supported	Supported	Not supported	Supported
Java	Not supported	Supported	Supported	Supported
Go	Not supported	Supported	Not supported	Supported
C#	Not supported	Supported	Not supported	Supported
PHP	Supported	Supported	Not supported	Supported
Custom runtime	Supported	Supported	Not supported	Supported
Cangjie	Not 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.

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, you need to 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.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 50 MB, upload the package using an OBS bucket.

  For details about function resource restrictions, see **Quotas and Usage Restrictions**.

- 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 **Quotas and Usage Restrictions**.

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, you need to 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.
- 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 50 MB, upload the package using an OBS bucket.

  For details about function resource restrictions, see **Quotas and Usage Restrictions**.

- 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 **Quotas and Usage Restrictions**.

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.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 50 MB, upload the package using an OBS bucket.

  For details about function resource restrictions, see **Quotas and Usage Restrictions**.
- 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 **Quotas and Usage Restrictions**.

Go

Uploading a Deployment Package

You can only upload a Go deployment package in ZIP format, and you can upload it 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.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 50 MB, upload the package using an OBS bucket.

  For details about function resource restrictions, see **Quotas and Usage Restrictions**.

- 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 **Quotas and Usage Restrictions**.

C#

Uploading a Deployment Package

You can only upload a C# deployment package in ZIP format, and you can upload it 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.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 50 MB, upload the package using an OBS bucket.

  For details about function resource restrictions, see **Quotas and Usage Restrictions**.

- 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 **Quotas and Usage Restrictions**.

PHP

Editing Code Inline

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

Uploading a Deployment Package

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

NOTICE

When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 50 MB, upload the package using an OBS bucket.

  For details about function resource restrictions, see **Quotas and Usage Restrictions**.
- 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 **Quotas and Usage Restrictions**.

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, you need to 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.

- Directly uploading a local deployment package

  After creating a ZIP deployment package, upload it to the FunctionGraph console. If the package size exceeds 50 MB, upload the package using an OBS bucket.

  For details about function resource restrictions, see **Quotas and Usage Restrictions**.

- 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 **Quotas and Usage Restrictions**.

Cangjie

Cangjie is a statically typed language that does not allow online code editing. You can only upload code through a ZIP file.

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.

- 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 **Quotas and Usage Restrictions**.

- 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 **Quotas and Usage Restrictions**.

2.1.3.3 Creating a Function

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

You can create a function from scratch or by using an existing template.

Creating a Function from Scratch

When creating a function from scratch, configure the basic and code information based on **Table 2-2** and **Table 2-3**. The parameters marked with an asterisk (*) are mandatory.

For environment information, retain the default values, as shown in **Table 2-5**.

Table 2-2 Basic information

Parameter	Description
*Function Name	Function name.
*App	Select an existing app or define a new app to which the function belongs. You can create multiple functions under an app.
*Enterprise Project	Select an enterprise project to add the function to it. **NOTE** If Enterprise Project Management Service (EPS) is not enabled, this parameter is unavailable.
Agency	Used to delegate FunctionGraph to access other cloud services. For example, an agency is required when FunctionGraph accesses cloud services such as OBS or SMN.
Description	Description of the function.

NOTE

- For details on how to create an agency, see **Creating an Agency**.
- 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.

  Function execution agency: After specifying such an agency, you can obtain a token and AK/SK from the context in the function handler for accessing other cloud services.

Table 2-3 Code information

Parameter	Description
*Runtime	Currently, Python, Node.js, Java, Go, C# (.NET Core), Cangjie, PHP, and Custom runtimes are supported.

Parameter	Description
*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].[file name].[function name]*. Example: **com.xxxxx.exp.Myfunction.myHandler** • For a Go function, the handler must be named in the format of *[plug-in name].[function name]*. The function name must start with an uppercase letter. The handler name can contain a maximum of 128 characters. Example: **function.Handler** • For a C# function, the handler must be named in the format of *[.NET assembly file name]*::*[namespace and class of the handler function]*::*[handler function name]*. Example: **HelloCsharp::Example.Hello::Handler** • For a Cangjie function, the handler must be named in the format of *[Dynamic dependency library name]***.so**. The name can contain a maximum of 128 characters. Example: **libuser_func_test_success.so**.
*Code Entry Mode	Method of entering the function code. For details about the supported code entry modes, see **Table 2-4**.

Table 2-4 Code entry modes

Runtime	Code Entry Mode	Description
Node.js	Edit code inline	Edit code in the code box. For more information, see **Table 2-28**.
	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Python	Edit code inline	Edit code in the code box. For more information, see **Table 2-28**.
	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Java	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.

Runtime	Code Entry Mode	Description
	Upload JAR file	Click **Select File** and upload a local JAR file to FunctionGraph. The JAR file must contain at least one **.class** file. The size of the JAR file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, convert it into a ZIP file, and upload the ZIP file to OBS.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Go	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
C#(.NET Core)	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Custom	Edit code inline	Edit code in the code box. For more information, see **Table 2-28**.

Runtime	Code Entry Mode	Description
	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
PHP	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Cangjie	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.

NOTICE

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

Table 2-5 Environment information

Parameter	Description
Memory (MB)	Set the memory required for running the function on the **Configuration** tab page.
*Initializer	Set this parameter on the **Code** tab page if you have enabled initialization for the function. For all runtimes, the naming rules of function initializers are the same as those of function handlers. For example, for a Node.js, Python, or PHP function, set an initializer name in the format of *[file name].[initialization function name]*.
*Initialization Timeout (s)	Maximum duration the function can be initialized. Set this parameter on the **Configuration** tab page if you have enable initialization for the function. The value ranges from 1s to 300s.
*Execution Timeout (s)	Maximum duration the function can be executed. You can set this parameter on the **Configuration** tab page. Use asynchronous invocation for functions that take longer than 90s. The value ranges from 3s to 900s.

NOTE

- When creating a function, you can retain the default values for the memory, execution timeout, encryption settings, and environment variables. After creating the function, you can modify these settings.
- After a function is created, the default version is latest. Each function has the latest version.
- Functions are billed based on the number of function execution requests and function execution duration. Please set the memory and execution timeout based on service requirements.

Creating a Function Using a Template

FunctionGraph provides built-in function templates for typical scenarios. These templates contain code, configurations, environment, and trigger information. After you choose a template, FunctionGraph automatically loads the function information, enabling you to quickly create functions. For details, see **Creating a Function Using a Template**.

Creating a Function Using a Container Image

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. For details, see **Deploying a Function Using a Container Image**.

2.1.3.4 Environment Variables

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

For example, you can use **getUserData(string key)** in the context class to obtain the values of environment variables and encryption settings in Node.js. For details, see **Developing Functions in Node.js**. For details about how to obtain the values of environment variables in other programming languages, see **Developing Functions**.

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 2048 characters.

2.1.3.5 Creating a Function from Scratch

2.1.3.5.1 Creating an Event Function

Introduction

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 using a blank template, using other templates (by referring to **Creating a Function Using a Template**), or based on an image (by referring to **Deploying a Function Using a Container Image**).

NOTE

When creating a function from scratch, configure the basic and code information based on **Table 2-6**. 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 **Programming Models**.
2. Create a package. For details, see **Creating a Deployment Package**.
3. (Optional) You have created an agency. For details, see **Creating an Agency**.

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 2-6**. The parameters marked with an asterisk (*) are mandatory.

Table 2-6 Basic information

Parameter	Description
*Function Type	● **Event Function**: triggered by triggers. ● **HTTP Function**: 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 **Creating an Agency**. 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.

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

Configuring Code

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

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

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.

2. View basic information.

   After a function is created, the default memory and execution timeout in each runtime are displayed. You can click **Edit** to switch to the **Basic Settings** page and modify **Handler**, **Memory (MB)**, and **Execution Timeout (s)** as required.

   NOTICE

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

2.1.3.5.2 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 Node.js as an example. To use another runtime, simply change the runtime path. The code package path does not need to be changed. For the paths of other runtimes, see **Table 2-7**.
- When a function initiates an HTTP request, the request IP address is dynamic for private network access and fixed for public network access. For more information, contact technical support.

Prerequisites

CAUTION

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

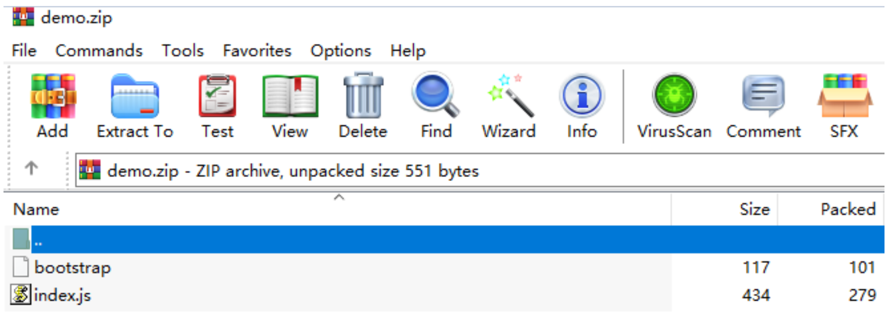
- If the service system and the HTTP functions are in the same VPC, the API can be directly accessed.
- If the service system and the HTTP functions are in different VPCs of a region, connect them using a peering connection. For details, see section "VPC Peering Connection" in the *Virtual Private Cloud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

1. Prepare a Node.js script. A code example is as follows:
```
const http = require('http'); // Import Node.js core module

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

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

console.log('Node.js web server at port 8000 is running..')
```
2. You have prepared a **bootstrap** file as the startup file of the HTTP function.

   Example

   The content of the **bootstrap** file is as follows:
```
/opt/function/runtime/nodejs12.13/rtsp/nodejs/bin/node $RUNTIME_CODE_ROOT/index.js
```
3. Compress the preceding two files into a ZIP package.

Figure 2-3 Compressing files into a ZIP package



demo.zip - ZIP archive, unpacked size 551 bytes

Name	Size	Packed
..		
bootstrap	117	101
index.js	434	279

NOTE

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

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

To use another runtime, change the runtime path by referring to **Table 2-7**. The code package path does not need to be changed.

Table 2-7 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 18	/opt/function/runtime/nodejs18.15/rtsp/nodejs/bin/node
Python 2.7	/opt/function/runtime/python2.7/rtsp/python/bin/python
Python 3.6	/opt/function/runtime/python3.6/rtsp/python/bin/python3
PHP 7.3	/opt/function/runtime/php7.3/rtsp/php/bin/php

Procedure

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

   NOTE

   Only APIG triggers can be created for HTTP functions.

   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

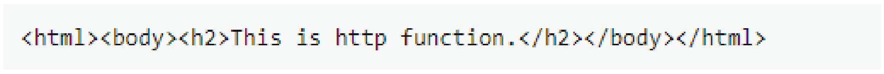
In this example, **Security Authentication** is set to **None**. You need to select an authentication mode based on site requirements.

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

Figure 2-4 Viewing the request result



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

Common Function Request Headers

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

Table 2-8 Default request header fields

Field	Description
X-CFF-Request-Id	ID of the current request
X-CFF-Memory	Allocated memory
X-CFF-Timeout	Function timeout duration
X-CFF-Func-Version	Function version
X-CFF-Func-Name	Function name
X-CFF-Project-Id	Project ID

Field	Description
X-CFF-Package	App to which the function belongs
X-CFF-Region	Current region

2.1.4 Triggering a Function

FunctionGraph applications are composed of functions and event sources. Event sources are cloud services or user-defined applications that publish events, and functions are custom code for processing events.

FunctionGraph functions can be triggered by multiple cloud services. For details, see **Supported Event Sources**. After you configure event sources to trigger your functions, FunctionGraph automatically invokes the functions when events occur.

FunctionGraph supports the following two function invocation modes:

- 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 requests when there are idle resources.

  If a function is invoked asynchronously and is throttled, FunctionGraph automatically retries the throttled event, with an interval between retries. Asynchronous events are queued before they are used to invoke a function.

Table 2-9 lists the function invocation modes for supported event sources.

Table 2-9 Function invocation modes

Event Source	Invocation Mode
SMN	Asynchronous invocation
APIG	Synchronous invocation
OBS	Asynchronous invocation
DIS	Asynchronous invocation
Timer	Asynchronous invocation
LTS	Asynchronous invocation
Cloud Trace Service (CTS)	Asynchronous invocation
DMS for Kafka	Asynchronous invocation

2.1.5 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 **Test Management**.

  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 **Logs** tab page, you can query all logs and their contexts, and view only error logs by selecting **Show error logs only**. You can also enable LTS to manage function logs. For details, see **Querying Logs**.

