03 Developer Guide

Issue 01

Date 2025-03-14

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Overview

1.1 Function Development

Runtime Overview

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

Supported Runtimes

The Node.js, Java, Python, Go, C#, PHP, Cangjie, and custom runtimes are supported. **Table 1-1** lists the supported runtimes.

NOTE

You are advised to use the latest runtime version.

Table 1-1 Runtime description

Runtime	Supported Version	SDK Download Link
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	**Java SDK** (software package verification file: **fss-java-sdk_sha256**) **NOTE** The Java runtime has integrated with Object Storage Service (OBS) SDKs.

Runtime	Supported Version	SDK Download Link
Go	1.x	**Go 1.x SDK** (software package verification file: **Go SDK_sha256**)
C#	.NET Core 2.1, .NET Core 3.1, .NET Core 6.0, .NET Core 8.0 (only in **ME-Riyadh** and **TR-Istanbul**)	**C# SDK** (software package verification file: **fssCsharp_sha256**)
PHP	7.3 and 8.3	-
Custom	-	-
Cangjie	1.0	-

Runtime Deprecation

For security and sustainable development, FunctionGraph will no longer provide technical support and security updates for some runtimes.

Table 1-2 shows the deprecation policy, which is divided into three stages.

Table 1-2 Deprecation stage description

Stage	Description
Stage 1: Notify customers 180 days in advance	Huawei Cloud will notify you of the runtime deprecation through product notices, runtime deprecation marks, and emails.
Stage 2: Terminate runtimes	Huawei Cloud will no longer provide security patches, feature updates, or technical support for these runtimes. Functions cannot be created using these runtimes. Existing functions can continue to update their code or configurations and run.
Stage 3: 30 days after runtime deprecation	Huawei Cloud will no longer provide security patches, feature updates, or technical support for these runtimes. Functions cannot be created or updated using these runtimes. Existing functions can continue running. You are advised to migrate your functions to the latest supported runtimes for technical support and security updates.

Table 1-3 is our runtime deprecation plan. Runtimes not included in this table are not subject to this deprecation plan.

Table 1-3 Runtime deprecation plan

Runtime	Stage 1	Stage 2	Stage 3
Node.js 6.10	December 15, 2025	June 15, 2026	July 15, 2026
Node.js 8.10	December 15, 2025	June 15, 2026	July 15, 2026
Python 2.7	December 15, 2025	June 15, 2026	July 15, 2026

Third-Party Components Integrated with the Node.js Runtime

Table 1-4 Third-party components integrated with the Node.js runtime

Name	Usage	Version
q	Asynchronous method encapsulation	1.5.1
co	Asynchronous process control	4.6.0
lodash	Common tool and method library	4.17.10
esdk-obs-nodejs	OBS SDK	2.1.5
express	Simplified web-based application development framework	4.16.4
fgs-express	Uses the Node.js application framework to run serverless applications and REST APIs in FunctionGraph and API Gateway. 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

Non-standard Libraries Integrated with the Python Runtime

Table 1-5 Non-standard libraries integrated with the Python Runtime

Library	Usage	Version
dateutil	Date and time processing	2.6.0
requests	HTTP library	2.7.0
httplib2	HTTP client	0.10.3
numpy	Mathematical computation	1.13.1
redis	Redis client	2.10.5
obsclient	OBS client	-
smnsdk	Simple Message Notification (SMN) access	1.0.1

Sample Project Packages

Table 1-6 provides the links for downloading the sample project packages mentioned in this document. You can download the project packages to a local path and upload them when creating functions.

Table 1-6 Download links of the sample project packages

Function	Project Package	Software Package Verification File
Node.js function	fss_examples_nodejs.zip	fss_examples_nodejs.sha256
Python function	fss_examples_python2.7.zip	fss_examples_python2.7_sha256
Java function	fss_example_java8.jar	fss_example_java8_sha256
Go function	fss_examples_go1.8.zip	fss_examples_go1.8_sha256
C# function	fss_example_csharp2.0 and fss_example_csharp2.1	fss_example_csharp2.0_sha256 fss_example_csharp2.1_sha256
PHP function	fss_examples_php7.3.zip	fss_examples_php7.3_sha256

1.2 Supported Event Sources

This section describes the cloud services that can be configured as event sources for your FunctionGraph functions. After you preconfigure the event source mapping, these event sources automatically invoke the relevant function when detecting events.

SMN

Simple Message Notification (SMN) sends messages to email addresses, mobile phones, or HTTP/HTTPS URLs. If you create a function with an SMN trigger, messages published to a specified topic will be passed as a parameter (**SMN example event**) to invoke the function. Then, the function processes the event, for example, publishing messages to other SMN topics or sending them to other cloud services. For details, see **Using an SMN Trigger**.

APIG

API Gateway (APIG) is an API hosting service that helps enterprises to build, manage, and deploy APIs at any scale. With APIG, your function can be invoked through HTTPS by using a custom REST API and a specified backend. You can map each API operation (such as, GET and PUT) to a specific function. APIG invokes the relevant function when an HTTPS request (**APIG example event**) is sent to the API backend. For details, see **Using an APIG Trigger**.