Downloading Logs

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

NOTE

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

2.1.6 Monitoring

2.1.6.1 Function Monitoring

FunctionGraph is interconnected with Cloud Eye, allowing you to view function metrics without the need for any configurations.

Function Metrics

FunctionGraph presents the monitoring information of all functions or of a single function created using your account.

- Account-level metrics

  The **Dashboard** page allows you to view metrics of all functions. The metrics include invocation count, execution duration (including the maximum, average, and minimum durations), invocation errors, and throttled invocations. For details, see **Dashboard**.

- Function-level metrics

  FunctionGraph differentiates the metrics of a function by version, allowing you to query the metrics of a specific function version.

  View the metrics on the **Monitoring** tab page of a function. The metrics include invocation count, execution duration (including the maximum, average, and minimum durations), invocation errors, and throttled invocations. For details, see **Viewing Function Metrics**.

– Creating an alarm rule

Create an alarm rule for a function to report metrics to Cloud Eye for better monitoring.

On the **Monitoring** tab page of the function details page, click **Create Alarm Rule**. Select a metric and set an alarm policy. After the alarm rule is created, click **View details** in the displayed message to view details of the alarm rule on the Cloud Eye console. On the **Alarm Rules** page, click **View Graph** in the **Operation** column of the alarm rule to view function metrics.

2.1.6.2 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 2-10 Monitoring metrics supported by FunctionGraph

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

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period of Raw Data (Minute)
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
failRate	Error Rate	Percentage of invocation errors to the total invocations Unit: %	0% ≤X≤ 100%	Functions	5
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	Concurrency	Number of concurrent requests during function invocation. Unit: Count	≥ 0 counts	Functions	1
reservedinstancenum	Reserved Instances	Number of reserved instances for function invocation. Unit: Count	≥ 0 counts	Functions	1

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period of Raw Data (Minute)
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
systemErrorCount	System Errors	Number of errors in function requests that result in failed executions. Unit: Count	≥ 0counts	Functions	5
functionErrorCount	Function Error Count	Number of syntax and execution errors. Unit: Count	≥ 0counts	Functions	5
payPerUseInstance	Elastic Instances	Number of elastic instances. Unit: Count	≥ 0 counts	Functions	5

Dimensions

Key	Value
package-functionname	*App name*-*Function name* Example: default-myfunction_Python
projectId	Project ID of the tenant

2.1.7 Versions and Aliases

Version Control

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.

For details about version management, see **Managing Versions**.

Creating a Function Alias

You can create an alias for a specific function version. If you need 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.

For details about alias management, see **Aliases**.

2.1.8 Relationship Between FunctionGraph and Other Services

FunctionGraph is interconnected with other cloud services to implement different functions.

Cloud Eye

Cloud Eye is a multi-dimensional monitoring platform that monitors resources such as Elastic Cloud Servers (ECSs) and bandwidth. With Cloud Eye, you can track the resource usage and status of services running on the cloud platform, receive alarm notifications about errors, and react to changes immediately to keep your services running smoothly.

FunctionGraph is interconnected with Cloud Eye to monitor the execution of functions and workflows in real time.

- For details about the function metrics that can be monitored by Cloud Eye, see **Function Monitoring**.
- For details on how to view function metrics, see **Function Monitoring**.

CTS

CTS records the operations performed on the resources under your account. The operation records can help you perform security analysis, track resource changes, and locate faults. To store operation records for a longer time, subscribe to OBS and synchronize the records to it in real time.

- FunctionGraph operations that can be recorded by CTS

  With CTS, you can record operations associated with FunctionGraph for later query, audit, and backtrack. **Table 2-11** lists the FunctionGraph operations that can be recorded by CTS.

Table 2-11 FunctionGraph operations

Operation	Resource Type	Event Name
Creating a function	Function	CreateFunction
Deleting a function	Function	DeleteFunction
Modifying function information	Function	ModifyFunctionMeta-data
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
Executing a function	Function instance	invokeFunction

- Auditing logs

  After CTS is enabled, the system starts recording operations on cloud resources. The CTS console stores operation records of the last seven days. For details, see section "Querying Real-Time Traces" in the *Cloud Trace Service User Guide*.

2.2 Getting Started

2.2.1 Creating and Initializing a Function

This section describes how to create and initialize a HelloWorld function.

Creating a Function

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

Step 2 Click **Create Function**.

Step 3 Configure the function information by performing the following steps:

1. Configure the basic information according to **Table 2-12**. The parameters marked with an asterisk (*) are mandatory.

Table 2-12 Basic information

Parameter	Description
Template	This example uses no template. To use a template, see **Creating a Function Using a Template**.
*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. Enter **HelloWorld**.
Function Type	– **Event Function**: 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. ■ Only APIG triggers can be created for HTTP functions.
*Region	Select a region where you will deploy your code.
Agency	An agency is required if FunctionGraph accesses other cloud services. For details on how to create an agency, see **Creating an Agency**. No agency is required if FunctionGraph does not access any cloud services. For this example, select **Use no agency**.
*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, Python, and PHP only.

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.

– Function execution agency: After specifying such an agency, you can obtain a token and AK/SK from the context in the function handler for accessing other cloud services.

2. Configure the code information according to **Table 2-13**. The parameters marked with an asterisk (*) are mandatory.

Table 2-13 Code information

Parameter	Description
*Runtime	Select the runtime of the function. In this example, select **Node.js 6.10**.
*Handler	For a Node.js function, the handler must be named in the format of *[file name].[function name]*, which must contain a period (.). Enter **index.handler**, which indicates that the file name is **index.js** and the function name is **handler**.
*Code Entry Mode	Select **Edit code inline**, and use the displayed sample code.

NOTICE

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

NOTE

The sample code enables you to obtain test events and print test event information. The code is as follows:

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

Step 4 Confirm the function configuration and billing information, and click **Create Now**.

NOTE

After the function is created, the memory is 128 MB and the execution timeout is 3s.

----End

Initializing the Function

If initialization is enabled for a function, a specified initializer will be invoked to initialize the function, and then a handler will be invoked to process requests. For details on how to create a function, see **Creating 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 created function.

Step 3 Choose **Configuration** > **Lifecycle** and enable **Initialization**.

Table 2-14 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.
Function 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. ● Ensure that the function initializer and handler are in the same file.

NOTE

- Set the initializer in the same way as the handler. For example, for a Node.js or Python function, set an initializer name in the format of *[file name].[initialization function name]*.
- For details on how to configure the code information, see **Table 2-3**.

Step 4 Click **Save**.

----End

Testing the Function

For details on how to configure test events and test functions, see **Test Management**.

Setting an Alias

For details about how to set an alias, see **Aliases**.

2.2.2 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.

Applications Scenarios

- 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.

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.

Procedure

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

Step 2 Click the name of the created function.

Step 3 Choose **Configuration** > **Configure Async Notification**. On the displayed page, click **Edit** next to **Asynchronous Notification Policy**.

Step 4 Set the parameters based on Table 2-15.

Table 2-15 Parameter description

Parameter	Description
Asynchronous 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** to complete the configuration.

----End

Configuration Description

For details about how to set the target for asynchronous invocation, see **Table 2-16**. The following shows an example:

```
{
      "timestamp": "2020-08-20T12:00:00.000Z",
   "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 2-16 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.

2.2.3 Configuring VPC Access

Functions can be configured to access resources in your Virtual Private Clouds (VPCs) or to access the Internet using elastic IP addresses bound through source network address translation (SNAT).

Prerequisites

Specify an agency with VPC administrator permissions for the function. For details, see **Creating an Agency**.

Configuring VPC Access

To configure VPC access for a function, perform the following steps:

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

Step 2 Create a function based on **Creating and Initializing a Function**.

Step 3 Click the name of the created function.

Step 4 Choose **Configuration** > **Network**, enable **VPC Access**, and configure VPC by referring to **Table 2-17**.

Table 2-17 Configuring VPC access

Parameter	Description
VPC	Mandatory. Select the VPC to be accessed by the function. For details on how to create a VPC and subnet, see section "Creating a VPC with a Subnet" in the *Virtual Private Cloud User Guide*.
Subnet	Mandatory. Select the subnet to be accessed by the function. All functions of a tenant in a project can be bound to a maximum of four subnets. (Each project has a unique 32-digit project ID, which is allocated when your account is created. The project IDs of your account and IAM user are the same.)
Domain Name	Optional. Enter one or more private domain names of the VPC so that the function can use them to access resources in this VPC. ● For details about how to create a private domain name, see section "Creating a Private Domain Name" in the *Domain Name Service User Guide*. ● Functions can resolve only domain names of the A record set type. For details about how to add a record set, see section "Record Set Types and Configuration Rules" in the *Domain Name Service User Guide*.
VPC CIDR Block	Optional. You can enter the VPC CIDR block used in the code to check whether it conflicts with FunctionGraph's VPC CIDR block. The blocks are separated by semicolons (;) and the number of blocks cannot exceed 5.

Parameter	Description
Security Group	Mandatory. Select a security group. The inbound and outbound rules of the security group are displayed below. Configure the rules by referring to section "Adding a Security Group Rule" in the *Virtual Private Cloud User Guide*. ● Inbound rule: Set **Action** to **Allow**, **Protocol & Port** to **ICMP**, and the minimum range for **Source** to the VPC CIDR block selected for the function. For example, if the VPC CIDR block of the function is **192.168.*x.x*/24**, add an inbound rule with **Allow** for **Action**, **ICMP** for **Protocol & Port**, and **192.168.*x.x*/24** for **Source**. ● Outbound rule: Set **Action** to **Allow**.

Step 5 Click **Save**.

----End

Example

The following is an example of configuring VPC access to connect to a Distributed Cache Service (DCS) Redis instance.

Step 1 On the DCS console, view the VPC and subnet of the DCS Redis instance. If no Redis instances are available, create one.

Step 2 On the FunctionGraph console, configure VPC access for the function as follows:

- **VPC Access**: Enable VPC access.
- **VPC**: Select the VPC queried in **Step 1**.
- **Subnet**: Select the subnet queried in **Step 1**.

Step 3 After the configuration is complete, use the following code to access the Redis instance:

```
# -*- coding:utf-8 -*-
import redis

def handler (event, context):
    r = redis.StrictRedis(host='192.168.1.143', port=6379, db=0)
    r.set('product', 'FunctionGraph')
    print r.keys('*')
    print r.get('product')
```

----End

Accessing the Internet from a VPC

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 section "Creating a VPC with a Subnet" in the *Virtual Private Cloud User Guide*.
2. You have obtained an elastic IP address according to section "Assigning an EIP" in the *Elastic IP User Guide*.

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 section "Creating a NAT Gateway" in the *NAT Gateway User Guide*.

Step 3 On the NAT gateway details page, click **Add SNAT Rule**, specify rule information, and click **OK**.

----End

2.2.4 Using an SMN Trigger

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

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

Prerequisites

- You have created an SMN topic, for example, **smn-test**. For details, see section "Creating a Topic" in the *Simple Message Notification User Guide*.
- You have created a function on FunctionGraph. For details, see **Creating and Initializing a Function**.

Creating an SMN Trigger

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

Step 2 Click the **HelloWorld** function.

Step 3 On the HelloWorld function details page, choose **Configuration** > **Triggers**.

Step 4 Click **Create Trigger**.

Step 5 Set the following parameters:

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

Step 6 Click **OK**.

NOTE

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

----End

Publishing a Message to Trigger the Function

On the SMN console, publish a message to the **smn-test** topic. For details, see section "Publishing a Text Message" in the *Simple Message Notification User Guide*.

Table 2-18 describes the parameters required for publishing a message. After the message is published, the function is automatically triggered.

Table 2-18 Parameters required for publishing a message

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

Viewing the Execution Result

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

Step 2 Click the **HelloWorld** function.

Step 3 On the function details page, choose **Monitoring** > **Logs** to view function logs.

----End

2.2.5 Using an APIG (Dedicated) Trigger

This section describes how to create an APIG trigger and use the generated API to invoke a function.

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

Prerequisites

You have created an API group, for example, **APIGroup_test**. For details, see section "Creating an API Group" in the *API Gateway User Guide*.

Creating an APIG Trigger

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

Step 2 Click **Create Function**.

Step 3 Set the following parameters:

- **Template**: Select **Create from scratch**.
- **Function Name**: Enter a function name, for example, **apig**.
- **Enterprise Project**: Select **default**.

- **Agency**: Select **Use no agency**.
- **Runtime**: Select **Python 2.7**.

Step 4 Click **Create Now**.

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**.

Step 7 Configure the trigger information.

Table 2-19 Trigger information

Parameter	Description
Trigger Type	Select **API Gateway (Dedicated Gateway)**.
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 provides the environment management function, 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 high security authentication. This authentication mode is recommended. For details, see **App authentication**. ● **IAM**: IAM medium security authentication. This mode grants access to IAM users only. For details, see **IAM Authentication**. ● **None**: No authentication. Access is granted to all users. Select **None**.

Parameter	Description
Protocol	There are two types of protocols: ● HTTP ● HTTPS Select **HTTPS**.
Timeout (ms)	Enter **5000**.

Step 8 Click **OK**.

NOTE

- **URL** indicates the calling address of the APIG trigger.
- 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.

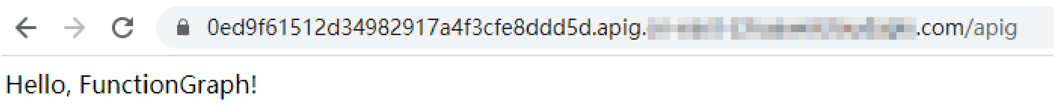
----End

Invoking the Function

Step 1 Enter the URL of the APIG trigger in the address bar of a browser, and press **Enter**.

Step 2 After the function is executed, check the execution result, as shown in **Figure 2-5**.

Figure 2-5 Returned result



0ed9f61512d34982917a4f3cfe8ddd5d.apig.[redacted].com/apig

Hello, FunctionGraph!

----End

Viewing the Execution Result

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

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

Step 3 On the function details page, choose **Monitoring** > **Logs** tab to view function logs.

----End

2.2.6 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.

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

Prerequisites

Before creating an OBS trigger, make sure you have prepared the following:

- You have created a function on FunctionGraph. For details, see **Creating and Initializing a Function**.
- You have created an OBS bucket, for example, **obs_cff**. For details, see section "Creating a Bucket" in the *Object Storage Service User Guide*.

Creating an OBS Trigger

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

Step 2 Click the **HelloWorld** function.

Step 3 Choose **Configuration** > **Triggers** and click **Create 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. For details, see section "Uploading a File" in the *Object Storage Service User Guide*.

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

Step 2 Click the **HelloWorld** function.

Step 3 On the function details page, choose **Monitoring** > **Logs** to view function logs.

----End

2.2.7 Using a DIS Trigger

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

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

Prerequisites

Before creating a DIS trigger, make sure you have prepared the following:

- You have created a function on FunctionGraph. For details, see **Creating and Initializing a Function**.
- You have created a DIS stream, for example, **dis-function**.

Setting an Agency

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

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

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

Step 2 Click the **HelloWorld** function.

Step 3 Choose **Configuration** > **Permissions**, and change the agency to **serverless-trust** created in **Creating an Agency**.

Step 4 Click **Save**.

----End

Creating a DIS Trigger

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

Step 2 Click the **HelloWorld** function.

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

Step 4 Set the following parameters:

- **Trigger Type**: Select **Data Ingestion Service (DIS)**.
- **Stream Name**: Select a DIS stream, for example, **dis-function**.
- **Max. Fetch Bytes**: Maximum volume of data that can be fetched in each request. Only the records smaller than this value will be fetched. The value ranges from 1 KB to 4 MB.
- **Starting Position**: Specify a position in the specified stream from which to start reading data.
  - **TRIM_HORIZON**: Data is read from the earliest valid records that are stored in the partition.

  - **latest**: Data is read just after the most recent record in the partition. This setting ensures that you always read the latest data.
- **Pull Period**: Set a period for pulling data from the stream.
- **Serial Data Processing**: If this option is selected, FunctionGraph pulls data from the stream only after previous data is processed. If this option is not selected, FunctionGraph pulls data from the stream as long as the pull period ends.

Step 5 Click **OK**.

----End

Configuring a DIS Event to Trigger the Function

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

Step 2 Click the **HelloWorld** function.

Step 3 On the function details page, click the **Code** tab and select **Configure Test Event**.

Step 4 Set the parameters described in **Table 2-20** and click **Save**.

Table 2-20 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 **Data Ingestion Service (DIS)** to use the built-in DIS event template.
Event Name	The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **dis-123test**.
Event data	The system automatically loads the built-in DIS event template, which is used in this example without modifications. The code in this template is as follows:

NOTE

The event template is as follows:

```
{
    "ShardID": "shardId-0000000000",
    "Message": {
        "next_partition_cursor":
"eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzLXN3dGVzdCIsInBhcnRpdGlvbi
1pZCI6InNoYXJkSWQtMDAwMDAwMDAwMCIsImN1cnNvci10eXBlIjoiVFJJTV9IT1JJWk9OIiwi
c3RhcnRpbmctc2VxdWVuY2UtbnVtYmVyIjoiNCJ9LCJnZW5lcmF0ZVRpbWVzdGFtcCI6MTUwO
TYwNjM5MjE5MX0",
        "records": [
            {
                "partition_key": "shardId_0000000000",
                "data": "d2VsY29tZQ==",
                "sequence_number": "0"
            },
            {
                "partition_key": "shardId_0000000000",
                "data": "dXNpbmc=",
                "sequence_number": "1"
            },
            {
                "partition_key": "shardId_0000000000",
                "data": "RnVuY3Rpb25TdGFnZQ==",
                "sequence_number": "2"
            },
            {
                "partition_key": "shardId_0000000000",
                "data": "c2VydmljZQ==",
                "sequence_number": "3"
            }
        ],
        "millis_behind_latest": ""
    },
    "Tag": "latest",
    "StreamName": "dis-swtest"
}
```

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

----End

2.2.8 Using a Timer Trigger

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

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

Prerequisites

You have created a function on FunctionGraph. For details, see **Creating and Initializing a Function**.

Creating a Timer Trigger

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

Step 2 Click the **HelloWorld** function.

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

Step 4 Set the following parameters:

- **Trigger Type**: Select **Timer**.
- **Timer Name**: Enter a timer name, for example, **Timer**.
- **Rule**: Set a fixed rate or a cron expression.
  - **Fixed rate**: The function is triggered at a fixed rate of minutes, hours, or days. You can set a fixed rate from 1 to 60 minutes, 1 to 24 hours, or 1 to 30 days.
  - **Cron expression**: The function is triggered based on a complex rule. For example, you can set a function to be executed at 08:30:00 from Monday to Friday. For more information, see **Cron Expression for a Function Timer Trigger**.
- **Enable Trigger**: Choose whether to enable the timer trigger.
- **Additional Information**: The additional information you configure will be put into the **user_event** field of the timer event source. For details, see **Supported Event Sources**.

Step 5 Click **OK**.

----End

Viewing the Execution Result

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

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

Step 2 Click the **HelloWorld** function.

Step 3 On the function details page, choose **Monitoring** > **Logs** to view function logs.

----End

2.2.9 Using an LTS Trigger

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

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

Prerequisites

- You have created a function on FunctionGraph. For details, see **Creating and Initializing a Function**.
- You have created a log group, for example, LogGroup1. For details, see section "Creating a Log Group" in the *Log Tank Service User Guide*.
- You have created a log stream, for example, LogStream1. For details, see section "Creating a Log Stream" in the *Log Tank Service User Guide*.

Creating an LTS Trigger

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

Step 2 Click the **HelloWorld** function.

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

Step 4 Set the following parameters:

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

Step 5 Click **OK**.

----End

Configuring an LTS Event to Trigger the Function

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

Step 2 Click the **HelloWorld** function.

Step 3 On the function details page, click the **Code** tab and select **Configure Test Event**.

Step 4 Set the parameters described in **Table 2-21** and click **Save**.

Table 2-21 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 **Log Tank Service (LTS)** and use the built-in LTS event template.
Event Name	The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **lts-123test**.
Event data	The system automatically loads the built-in LTS event template, which is used in this example without modifications.

NOTE

The event template is as follows:

```
{
    "lts": {
        "data":
"ewogICAgICAgICJsb2dzIjpbewogICAgICAgICAgICAgICAgIm1lc3NhZ2UiOiIyMDE4LTA2LTI2Lz
E4OjQwOjUzIFtJTkZdIFtjb25maWcuZ286NzJdIFN1Y2Nlc3NmdWxseSBsb2FkZWQgZ2VuZXJh
bCBjb25maWd1cmF0aW9uIGZpbGVcXHJcXG4iLAogICAgICAgICAgICAgICAgInRpbWUiOjE1M
zAwMDk2NTMwNTksCiAgICAgICAgICAgICAgICAiaG9zdF9uYW1lIjoiZWNzLXRlc3RhZ2VudC5
ub3ZhbG9jYWwiLAogICAgICAgICAgICAgICAgImlwIjoiMTkyLjE2OC4xLjk4IiwKICAgICAgICAgIC
AgICAgICJwYXRoIjoidXNyL2xvY2FsL3RlbGVzY29wZS9sb2cvY29tbW9uLmxvZyIsCiAgICAgICAg
ICAgICAgICAibG9nX3VpZCI6IjY2M2Q2OTMwLTc5MmQtMTFlOC04YjA4LTI4NmVkNDg4Y2U3
MCIsCiAgICAgICAgICAgICAgICAibGluZV9ubyI6NjE1CiAgICAgICAgICAgIH0sCiAgICAgICAgICA
gIHsKICAgICAgICAgICAgICAgICJtZXNzYWdlIjoiMjAxOC0wNi0yNi8xODo0MDo1MyBbV1JOXS
BbY29uZmlnLmdvOjgyXSBUaGUgcHJvamVjdElkIG9yIGluc3RhbmNlSWQgb2YgY29uZmlnLm
pzb24gaXMgbm90IGNvbnNpc3RlbnQgd2l0aCBtZXRhZGF0YSwgdXNlIG1ldGFkYXRhLlxcbiIsCi
AgICAgICAgICAgICAgICAidGltZSI6MTUzMDAwOTY1MzA1OSwKICAgICAgICAgICAgICAgICJob
3N0X25hbWUiOiJlY3MtdGVzdGFnZW50Lm5vdmFsb2NhbCIsCiAgICAgICAgICAgICAgICAiaXAi
OiIxOTIuMTY4LjEuOTgiLAogICAgICAgICAgICAgICAgInBhdGgiOiIvdXNyL2xvY2FsL3RlbGVzY2
9wZS9sb2cvY29tbW9uLmxvZyIsCiAgICAgICAgICAgICAgICAibG9nX3VpZCI6IjY2M2Q2OTMwL
Tc5MmQtMTFlOC04YjA5LTI4NmVkNDg4Y2U3MCIsCiAgICAgICAgICAgICAgICAibGluZV9ubyI
6NjE2CiAgICAgICAgICAgIH0sCiAgICAgICAgICAgIHsKICAgICAgICAgICAgICAgICJtZXNzYWdlIjo
iIEluIGNvbmYuanNvbiwgcHJvamVjdElkIGlzIFtdLCBpbnN0YW5jZUlkIGlzIFtdLiBNZXRhRGF0YS
BpcyB7NDU0MzI5M2EtNWIyYy00NGM0LWI3YTAtZGUyMThmN2YyZmE2IDYyODBlMTcwYm
Q5MzRmNjBhNGQ4NTFjZjVjYTA1MTI5ICB9XFxyXFxuIiwKICAgICAgICAgICAgICAgICJ0aW1lIjo
xNTMwMDA5NjUzMDU5LAogICAgICAgICAgICAgICAgImhvc3RfbmFtZSI6ImVjcy10ZXN0YWd
lbnQubm92YWxvY2FsIiwKICAgICAgICAgICAgICAgICJpcCI6IjE5Mi4xNjguMS45OCIsCiAgICAgI
CAgICAgICAgICAicGF0aCI6Ii91c3IvbG9jYWwvdGVsZXNjb3BlL2xvZy9jb21tb24ubG9nIiwKICAg
ICAgICAgICAgICAgICJsb2dfdWlkIjoiNjYzZDY5MzAtNzkyZC0xMWU4LThiMGEtMjg2ZWQ0OD
hjZTcwIiwKICAgICAgICAgICAgICAgICJsaW5lX25vIjo2MTcKICAgICAgICAgICAgfQogICAgICAgI
CAgICBdLAogICAgICAgICJvd25lciI6ICI2MjgwZTE3MGJkOTM0ZjYwYTRkODUxY2Y1Y2EwNTEy
OSIsCiAgICAgICAgImxvZ19ncm91cF9pZCI6ICI5N2E5ZDI4NC00NDQ4LTExZTgtOGZhNC0yOD
ZlZDQ4OGNlNzAiLAogICAgICAgICJsb2dfdG9waWNfaWQiOiAiMWE5Njc1YTctNzg0ZC0xMW
U4LTlmNzAtMjg2ZWQ0ODhjZTcwIgogICAgICAgIH0="
    }
}
```

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

----End

2.2.10 Using a CTS Trigger

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

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

Prerequisites

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

Creating a CTS Trigger

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

Step 2 Click **Create Function**.

Step 3 Set the following parameters:

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

Step 4 Click **Create Now**.

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**.

Step 7 Configure the trigger information.

Table 2-22 Trigger information

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

Parameter	Description
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.

NOTE

A maximum of 10 services, each with up to 10 operations, can be added for each CTS trigger. The total number of operations cannot exceed 100. For details, see **Supported Services and Operations**.

Step 8 Click **OK**.

----End

Configuring a CTS Event to Trigger the Function

Step 1 On the function details page, click the **Code** tab and select **Configure Test Event**.

Step 2 Set the parameters described in **Table 2-23** and click **Save**.

Table 2-23 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 **Cloud Trace Service (CTS)** and use the built-in CTS event template.
Event Name	Enter an event name, for example, **cts-test**.
Event data	The system automatically loads the event data in the CTS event template. You can modify the event data as required.

Step 3 Click **Test**.

----End

2.2.11 Using a Kafka Trigger

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

After a Kafka trigger is used, FunctionGraph periodically polls for new messages in a specific topic in a Kafka instance and passes the messages as input parameters to invoke functions. For details about the DMS for Kafka event source, see **Supported Event Sources**.

Prerequisites

Before creating a Kafka trigger, make sure you have prepared the following:

- You have created a function on FunctionGraph. For details, see **Creating and Initializing a Function**.
- You have enabled VPC access for the function. For details, see **Configuring VPC Access**.
- You have created a Kafka instance. For details, see section "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 *Distributed Message Service for Kafka User Guide*.

Creating a Kafka Trigger

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

Step 2 Click the **HelloWorld** function.

Step 3 Choose **Configuration** > **Triggers** and click **Create 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 the 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 enable VPC access, see **Configuring VPC Access**.

----End

Configuring a Kafka Event to Trigger the Function

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

Step 2 Click the **HelloWorld** function.

Step 3 On the function details page, click the **Code** tab and select **Configure Test Event**.

Step 4 Set the parameters described in **Table 2-24** and click **Save**.

Table 2-24 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 **DMS (for Kafka)** to use the built-in Kafka event template.
Event Name	The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **kafka-123test**.
Event data	The system automatically loads the built-in Kafka event template, which is used in this example without modifications.

NOTE

The event template is as follows:

```
{
    "event_version": "v1.0",
    "event_time": 1576737962,
    "trigger_type": "KAFKA",
    "region": "xx-xxxx-1",
    "records": [{
        "messages": [
            "kafka message1",
            "kafka message2",
            "kafka message3",
            "kafka message4",
            "kafka message5"
        ],
    "instance_id": "81335d56-b9fe-4679-ba95-7030949cc76b",
    "topic_id": "topic-test"
    }]
}
```

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

----End

2.2.12 Using a DMS (for RabbitMQ) Trigger

This section describes how to create a DMS (for RabbitMQ) trigger for a function. Currently, only the fan-out exchange mode is supported. After a DMS (for RabbitMQ) trigger is used, FunctionGraph periodically polls for new messages in a specific topic bound to the exchange of a RabbitMQ instance and passes the messages as input parameters to invoke functions. For details about RabbitMQ triggers, see **Supported Event Sources**.

Prerequisites

- You have created a function on FunctionGraph. For details, see **Creating an Event Function**.
- You have enabled VPC access for the function. For details, see **Configuring VPC Access**.
- A RabbitMQ instance has been created. For details, see **Buying a RabbitMQ Instance**.
- A virtual host, exchange, and queue have been created.
  a. To create a virtual host, see **Creating a RabbitMQ Virtual Host**.
  b. To create an exchange, see **Creating a RabbitMQ Exchange**.
  c. To create a queue, see **Creating a RabbitMQ Queue**.
  d. An exchange-queue binding has been configured. For details, see **Binding a RabbitMQ Exchange** and **Binding a RabbitMQ Queue**.

  NOTE

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

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

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

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

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

Creating a RabbitMQ Trigger

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

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

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

Step 4 Set the following parameters:

- **Trigger Type**: Select **DMS (for RabbitMQ)**.
- ***Instance**: Select a RabbitMQ instance.

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

Step 5 Click **OK**.

----End

NOTE

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

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

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

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

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

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

Step 5 Set the parameters described in **Table 2-25** and click **Save**.

Table 2-25 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 **DMS (for RabbitMQ)** to use the built-in RabbitMQ event template.
Event Name	The event name can contain 1 to 25 characters and must start with a letter and end with a letter or digit. Only letters, digits, underscores (_), and hyphens (-) are allowed. For example, **kafka-123test**.
Test event	The system automatically loads the built-in RabbitMQ event template, which is used in this example without modifications.

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

----End

2.2.13 Using an Open-Source Kafka Trigger

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

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

Prerequisites

Before creating a trigger, make sure you have prepared the following:

- You have created a function on FunctionGraph. For details, see **Creating and Initializing a Function**.
- You have enabled VPC access for the function. For details, see **Configuring VPC Access**.

Creating an Open-Source Kafka Trigger

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

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

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

Step 4 Set the following parameters:

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

Step 5 Click **OK**.

NOTE

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

----End

Enabling an Open-Source Kafka Trigger

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

NOTE

If a trigger cannot be disabled, contact technical support.

Configuring an Open-Source Kafka Event to Trigger the Function

Step 1 On the **Code** tab page, click **Configure Test Event**.

Step 2 Set the parameters described in **Table 2-26** and click **Create**.

Table 2-26 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 (OPENSOURCEKAFKA)** to use the built-in open-source 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**.
Test event	The system automatically loads the built-in Kafka event template, which is used in this example without modifications.

Step 3 Click **Test**. The function test result is displayed.

----End

2.3 Basic Operations

2.3.1 Dashboard

The **Dashboard** page presents your function resources, resource quota, and account-level metrics.

Using the Dashboard Page

Step 1 Log in to the FunctionGraph console, and choose **Dashboard** in the navigation pane.

Step 2 View your function resources, code storage, and quota information.

Step 3 To create a function, click **Create Function**. For details, see **Creating a Function**.

Step 4 To access a related cloud service console, click the icon in the **Related Services** area.

Step 5 View the monthly statistics of your functions.

NOTE

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

Step 6 View the account-level function metrics in the **Metrics** area.

NOTE

FunctionGraph presents the following metrics: invocations, top 10 functions by invocation, errors, top 10 functions by error, duration, and throttles.

----End

Function Metrics

Table 2-27 describes the function metrics.

Table 2-27 Function metrics

Metric	Unit	Description
Invocations	Count	Total number of invocation requests, including invocation errors and throttled invocations. In asynchronous invocation mode, the count starts only when a function is executed in response to a request.
The 10 Functions with the Most Invocations	-	Top 10 functions by invocation in the last day, last 3 days, or a custom period.
Duration	ms	**Maximum Duration**: the maximum duration all functions are executed at a time within a period. **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.

Metric	Unit	Description
Errors	Count	Number of times that your functions failed with error code **200** being returned. Errors caused by function syntax or execution are also included.
The 10 Functions with the Most Errors	-	Top 10 functions by error in the last day, last 3 days, or a custom period.
Throttles	Count	Number of times that FunctionGraph throttles your functions due to the resource limit.

2.3.2 Function Management

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.

An app groups functions as a service logic unit for easier management. You can create multiple functions under an app.

Editing 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 view and edit the code on the console. **Table 2-28** describes the menus of the inline editor.

Table 2-28 Menus of the inline editor

Menu	Description
File	Creates files and folders. You can create files, create folders based on templates, and close all files.
Edit	Edits function code. You can perform the undo, edit, or comment operation.
Find	Finds and replaces code.
Go to	Goes to a certain location in the code. You can go to a specified line, brackets, or the next problematic line.
View	Provides common functions. You can view the command palette and change the display theme.

Menu	Description
Test	Tests a function online, and displays the test result, digest, and logs.

NOTE

- 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.
- If you edit code in Go, compress the compiled file in ZIP format, 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 **function.so**, set the handler to **function.Handler** according to **Table 2-29**.
- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.
- For details about editing code inline, see **Using CloudIDE Inline**.

Modifying Function Code

After a function is created, the default version is latest. Each function has the latest version. You can modify a function based only on its latest version.

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 (Optional) On the function details page, select the latest version.

Step 4 On the **Code** or **Configuration** tab page, modify the code information according to **Table 2-29** and **Table 2-30**.

Table 2-29 Code information

Parameter	Description
Handler	● For a Node.js or Python function, the handler must be named in the format of *[file name].[function name]*, which must contain a period (.). Enter **myfunction.handler**, which indicates that the file name is **myfunction.js** (**myfunction.py** for a Python function and **myfunction.java** for a Java function) and the function name is **handler**. ● For a Java function, the handler must be named in the format of *[package name].[file name].[function name]*, for example, **com.***xxxxx***.exp.Myfunction.myHandler**. ● For a Go function, the handler must be named in the format of *[plug-in name].[function name]*, for example, **function.Handler**. The function name must start with an uppercase letter and the handler name can contain a maximum of 128 characters. ● For a C# function, the handler must be named in the format of *[.NET assembly file name]*::*[namespace and class of the handler function]*::*[handler function name]*, for example, **HelloCsharp::Example.Hello::Handler**. ● For a Cangjie function, the handler must be named in the format of *[Dynamic dependency library name]***.so**. The name can contain a maximum of 128 characters. Example: **libuser_func_test_success.so**.

Parameter	Description
Initializer	You can enable function initialization on the **Code** tab page after creating a function. 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. For details on how to create an initialization function, see **Initializing the Function**.
Dependencies	Third-party software packages required by the function. You can manage your dependencies by uploading them to FunctionGraph. If the dependencies are too large, upload them through OBS. For more information, see **Dependent Libraries**. **NOTE** Except your private dependencies, FunctionGraph provides some common dependencies, which you can choose when creating a function.
Code Entry Mode	Method of entering the function code. For details about the supported code entry modes, see **Table 2-30**.

Table 2-30 Code entry modes

Runtime	Code Entry Mode	Description
Node.js	Edit code inline	Edit code in the code box. For more information, see **Table 2-31**.
	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.

Runtime	Code Entry Mode	Description
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Python	Edit code inline	Edit code in the code box. For more information, see **Table 2-31**.
	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Java	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload JAR file	Click **Select File** and upload a local JAR file to FunctionGraph. The size of the JAR file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, convert it into a ZIP file, and upload the ZIP file to OBS.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.

Runtime	Code Entry Mode	Description
Go	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
C#(.NET Core)	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
PHP	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Custom	Edit code inline	Edit code in the code box. For more information, see **Table 2-28**.

Runtime	Code Entry Mode	Description
	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.
Cangjie	Upload ZIP file	Click **Select File** and upload a local code package to FunctionGraph. The size of the ZIP file to be uploaded cannot exceed 50 MB. If it exceeds 50 MB, upload the ZIP file using an OBS bucket.
	Upload file from OBS	Paste the link URL of the OBS bucket storing a code ZIP file.

NOTICE

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

Table 2-31 Menu description of the inline editor

Menu	Description
File	Creates files and folders. You can create files, file templates, and Python modules, and close all files.
Edit	Edits function code. You can undo or redo your editing, delete or join lines, add comments, fold and unfold lines, and sort lines in ascending or descending order.
Find	Finds and replaces code.

Menu	Description
Go to	Goes to a certain location in the code. You can go to a specified line, brackets, or the next problematic line.
View	Provides common functions. You can view the command palette and change the display theme.
Test	Tests a function online, and displays the test result, digest, and logs.

NOTE

- 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.
- If you edit code in Go, compress the compiled file in ZIP format, 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 **function.so**, set the handler to **function.Handler** according to **Table 2-29**.
- When creating a ZIP file, place the handler file under the **root** directory to ensure that your code can run normally after being decompressed.

Step 5 Click **Save**.

----End

Exporting a Function

You can export the created functions and their code and configurations.

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

Step 2 Click a function name.

Step 3 On the displayed function details page, choose **Operation** > **Export function** in the upper right corner.

NOTE

- A user can export only one function at a time.
- The exported function resource package cannot exceed 50 MB.
- The name of the exported function resource package is in the format of *function name* +*MD5 value of function code*.zip.
- The exported function resource package does not include alias information.
- If a function is disabled or enabled, all versions of the function will be disabled or enabled.

----End

Importing a Function

You can import a ZIP file that contains configurations and code into FunctionGraph.

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

Step 2 Click **Import Function** in the upper right corner.

Step 3 On the displayed page, select an existing app or define a new app, and then upload a ZIP file.

NOTICE

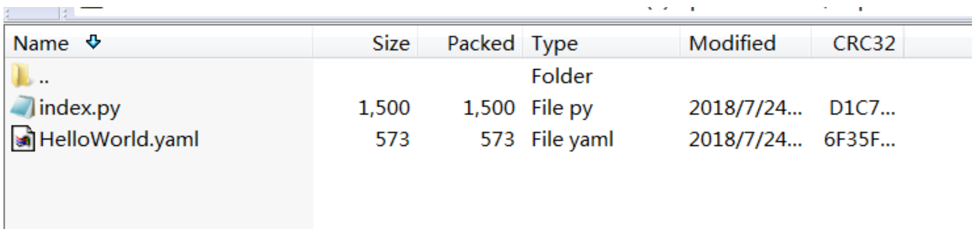
If the ZIP file to be uploaded contains sensitive information (such as account passwords), encrypt the sensitive information to prevent leakage.

Step 4 Click **OK**.

NOTE

- You can import only one function at a time.
- The resource package to be imported must be a ZIP package that contains a YAML configuration file and code file, as shown in **Figure 2-6**. The package cannot exceed 10 MB.

Figure 2-6 Resource package



Name	Size	Packed	Type	Modified	CRC32
..			Folder		
index.py	1,500	1,500	File py	2018/7/24...	D1C7...
HelloWorld.yaml	573	573	File yaml	2018/7/24...	6F35F...

- The resource package must contain only one YAML configuration file.
- Fields in the YAML file must meet the same requirements as those configured when you create a function.
- The code file is located through the **codeFilename** field in the YAML file.
- For the **Resource** field in the YAML file, you can specify only one function.
- When a dependency package is used, if the code in this package can be pulled from OBS, the import is successful. Otherwise, the import fails.
- You can import an exported resource package again, but must specify a unique function name. The version of the imported function is **latest** by default.

The following shows an example YAML configuration file. The fields in this file are described in **Table 2-32**.