DIS

Data Ingestion Service (DIS) can ingest large amounts of data in real time. You can create a function to automatically poll a DIS stream and process all new data records, such as website click streams, financial transactions, social media streams, IT logs, and location-tracking events (**DIS example event**). FunctionGraph periodically polls the stream for new data records. For details, see **Using a DIS Trigger**.

Timer

You can schedule a timer (**timer example event**) to invoke your code based on a fixed rate of minutes, hours, or days or a cron expression. For details, see **Using a Timer Trigger**.

DMS for Kafka

DMS for Kafka is a message queuing service that provides Kafka premium instances. If you create a Kafka trigger for a function, when a message is sent to a Kafka instance topic, FunctionGraph will retrieve the message and trigger the function to perform other operations. For details, see **Using a Kafka Trigger**.

Cloud Eye

Cloud Eye is a multi-dimensional resource monitoring platform. FunctionGraph is interconnected with Cloud Eye to report metrics, allowing you to view function

metrics and alarm messages through Cloud Eye. For more information about metrics, see **Viewing Function Metrics**.

DMS for RabbitMQ

When a DMS (for RabbitMQ) trigger is used, FunctionGraph periodically polls for new messages in a specific topic bound to the exchange of a RabbitMQ instance and passes the messages as input parameters to invoke functions.

Example Events

- SMN example event

```
{
    "record": [
        {
            "event_version": "1.0",
            "smn": {
                "topic_urn": "urn:smn:{region}:0162c0f220284698b77a3d264376343a:{function_name}",
                "timestamp": "2018-01-09T07:11:40Z",
                "message_attributes": null,
                "message": "this is smn message content",
                "type": "notification",
                "message_id": "a51671f77d4a479cacb09e2cd591a983",
                "subject": "this is smn message subject"
            },
            "event_subscription_urn": "urn:fss:
{region}:0162c0f220284698b77a3d264376343a:function:default:read-smn-message:latest",
            "event_source": "smn"
        }
],
    "functionname": "test",
    "requestId": "7c307f6a-cf68-4e65-8be0-4c77405a1b2c",
    "timestamp": "Wed Nov 15 2017 12:00:00 GMT+0800 (CST)"
}
```

Table 1-7 Parameter description

Parameter	Type	Example Value	Description
event_version	String	1.0	Event version
topic_urn	String	See the example.	ID of an SMN event
type	String	notification	Event type
RequestID	String	7c307f6a-cf68-4e65-8be0-4c77405a1b2c	Request ID. The ID of each request is unique.
message_id	String	a51671f77d4a479cacb09e2cd591a983	Message ID. The ID of each message is unique.
Message	String	this is smn message content	Message content

Parameter	Type	Example Value	Description
event_source	String	smn	Event source
event_subscription_urn	String	See the example.	Subscription ID
timestamp	String	Wed Nov 15 2017 12:00:00 GMT+0800 (CST)	Time when an event occurs

- APIG example event

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

Constraints:

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

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


Table 1-8 Parameter description

Parameter	Type	Example Value	Description
body	String	"{\"test\":\"body\"}"	Actual request in string format
requestContext	Map	See the example.	Request information, including the API gateway configuration, request ID, authentication information, and source
httpMethod	String	GET	HTTP method
queryStringParameters	Map	See the example.	Query strings configured in APIG and their actual values
pathParameters	Map	See the example.	Path parameters configured in APIG and their actual values
headers	Map	See the example.	Complete headers
path	String	/apig-event-template	Complete path
isBase64Encoded	Boolean	True	Default value: **true**

- DIS example event

```
{
    "ShardID": "shardId-0000000000",
    "Message": {
        "next_partition_cursor":
"eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzLXN3dGVzdCIsInBhcnRpdGlvbi1pZCI6InNoYXJkSWQtMDAwMDAwMDAwMCIsImN1cnNvci10eXBlIjoiVFJJTV9IT1JJWk9OIiwic3RhcnRpbmctc2VxdWVuY2UtbnVtYmVyIjoiNCJ9LCJnZW5lcmF0ZVRpbWVzdGFtcCI6MTUwOTYwNjM5MjE5MX0",
        "records": [
            {
                "partition_key": "shardId_0000000000",
                "data": "d2VsY29tZQ==",
                "sequence_number": "0"
            },
            {
                "partition_key": "shardId_0000000000",
                "data": "dXNpbmc=",
                "sequence_number": "1"
            },
            {
                "partition_key": "shardId_0000000000",
                "data": "RnVuY3Rpb25TdGFnZQ==",
```

```
            "sequence_number": "2"
        },
        {
            "partition_key": "shardId_0000000000",
            "data": "c2VydmljZQ==",
            "sequence_number": "3"
        }
    ],
    "millis_behind_latest": ""
  },
  "Tag": "latest",
  "StreamName": "dis-swtest"
}
```

Table 1-9 Parameter description

Parameter	Type	Example Value	Description
ShardID	String	shardId-0000000000	Partition ID
next_partition_cursor	String	See the example.	Next partition cursor
Records	Map	See the example.	Data records stored in a DIS stream
partition_key	String	See the example.	Partition key
data	String	See the example.	Data blocks, which are added by the data producer to the stream
sequence_number	Int	See the example.	Record ID, which is automatically allocated by DIS
Tag	String	latest	Stream tag
StreamName	String	dis-swtest	Stream name