```
HcCrmTemplateVersion: v2
Resources:
  HelloWorld:
    Type: HC::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: Node.js6.10
      CodeType: inline
      CodeFileName: index.js
      DependencyPkg: ""
      Description: ""
      MemorySize: 128
      Timeout: 3
      Version: latest
      Environment:
        Variables: {}
```

Table 2-32 YAML file description

Parameter	Description
HcCrmTemplateVersion	Fixed value: **v2**.
Resources	Function details.
Type	Fixed value: **HC::Serverless::Function**.
Properties	Function information.
Handler	Handler of the function.
Runtime	Running environment.
CodeType	Code entry mode, which can be **jar**, **zip**, or **inline**.
CodeFileName	Code file name.
DependencyPkg	Dependency address.
Description	Description of the function.
MemorySize	Memory size.
Timeout	Function execution timeout.
Version	Function version.
Environment	Environment variables.

----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 In the **Function List**, locate the row that contains the target function and click **Delete** in the operation column. In the displayed dialog box, enter **DELETE** and click **OK**.

----End

2.3.3 Using CloudIDE to Manage Functions

2.3.3.1 Using CloudIDE Inline

FunctionGraph allows you to edit function code in the same way as managing a project. You can create and edit files and folders. With the inline editor, you can edit function code from scratch, or modify the function code of an uploaded package. In addition, when the editor is displayed in full screen, you can save code modifications, test functions online, and view the test result, digest, and logs.

Managing Folders and Files

Choose **File** from the menu bar, and create files or folders.

Choose **New File** from the drop-down list to create a file with a specified name.

Choose **New From Template** from the drop-down list to create a file with a template.

Select a template, and create a file with a specified name.

To rename a file or folder, right-click and choose **Rename**. To delete the file or folder, right-click and choose **Delete**.

Choose **File** > **New Folder** to create a folder with a specified name.

To create a file under the folder, right-click the folder and choose **New File**. You can also rename and delete folders and create Python modules.

Choose **File** > **Close All Files** to close all files.

You can quickly create a multi-level Python module.

Edit

The **Edit** menu provides options for code editing. Choose **Undo** to cancel the previous operation, choose **Redo** to restore the previous operation, and choose **Cut** to cut the selected code.

In the **Edit** menu, choose **Line** to edit code by line.

In the **Line** submenu, choose **Delete Line** to delete a code line, choose **Delete All Left** to delete the code on the left of the cursor in the current line, choose **Delete All Right** to delete the code on the right of the cursor in the current line, and choose **Join Lines** to merge the current line with the next line.

Choose **Edit** > **Comment** to edit comments.

In the **Comment** submenu, choose **Toggle Line Comment** to open the comment of a code line, choose **Toggle Block Comment** to open the comment of a code block, choose **Add Line Comment** to add a comment to a code line, and choose **Remove Line Comment** to delete the comment of a code line.

Choose **Edit** > **Folding** to expand or collapse code.

In the **Folding** submenu, choose **Unfold** or **Fold** to expand or collapse all code, and choose **Fold Level** *XX* to fold code by level.

Choose **Edit** > **Sort** to sort code.

In the **Edit** menu, you can also trim trailing spaces, transform code to uppercase or lowercase letters, and format documents.

Find and Replace

Choose **Find** from the menu bar to find or replace a keyword.

Choose **Find** > **Find** or **Replace** to find or replace a keyword.

Go to

Choose **Go to** from the menu bar, and go to a specified position in the code.

Choose **Go to** > **Go to Line** to go to a specified code line.

For a large number of code lines, you can choose **Go to Bracket** or **Go to Next Problem (Error, Warning, Info)** to respectively switch to the other bracket of a bracket pair or to the next problematic line.

Online Function Test

After configuring a test event for a function, you can click **Test** in the full screen mode of the editor to test the function, and view the test result, digest, and logs.

Other Common Functions

1. In the **View** menu, choose **Theme** to change the editor theme, and choose **Show Command Palette** to view all commands.
2. The inline editor can autocomplete a keyword when you input only the first several letters.
3. The left pane can be adjusted to expand the working area. To open a local code file, drag it to the editor.

2.3.4 Dependency Management

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.

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:

- **Name**: Enter a dependency name.
- **Runtime**: Select a runtime.
- **Description**: Enter a description for the dependency. This parameter is optional.
- **Upload Mode**: Upload a ZIP file or upload a file from OBS.

Step 4 Click **OK**.

----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 **Select Dependency** dialog box, select dependencies and click **OK**.

NOTE

- You can add a maximum of 20 dependencies for a function.
- Except your private dependencies, FunctionGraph provides some common dependencies, which you can choose when creating a function.

Step 5 After configuring dependencies, click **Save** in the upper right corner.

----End

Deleting a Dependency

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.

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 2-33** and **Table 2-34**. You can import these libraries to your inline code in the same way as you import standard libraries.

Table 2-33 Third-party components integrated with the Node.js runtime

Name	Usage	Version
q	Asynchronous method encapsulation	1.5.1
co	Asynchronous process control	4.6.0
lodash	Common tool and method library	4.17.10
esdk-obs-nodejs	OBS SDK	2.1.5
express	Simplified web-based application development framework	4.16.4

Name	Usage	Version
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 2-34 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.)

  To use functions of third-party libraries, package these libraries and upload them to a specified OBS bucket, and paste the OBS link URL of these libraries when creating a function.

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 = '******'
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**.
5. On the function details page, click the **Code** tab, click **Add**, select the dependency created in **4**, and click **OK**.
6. Click **Save**.

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.

2.3.5 Trigger Management

Enabling or Disabling a Trigger

You can enable or disable triggers as required. Note that SMN, OBS, and APIG triggers cannot be disabled and can only be deleted.

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 Choose **Configuration** > **Triggers**. On the displayed page, locate the row that contains the target trigger, and click **Disable** or **Enable**.

----End

Deleting a Trigger

You can delete triggers that will no longer be used.

Step 1 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 Choose **Configuration** > **Triggers**.

Step 4 On the right of the desired trigger, click **Delete**.

----End

2.3.6 Test Management

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 a function name.

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

Step 4 In the **Configure Test Event** dialog box, configure the test event information according to **Table 2-35**. The parameter marked with an asterisk (*) is mandatory.

Table 2-35 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 2-36**.
*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 2-36 Event template description

Template Name	Description
blank-template	The template event is **{"key": "value"}**, which can be changed based on requirements.
API Gateway (Dedicated Gateway)	Simulates an API Gateway event to trigger your function.
Data Injection Service (DIS)	Simulates a DIS event to trigger your function.
Simple Message Notification (SMN)	Simulates an SMN event to trigger your function.
Object Storage Service (OBS)	Simulates an OBS event to trigger your function.
timer	Simulates a timer event to trigger your function.
Log Tank Service (LTS)	Simulates an LTS event to trigger your function.
Cloud Trace Service (CTS)	Simulates a CTS event to trigger your function.
DMS (for Kafka)	Simulates a Kafka event to trigger your function.

Template Name	Description
Login Security Analysis	Serves as an input for the **loginSecurity-realtime-analysis-python** function template.
Pornographic Image Analysis	Serves as an input for the **porn-image-analysis** function template.
Speech Recognition	Serves as an input for the **voice-analysis** function template.
Image Classification	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, 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 test event, and click **Test**.

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 **Querying 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 2-37**.

Table 2-37 Test event information

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

Step 4 On the **Configure Test Event** page, select **Edit saved test event**. In the **Saved Test Events** list on the left, select the event to be deleted and click **Delete**. For test event configuration, see **Table 2-38**.

Table 2-38 Configuring test event information

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

----End

2.3.7 Version and Alias Management

Managing Versions

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.

Publishing a Version

1. Log in to the FunctionGraph console, and choose **Functions** > **Function List** in the navigation pane.
2. Click the name of the desired function.
3. Select the latest version, click the **Version** tab, and click **Publish New Version**.

4. Set the following information.
   - **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, **v20170819-190658**.
   - **Description**: Enter a description for the version. This parameter is optional.
5. Click **OK**. The system automatically publishes a version. Then you will be redirected to the new version.

   NOTE

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

Deleting a Version

1. Log in to the FunctionGraph console, and choose **Functions** > **Function List** in the navigation pane.
2. Click the name of the desired function.
3. On the **Version** tab page of the latest version, select the version to delete.

   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.

2.3.8 Aliases

Introduction

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**.
   - **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 a maximum of 10 aliases for a function.

Modifying an Alias

1. Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click the function to be configured to go to the function details page.
3. On the **Aliases** tab page of the latest version, select the alias to modify.
4. Modify the alias information, and click **OK**.

Deleting an Alias

1. Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click the function to be configured to go to the function details page.
3. On the **Aliases** tab page of the latest version, select the alias to delete.
4. Click **OK** to delete the version.

2.3.9 Function Monitoring

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

Viewing Function Metrics

FunctionGraph differentiates the metrics of a function by version, allowing you to query the metrics of a specific function version.

Procedure

1. Log in to the FunctionGraph console, and choose **Functions** > **Function List** in the navigation pane.
2. Click the name of the desired function.
3. Choose **Monitoring** > **Metrics**, select an interval (5 minutes, 15 minutes, or 1 hour), 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.

Function metrics

Table 2-39 describes the function metrics.

Table 2-39 Function metrics

Metric	Unit	Description
Invocations	Count	Total number of invocation requests, including invocation errors and throttled invocations. In asynchronous invocation mode, 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	N/A	Numbers of concurrent requests and reserved instances.

Viewing Monitoring Graphs

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, 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, select a function version or alias, and choose **Monitoring** > **Metrics**.

Step 4 Click **Create Alarm Rule**.

Step 5 In the **Create Alarm Rule** dialog box, set alarm parameters and click **Next**.

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

----End

Function Metrics

Table 2-40 lists the function metrics that can be monitored by Cloud Eye.

Table 2-40 Function metrics

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

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

Metric	Display Name	Description	Unit	Upper Limit	Lower Limit	Recommended Threshold	Value Type	Dimension
functionErrorCount	Function Error Count	Number of function errors that occur during invocation	Count	-	0	-	float	package-functionname
memoryUsed	Memory	Memory used by the function	MB	-	0	-	float	package-functionname
duration_p500	Duration (P50)	P50 execution duration of the function	ms	-	0	-	float	package-functionname
duration_p800	Duration (P80)	P80 execution duration of the function	ms	-	0	-	float	package-functionname
duration_p950	Duration (P95)	P95 execution duration of the function	ms	-	0	-	float	package-functionname
duration_p990	Duration (P990)	P990 execution duration of the function	ms	-	0	-	float	package-functionname

Metric	Display Name	Description	Unit	Upper Limit	Lower Limit	Recommended Threshold	Value Type	Dimension
duration_p999	Duration (P999)	P999 execution duration of the function	ms	-	0	-	float	package-functionname
instances	Instances	Number of reserved instances for function invocation.	N/A	-	0	-	int	package-functionname
systemErrorCount	System Errors	Number of system errors that occur during function invocation	Count	-	0	-	int	package-functionname
reservedinstancenum	Reserved Instances	Number of reserved instances	N/A	-	0	-	int	package-functionname
functionCost	Resource Usage	Resources used by the function (Memory x Duration)	MB	-	0	-	float	package-functionname

Querying Logs

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

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 Select the version or alias of the function, and choose **Monitoring** > **Logs** > **Request Logs**.

Step 4 Set the search criteria.

----End

Downloading Logs

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 Choose **Monitoring** > **Logs** > **Request Logs**.

Step 4 Select a version and time range, and click **Download Log**.

NOTE

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

----End

2.3.10 Reserved Instance Management

Overview

FunctionGraph provides on-demand and reserved instances.

- On-demand instances are created and released by FunctionGraph based on actual function usage. When receiving requests to call functions, FunctionGraph automatically allocates execution resources to the requests.
- Reserved instances (RIs) can be created and released by you as required. After you create RIs for a function, FunctionGraph preferentially forwards requests for invoking the function to the RIs. If the number of requests exceeds the processing capability of the RIs, FunctionGraph will forward the excessive requests to on-demand instances and automatically allocates execution resources to these requests.

  After RIs are created for a function, the code, dependencies, and initializer of the function are automatically loaded. RIs are always alive in the execution environment and eliminate the influence of cold starts on your services. (Do not execute one-time services using the initializer of reserved instances.)

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**.
4. Set parameters by referring to **Table 2-41**.

   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.

Table 2-41 Basic settings

Parameter	Description
Function Name	Name of the current function.
Type	Select **Version** or **Aliases**.
Version	Set this parameter when you select **Version** for **Type**.
Alias	Set this parameter when you select **Aliases** for **Type**.
Reserved Instances	Minimum number of instances. Max.: **1000**. FunctionGraph reserves the specified number of instances for the function. These instances will always run unless you change **Min. Instances** to **0**.
Idle Mode	This mode saves costs as CPU resources are not used when reserved instances are not invoked.

NOTE

- Reserved instances cannot be configured for both a function alias and the corresponding version. For example, if the alias of the latest version is 1.0 and reserved instances have been configured for this version, no more instances can be configured for alias 1.0.
- After the idle mode is enabled, reserved instances are initialized and the mode change needs some time to take effect. You will still be billed at the price of reserved instances for non-idle mode in this period.
- If the function concurrency is greater than the number of reserved instances, the excess requests will be allocated to on-demand instances, which involve a cold start.

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

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 2-41**, and then click **Add Policy**.

2. Set parameters by referring to **Table 2-42**.

Table 2-42 Scheduled scaling policy parameters

Parameter	Description
Policy Name	Policy name.
Cron Expression (UTC)	Set this parameter by referring to **Cron expression rule**.
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.
Reserved 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.
4. To modify the reserved instance policy, click **Edit** in the **Operation** column. Then modify or add scheduled scaling policies.
5. To delete a reserved instance policy under a function version or alias, click **Delete** in the **Operation** column.
6. To view concurrent instances, click a quantifier in the reserved instance policy list, and click a scheduled scaling policy name.

NOTE

Multiple scheduled policies can be configured. For example, the number of reserved instances at 08:00 and 21:00 is updated to 100 and 10 respectively.

2.3.11 Tag Configuration

Overview

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

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

Applications Scenarios

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

Prerequisites

You have enabled Tag Management Service (TMS), or the pre-defined tags cannot be used. For details, see section "Permissions" in the *Tag Management Service User Guide*.

Adding a Tag

1. Log in to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click the function to be configured to go to the function details page.
3. Choose **Configuration** > **Tags**, and click **Add Tag**.
4. Add a tag key and value that meet the following rules:
   - Each tag consists of a key-value pair. Each key must be unique and have only one value.
   - Each function can have a maximum of 20 tags.

   NOTE

   If your organization has configured tag policies for FunctionGraph, add tags to functions based on the policies. If a tag does not comply with the tag policies, function creation may fail. Contact your administrator to learn more about tag policies.

   Table 2-43 Tag naming rules

Parameter	Rules
Tag key	● Cannot be empty. ● Cannot start with **_sys_** or a space, or end with a space. ● Can contain UTF-8 letters, digits, spaces, and these characters: _.:=+-@ ● Must be unique and cannot exceed 128 characters.
Tag value	● Can be an empty string. ● Can contain UTF-8 letters, digits, spaces, and these characters: _.:/=+-@ ● Max. 255 characters.

5. Click **OK**.

   Tag values can be modified but tag keys cannot.

Searching for Functions by Tag

1. Return to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. In the search box, select the **Tag** filter, and then select one or more key-value pairs.
3. (Optional) Add more filters, such as runtime and package type.
4. View the search result in the function list.

2.3.12 Single-Instance Multi-Concurrency Management

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.

Applications Scenarios

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 2-44 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 2-45** and click **Save**.

Table 2-45 Parameter description

Parameter	Description
Max. Requests per Instance	Number of concurrent requests supported by a single instance. The default value is 1. The value ranges from 1 to 1,000.
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.

2.4 Advanced Operations

2.4.1 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.

Creating a Function

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

NOTE

The **context-class-introduction** template for Python 2.7 is used as an example. You can also select other templates.

3. After you select a function template, the built-in code and configurations of the template are automatically loaded. The **Create Function** page is displayed.
4. Set **Function Name** to **context**, select a created agency, retain default values for other parameters, and click **Create Function**.

   NOTE

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

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

5. Set the parameters based for your service requirements.

Triggering a Function

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

2.4.2 Deploying a Function Using a Container Image

Overview

You can package your container images complying with the Open Container Initiative (OCI) standard, and upload them to FunctionGraph. The images will be loaded and run by FunctionGraph. Unlike the code upload mode, you can use a custom code package, which is flexible and reduces migration costs. You can create event and HTTP functions by using a custom image.

The following features are supported:

- **Downloading images**

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

  Encryption settings and environment variables are supported. For details, see **Environment Variables**.
- **Attaching external data disks**

  External data disks can be attached. For details, see **Mounting a File System**.
- **Reserved instances**

  Reserved instances are supported. For details, see **Reserved Instance Management**.

NOTE

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

Prerequisites

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

Procedure

1. Log in to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. On the **Function List** page, click **Create Function** in the upper right corner.
3. Select **Container Image**. For details, see **Table 2-46**.

Table 2-46 Parameter description

Parameter	Description

*Function Type	Select a function type.
	Event Function: triggered by triggers.
	HTTP Function: 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.
	● When creating an event function, create an HTTP server to implement a handler (method: **POST**, path: **/invoke**) and an initializer (method: **POST**, path: **/init**).
	● When calling a function using APIG, **isBase64Encoded** is valued **true** by default, indicating that the request body transferred to FunctionGraph is encoded using Base64 and must be decoded for processing.
	● The function must return characters strings by using the following structure.
	`{`
	`    "isBase64Encoded": true|false,`
	`    "statusCode": httpStatusCode,`
	`    "headers": {"headerName":"headerValue",...},`
	`    "body": "..."`
	`}`

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

Parameter	Description
*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 of the container. The folder path can only be **/** and cannot be created or modified. The path will be **/** by default if not specified. ● **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 the **SWR Admin** permission. To create an agency, see **Creating an Agency**.

NOTE

- The total length of **CMD**, **Args**, and **Working Dir** cannot exceed 5,120 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 type can be public or private. For details, see section "Setting Image Attributes" in the *Software Repository for Container User Guide*.
- 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}`);
});
```

2.4.3 Mounting a File System

Scenarios

You can mount file systems to your function to provide scalable file storage. The function can then read and write data in the file systems as it would do in local file systems. Each file system can be shared by different functions and instances. You only need to specify information such as file systems and function access paths.

NOTICE

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

1. 111, 445, 2049, 2051, 2052, and 20048
2. Another three ports for Ubuntu. To obtain the port numbers, run the following command:
```
rpcinfo -p|grep mountd|grep tcp
```

For details, see section "What Resources Does SFS Occupy?" in the *Scalable File Service User Guide*.

FunctionGraph supports the following types of file systems:

- SFS

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

- SFS Turbo

  SFS Turbo 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.

- ECS

  A directory on an ECS is specified as a shared file system (see **Creating an NFS Shared Directory on ECS**) 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 execution space of each function can be expanded in addition to the **/tmp** directory (512 MB).
- A file system can be shared by multiple functions.
- ECS compute resources can be dynamically expanded and existing ECS storage capability can be used to achieve stronger computing performance.

  NOTE

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

Creating an Agency

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

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

Mounting an SFS File System

Setting an Agency

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

If no agencies are available, create one in IAM.

Mounting a File System

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

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

NOTE

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

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

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

Select an SFS file system and set the access path.

NOTICE

The function access path can contain a maximum of two levels. **/mnt** and **/home** are existing paths. You are advised to set an access path starting with **/mnt** or **/home**. If you set the path to **/mnt** or **/home**, message "failed to mount exist system path" will be displayed.

Mounting an SFS Turbo File System

Setting an Agency

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

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 section "File System Management" in the *Scalable File Service User Guide*.
2. Enable VPC access by referring to **Configuring VPC Access** and enter the VPC and subnet obtained in **1**.

Mounting an SFS Turbo File System

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

Mounting an ECS Shared Directory

Specifying an Agency

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

Configuring VPC Access

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

Set the acquired VPC and subnet for the function.

Mounting an ECS Directory

Enter a shared directory and function access path.

Follow-up Operations

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

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

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

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

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

Step 3 In the search result, click **Configure**. The configuration page is displayed. Set the parameters as follows:

- **Region**: Select the same region as the created VPC and file system. For details about how to create a VPC and file system, see **Configuring VPC Access** and section "Creating a File System" in the *Scalable File Service User Guide*.
- **Project**: Use **default**.
- **Function Name**: Enter a custom name.
- **Agency**: Select an agency with the file system, VPC, and APIG permissions. For details about how to create an agency, see **Creating an Agency**.
- **Enterprise Project**: Select an enterprise project as required.
- **Environment Variables**: **access_log_path** indicates the log file address. Set this parameter to **/home/test/access_log.log**.

  NOTE

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

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

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

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

In addition, add the public dependency **Jinja2-2.10**. For details, see **How Do I Add a Dependency to a Function?**

NOTE

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

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

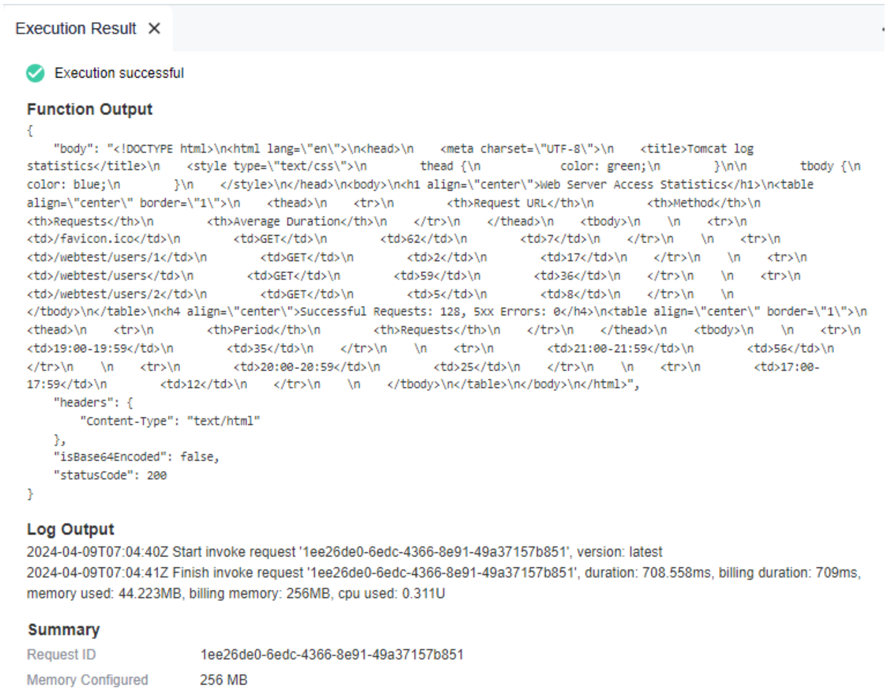
Step 7 Choose **Disk Mounting**, click **Mount File System**, and select **SFS Turbo**.

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

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

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

Figure 2-7 Test result



Function Output

```
{
    "body": "<!DOCTYPE html>\n<html lang=\"en\">\n<head>\n    <meta charset=\"UTF-8\">\n    <title>Tomcat log
statistics</title>\n    <style type=\"text/css\">\n        thead {\n            color: green;\n        }\n\n        tbody {\n
color: blue;\n        }\n    </style>\n</head>\n<body>\n<h1 align=\"center\">Web Server Access Statistics</h1>\n<table
align=\"center\" border=\"1\">\n    <thead>\n    <tr>\n        <th>Request URL</th>\n        <th>Method</th>\n
<th>Requests</th>\n        <th>Average Duration</th>\n    </tr>\n    </thead>\n    <tbody>\n    \n    <tr>\n
<td>/favicon.ico</td>\n        <td>GET</td>\n        <td>62</td>\n        <td>7</td>\n    </tr>\n    \n    <tr>\n
<td>/webtest/users/1</td>\n        <td>GET</td>\n        <td>2</td>\n        <td>17</td>\n    </tr>\n    \n    <tr>\n
<td>/webtest/users</td>\n        <td>GET</td>\n        <td>59</td>\n        <td>36</td>\n    </tr>\n    \n    <tr>\n
<td>/webtest/users/2</td>\n        <td>GET</td>\n        <td>5</td>\n        <td>8</td>\n    </tr>\n    \n
</tbody>\n</table>\n<h4 align=\"center\">Successful Requests: 128, 5xx Errors: 0</h4>\n<table align=\"center\" border=\"1\">\n
<thead>\n    <tr>\n        <th>Period</th>\n        <th>Requests</th>\n    </tr>\n    </thead>\n    <tbody>\n    \n    <tr>\n
<td>19:00-19:59</td>\n        <td>35</td>\n    </tr>\n    \n    <tr>\n        <td>21:00-21:59</td>\n        <td>56</td>\n
</tr>\n    \n    <tr>\n        <td>20:00-20:59</td>\n        <td>25</td>\n    </tr>\n    \n    <tr>\n        <td>17:00-
17:59</td>\n        <td>12</td>\n    </tr>\n    \n    </tbody>\n</table>\n</body>\n</html>",
    "headers": {
        "Content-Type": "text/html"
    },
    "isBase64Encoded": false,
    "statusCode": 200
}
```

Log Output

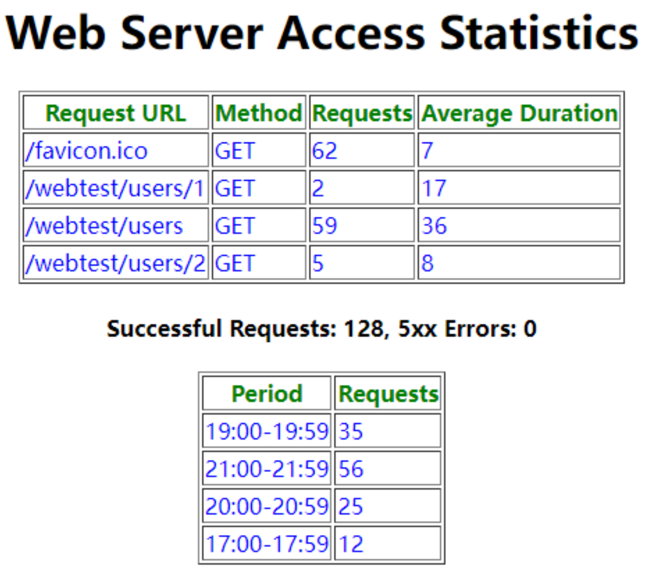
```
2024-04-09T07:04:40Z Start invoke request '1ee26de0-6edc-4366-8e91-49a37157b851', version: latest
2024-04-09T07:04:41Z Finish invoke request '1ee26de0-6edc-4366-8e91-49a37157b851', duration: 708.558ms, billing duration: 709ms,
memory used: 44.223MB, billing memory: 256MB, cpu used: 0.311U
```

Summary

Request ID	1ee26de0-6edc-4366-8e91-49a37157b851
Memory Configured	256 MB

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

Figure 2-8 Results display



Web Server Access Statistics

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

Successful Requests: 128, 5xx Errors: 0

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

----End

2.4.4 Using a 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 plugin 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 2-47 Response header information

Parameter	Description
X-Cff-Request-Id	Request ID.
X-CFF-Access-Key	AK of the account. An agency must be configured for the function if this variable is used.
X-CFF-Auth-Token	Token of the account. An agency must be configured for the function if this variable is used.
X-CFF-Invoke-Type	Invocation type of the function.
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 2-47**.

  $RUNTIME_API_ADDR is a system environment variable. For more information, see **Table 2-48**.

  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 2-48 Runtime environment variables

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

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: '$EVENT_DATA'"
  # Put response
  curl -X POST
"http://$RUNTIME_API_ADDR/v1/runtime/invocation/response/$REQUEST_ID"
-d "$RESPONSE"
done
```

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

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

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

```
package main

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    return nil
}

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

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

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

    return data
}

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

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

    return data
}
```

Table 2-49 describes the environment variables used in the preceding code.

Table 2-49 Environment variables

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

2.4.5 Environment Variables

You can use environment variables to configure which directory to install files in, where to store outputs, and how to store connection and logging settings. These settings are decoupled from the application logic, so you do not need to update your function code when you change the settings.

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.

Configuring Environment Variables

1. Log in to the FunctionGraph console. In the navigation pane, choose **Functions** > **Function List**.
2. Click the name of the function to be configured. The function details page is displayed.
3. Choose **Configuration** > **Environment Variables** and click **Edit Environment Variable**.

2.4.6 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.

Applications Scenarios

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

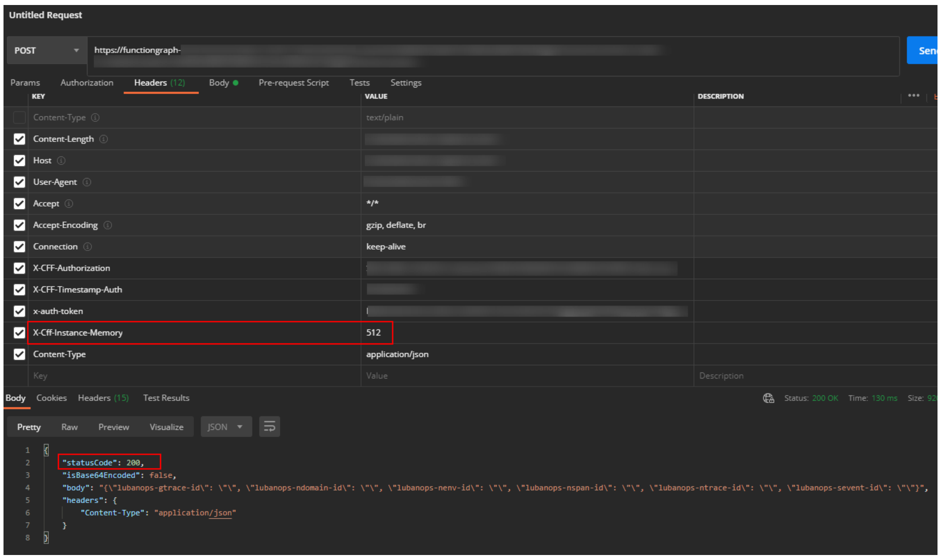
Step 1 Log in to the FunctionGraph console, choose **Functions** > **Function List** in the navigation pane, and click the name of the created function.

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

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

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

Figure 2-9 Adding a request header and calling the function



Untitled Request

POST https://functiongraph-[redacted]

Params | Authorization | Headers (12) | Body | Pre-request Script | Tests | Settings

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

Body | Cookies | Headers (15) | Test Results — Status: 200 OK Time: 130 ms

Pretty | Raw | Preview | Visualize | JSON

```
{
    "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**, **4096**, **8192**, or **10240**, error code FSS.0406 will be returned when the API is called. You only need to reset the memory value.

----End

2.4.7 Configuring Heartbeat Function

Overview

A heartbeat function detects the following exceptions of your functions:

- Deadlock
- Memory overflow
- Network errors

After you configure a heartbeat function, FunctionGraph sends a heartbeat request to your function instance every 5s. If the response is abnormal, FunctionGraph terminates the function instance.

The heartbeat request timeout is 3s. If no response is returned for six consecutive times, the function instance will be stopped.

Constraints

1. **Heartbeat detection is available for Java functions.**
2. The heartbeat function entry must be in the same file as your function handler.

Java heartbeat function format:

```
public boolean heartbeat() {
    // Custom detection logic
    return true
```

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

Procedure

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

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

Step 3 Click the **Configuration** tab and choose **Advanced Settings**.

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

Table 2-50 Heartbeat function configuration

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

Step 5 Click **Save**.

----End

2.5 Appendixes

2.5.1 Creating an Agency

Services of the cloud platform interwork with each other, and some cloud services are dependent on other services. To delegate a cloud service to access other services and perform resource O&M, create an agency for the service.

Create an agency based on section "Creating an Agency" in the *Identity and Access Management User Guide* and set parameters as follows:

- For **Agency Type**, select **Cloud service**.
- For **Cloud Service**, select **FunctionGraph**.
- For **Validity Period**, select **Unlimited**.
- For **Permissions**: Select permissions based on requirements. Select **Region-specific projects**. The following takes **VPC Administrator** and **DNS ReadOnlyAccess** permissions as examples.

Table 2-51 Example of agency permissions

Policy Name	Scenario
VPC Administrator	Users with the **VPC Administrator** permissions can perform any operations on all cloud resources of the VPC. For example, to configure cross-VPC access, you must specify an agency with VPC management permissions.
DNS ReadOnlyAccess	Users with the **DNS ReadOnlyAccess** permissions can read DNS resources. For example, to invoke a DNS API to resolve private domain names, you must specify an agency with the permissions to read DNS resources.

Related Operations

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

CAUTION

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

2.5.2 Creating an NFS Shared Directory on ECS

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

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

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

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

        The EulerOS 2.0 SP3 YUM configuration is as follows:

        ```
        [base]
        name=EulerOS-2.0SP3 base
        baseurl=http://repo.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

Parameter description

name: repository name

baseurl: URL of the repository

- HTTP-based network address: **http://path/to/repo**
- Local repository address: **file:///path/to/local/repo**

gpgcheck: indicates whether to enable the GNU privacy guard (GPG) to check the validity and security of RPM package resources. **0**: The GPG check is disabled. **1**: The GPG check is enabled. If this option is not specified, the GPG check is enabled by default.

3. Save the configurations.
4. Run the following command to clear the cache:

```
yum clean all
```

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

```
yum install nfs-utils
```

iii. Create a shared directory.

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

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

NOTE

- The preceding configuration is used to share the **/sharedata** directory with other servers in the **192.168.0.0/24** subnet.
- After the preceding command is run, run the **exportfs -v** command to view the shared directory and check whether the setting is successful.

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

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

v. Create another shared directory.

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

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

Restart the NFS service.

```
service nfs restart
```

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

```
exportfs -rv
```

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

Run the following command:

```
systemctl enable rpcbind
```

– **Ubuntu**

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

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

ii. Create a shared directory.

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

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

NOTE

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

iii. Start the NFS service.

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

NOTE

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

iv. Create another shared directory.

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

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

Restart the NFS service.

```
service nfs restart
```

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

```
exportfs -rv
```

2. **Windows**

1. Install the NFS server.

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

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

2. Enable the NFS function.

   – In the case of WinNFSd, visit the **WinNFSd GitHub website**.

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

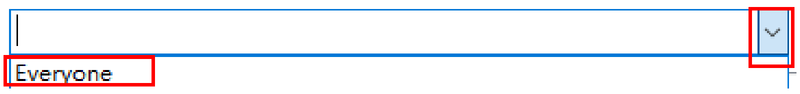
     ii. Set the sharing and read/write permissions on the **nfs** file.

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

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

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

Figure 2-10 Adding Everyone



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

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

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

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

v. Run the following command. For details about more commands, visit the third-party website.

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

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

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

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

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

NOTE

The shared directory format can be referenced as **D:\share -public -name:nfs**, which means to set the permission on the **share** folder to **public** and define an alias **nfs**.

iv. Click **OK**.

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

NOTE

Run the following command in Linux to mount the directory and check whether the file sharing is successful:

```
mount -t nfs -o nolock 192.168.xxx.xxx:/nfs /mnt
```

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

2.5.3 Cron Expression 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 invoking a function every *N* time units, as shown in **Table 2-52**.

Table 2-52 Example expressions

Expression	Meaning
@every 30m	Triggers a function every 30 minutes.
@every 1.5h	Triggers a function every 1.5 hours.
@every 2h30m	Triggers a function every 2.5 hours.

- Standard format

  The format is "*seconds minutes hours day-of-month month day-of-week*". *day-of-week* is optional. The fields must be separated from each other using a space. **Table 2-53** describes the fields in a standard cron expression.

Table 2-53 Parameter description

Parameter	Mandatory	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 2-54**.	, - * /
Day-of-week	No	0–6 or Sun–Sat. The value is case-insensitive, as shown in **Table 2-55**. **0** means Sunday.	, - * ? /

Table 2-54 Value description of the month field

Month	Digit	Abbreviation
January	1	Jan
February	2	Feb
March	3	Mar
April	4	Apr

Month	Digit	Abbreviation
May	5	May
June	6	Jun
July	7	Jul
August	8	Aug
September	9	Sep
October	10	Oct
November	11	Nov
December	12	Dec

Table 2-55 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 2-56 describes the special characters that can be used in a cron expression.

Table 2-56 Special character description

Special Character	Meaning	Remarks
*	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.

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

Table 2-57 describes several example cron expressions.

Table 2-57 Example cron expressions

Cron Expression	Meaning
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.

Cron Expression	Meaning
0 0/30 * * * ?	Executes a function every 30 minutes.

3 FAQs

3.1 General FAQs

3.1.1 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.

3.1.2 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.

3.1.3 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**.

3.1.4 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 **Quotas and Usage Restrictions**.

3.1.5 Does FunctionGraph Support Function Versioning?

Yes. For details, see **Version and Alias Management**.

3.1.6 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.

3.1.7 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. For more information, see **Developer Guide**.

Alternatively, use your own dependencies. For more information, see section "dependency management" in the *FunctionGraph User Guide*.

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

If you are prompted insufficient permissions when creating, deleting, modifying, or querying functions and triggers in FunctionGraph as an IAM user, contact the administrator to grant permissions to your user group. If you want to invoke other cloud services, such as OBS, configure an agency with the required permissions. For security purpose, do this by following the principle of least privilege.

3.1.9 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 *
   ```

3.1.10 What Is the Quota of FunctionGraph?

For details about the resource quota of FunctionGraph, see **Quotas and Usage Restrictions**.

3.1.11 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

3.1.12 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.

3.1.13 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.

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

Not currently.

3.1.15 What If the VPC Quota Is Used Up?

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

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

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

3.1.17 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

3.1.18 Can I Change the Runtime?

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

3.1.19 Can I Change a Function's Name?

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

3.1.20 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.

3.1.21 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.

3.1.22 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.

3.1.23 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.

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

The actions listed in **Table 3-1** can be used.

Table 3-1 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

3.1.25 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.

3.1.26 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;
    }
?>
```

3.1.27 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.

Prerequisites

You have enabled asynchronous invocation status persistence.

Procedure

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

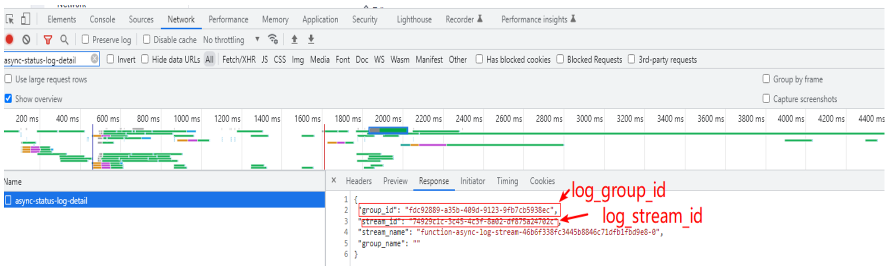
Step 2 On the **Configure Async Notification** page, click **Enable LTS**.

Step 3 Click **Edit** next to **Asynchronous Notification Policy**, and enable **Asynchronous Invocation Status Persistence**.

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 3-1**.

   Figure 3-1 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, expand the log group and find the log stream by ID. Click the log stream name.
3. On the log stream details page, click the gear icon in the upper right.
4. Configure log structuring.
5. Click **Intelligent Extraction**.
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**.

7. Click **Save**.

----End

3.1.28 Can I Enable a Listening Port in a Function to Receive External TCP Requests via EIP?

No. FunctionGraph does not support this feature currently. Functions are about serverless computing, and compute resources are allocated while functions are running. Customizing a listening port is not suitable.

3.2 Function Creation FAQs

3.2.1 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.

3.2.2 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 3-2** describes the rules for packaging a function project.

Table 3-2 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
PHP	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	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	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 1.8	Not supported.	Compress project files into a ZIP file, and ensure that the name of the dynamic library file is consistent with the handler plugin name. For example, if the name of the dynamic library file is **testplugin.so**, set the handler plugin name to **testplugin.Handler**. **Handler** indicates the function handler.	Compress project files into a ZIP file and upload it to an OBS bucket.
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.
C#	Not supported.	Compress project files into a ZIP file. The ZIP file must contain the following files: *Project_name*.**deps.json**, *Project_name*.**dll**, *Project_name*.**runtimeconfig.json**, *Project_name*.**pdb**, and **HC.Serverless.Function.Common.dll**.	Compress project files into a ZIP file and upload it to an OBS bucket.
Custom	Not supported.	Compress project files into a ZIP file. The ZIP file must contain a bootstrap file.	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 PHP project package

```
Example.zip                    Example project package
|--- ext                       Extension library directory
|--- pear                      PHP extension and application repository
|--- index.php                 PHP handler 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
```

- Example directory of a C# project package

```
Example.zip                                  Example project package
|--- fssExampleCsharp2.0.deps.json           File generated after project compilation
|--- fssExampleCsharp2.0.dll                 File generated after project compilation
|--- fssExampleCsharp2.0.pdb                 File generated after project compilation
|--- fssExampleCsharp2.0.runtimeconfig.json  File generated after project compilation
|--- Handler                                 Help file, which can be directly used
|--- HC.Serverless.Function.Common.dll       .dll file provided by FunctionGraph
```

- Example directory of a Cangjie project package

```
fss_example_cangjie.zip                  Example project package
|--- libuser_func_test_success.so        Service function package
```

- Custom

```
Example.zip                              Example project package
|--- bootstrap                           Executable boot file
```

3.2.3 How Does FunctionGraph Isolate Code?

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

3.2.4 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**.

3.3 Trigger Management FAQs

3.3.1 What Events Can Trigger a FunctionGraph Function?

For details, see **Supported Event Sources**.

3.3.2 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;
    }
}
```

3.3.3 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.

3.3.4 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.

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

For details, see **Using an APIG (Dedicated) Trigger**.

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

The request path or request parameters are carried in the input parameters of **event** by default. The input value of an APIG trigger is the event template of the function. For details about the event template, see **Supported Event Sources**.

How to obtain

The following are the formats for obtaining the request path and request parameters from the **event** object:

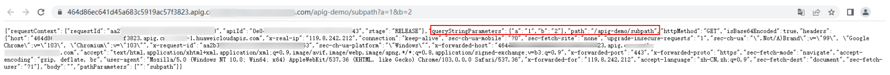
Format of the request path: event['path']

Format of the request parameter: event['queryStringParameters']['*Parameter name*']

How to call

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

You can also call the API by adding request parameters: **https://464d86ec641d45a683c5919ac57f3823.apig.projectID.com/apig-demo/subpath?a=1&b=2**



464d86ec641d45a683c5919ac57f3823.apig.c[redacted].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.[redacted]
[redacted].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"}}
```

Table 3-3 Request path and parameters

Parameter	Description
queryStringParameters	Request parameter. Parameters following the URL in a GET request. When a GET request is initiated, the parameters are transferred in the form of a URL string. That is, the character string after the question mark (?) are the request parameters and are separated by **&**.
path	Request path. Access address of the API.

3.3.7 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.

3.4 Dependency Management FAQs

3.4.1 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.

3.4.2 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.

3.4.3 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.

3.4.4 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 3-4** and **Table 3-5**. You can import these components to your inline code in the same way as you import standard libraries.

Table 3-4 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 3-5 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.

3.4.5 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?**

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

1. Package third-party libraries into a ZIP package by referring to dependency creation.
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.

3.4.7 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 3-6 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.

3.4.8 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**.

3.5 Function Execution FAQs

3.5.1 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.

3.5.2 Which Steps Are Included in Function Execution?

Function execution includes two steps:

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

3.5.3 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.

3.5.4 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.

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

The returned information about a function contains the maximum memory consumed. For more information, see **SDK APIs** in the *FunctionGraph Developer Guide*. Alternatively, check the memory usage in the execution result.

3.5.6 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.

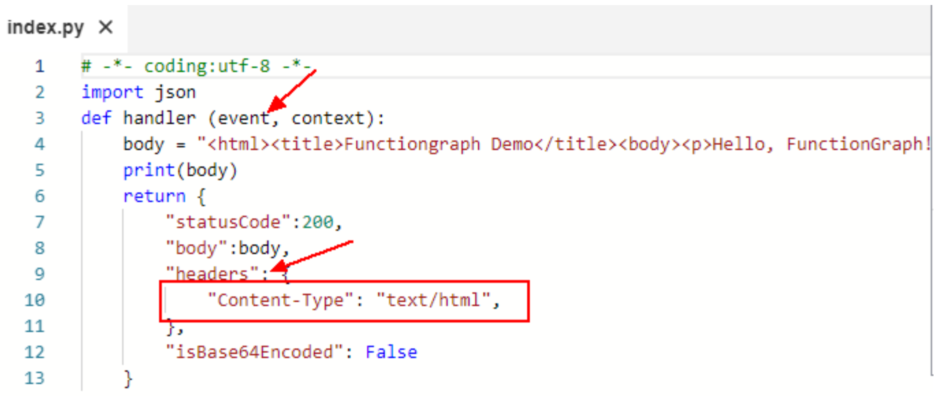
3.5.7 What Do I Do If an Error Occurs When Calling an API?

Rectify the fault by referring to section "Error Codes" in the *FunctionGraph API Reference*. If the fault persists, contact technical support.

3.5.8 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.

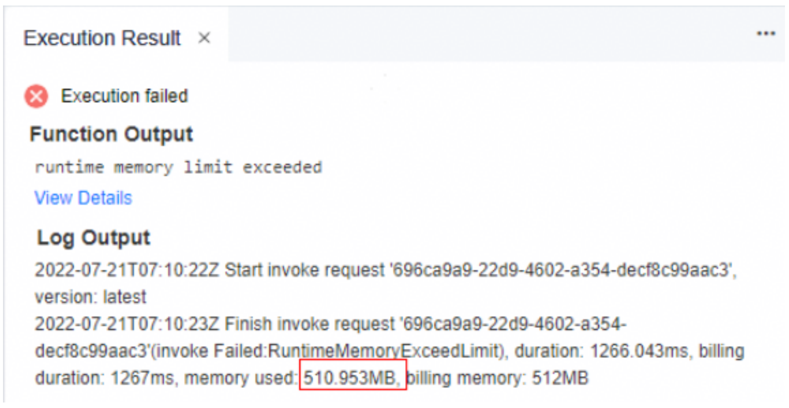
As shown in the following figure, **event** is the first parameter in the function handler, and **headers** is the request header.



```python
# -*- 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
    }
```

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

Check the used memory in the response.



```
runtime memory limit exceeded
```

```
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
```

Figure 3-2 Checking the used memory

3.5.10 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 3-3 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 section "Deploying a Function Using a Container Image" in the *FunctionGraph User Guide*.
3. Check whether the image uses the Linux x86 architecture. Currently, only Linux x86 images are supported.

3.5.11 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.

3.6 Function Configuration FAQs

3.6.1 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 **Environment Variables**.

3.6.2 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.

3.7 External Resource Access FAQs

3.7.1 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. Set the parameters as required and click **Create Function**.
3. After the MySQL function is created, choose **Configuration** > **Environment Variables**, click **Edit Environment Variable**. In the displayed dialog box, add environment variables and enable encryption as required, and click **OK**.

   NOTE

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

3.7.2 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.

3.7.3 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 section "Creating a VPC" in the *Virtual Private Cloud User Guide*.
2. You have obtained an elastic IP address according to section "Assigning an EIP" in the *Elastic IP User Guide*.

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 section "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

3.8 Other FAQs

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

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

3.8.2 Does FunctionGraph Support ZIP Decompiling During Video Transcoding?

No. Please decompile your files before uploading them.