- Timer example event

```
{
  "version": "v1.0",
  "time": "2018-06-01T08:30:00+08:00",
  "trigger_type": "TIMER",
  "trigger_name": "Timer_001",
  "user_event": "User Event"
}
```

Table 1-10 Parameter description

Parameter	Type	Example Value	Description
version	String	V1.0	Event version

Parameter	Type	Example Value	Description
time	String	2018-06-01T08:30:00+08:00	Time when an event occurs.
trigger_type	String	TIMER	Trigger type
trigger_name	String	Timer_001	Trigger name
user_event	String	User Event	Additional information of the trigger

- Kafka example event

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

Table 1-11 Parameter description

Parameter	Type	Example Value	Description
event_version	String	v1.0	Event version
event_time	String	2018-01-09T07:50:50.028Z	Time when an event occurs
trigger_type	String	KAFKA	Event type
region	String	ap-southeast-3	Region where a Kafka instance resides
instance_id	String	81335d56-b9fe-4679-ba95-7030949cc76b	Kafka instance ID
messages	String	See the example.	Message content
topic_id	String	topic-test	Message ID

- GeminiDB example event

```
{
   "records": [
      {
         "event_name": "\"insert\"",
         "event_version": "1.0",
         "event_source": "gemini_mongo",
         "region": "{region}",
         "gemini_mongo": {
            "full_document": "{\"_id\": {\"$oid\":\"5f61de944778db5fcded3f87\"},\"zhangsan\":
\"zhangsan\"}",
            "ns": "{\"db\": \"zhangsan\",\"coll\": \"zhangsan\"}",
            "size_bytes": "100",
            "token": "{\"_data\":
\"825F61DE940000000129295A1004A2D9AE61206C43A5AF47CAF7C5C00C5946645F696400645F61DE
944778DB5FCDED3F870004\"}"
         },
         "event_source_id": "51153d19-2b7d-402c-9a79-757163258a36"
      }
   ],
   "vernier": "{\"_data\":
\"825F61DE940000000129295A1004A2D9AE61206C43A5AF47CAF7C5C00C5946645F696400645F61DE
944778DB5FCDED3F870004\"}"
}
```

Table 1-12 Parameter description

Parameter	Type	Example Value	Description
region	String	ap-southeast-3	Region where a GeminiDB instance resides
event_source	String	gemini_mongo	Event source
event_version	String	1.0	Event version
full_document	String	See the example.	Complete file information
size_bytes	Int	100	Message bytes
token	String	See the example.	Base64-encoded data
vernier	String	See the example.	Cursor

- RabbitMQ example event

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



```
    }
  ]
}
```

Table 1-13 Parameter description

Parameter	Type	Example Value	Description
event_version	String	v1.0	Event version
Region	String	ap-southeast-3	Region where a RabbitMQ instance resides
instance_id	String	81335d56-b9fe-4679-ba95-7030949cc76b	RabbitMQ instance ID

1.3 Function Project Packaging Rules

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 1-14** describes the rules for packaging a function project.

Table 1-14 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 2.7	Not supported.	• If the function project files are saved under the **~/Code/** directory, select and package all files under this directory to ensure that the function handler is under the root directory after the ZIP file is decompressed. • If the function project uses third-party dependencies, package the dependencies into a ZIP file, and import the ZIP file on the function code page. Alternatively, package the third-party dependencies and the function project files together.	Compress project files into a ZIP file and upload it to an OBS bucket.

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

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

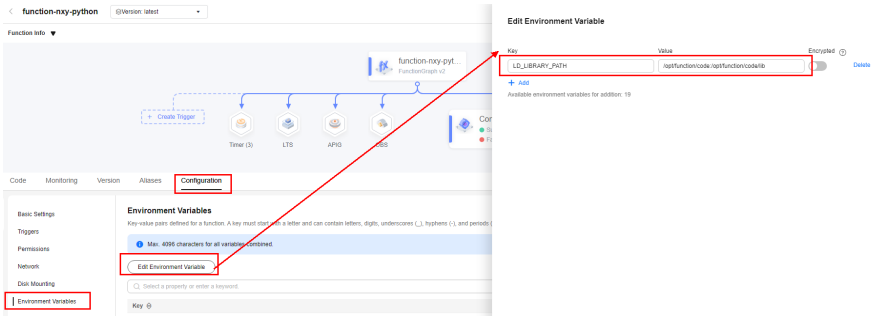
```
Example.zip                      Example project package
|--- bootstrap                   Executable boot file
```

1.4 Referencing DLLs in Functions

- By default, the root directory and the **lib** folder in this directory have been configured in the **LD_LIBRARY_PATH** environment variable. You only need to add dynamic link libraries (DLLs) here.
- You can directly modify the **LD_LIBRARY_PATH** variable in the code.
- If the dependent **.so** file is stored in another directory, you can specify it when setting the **LD_LIBRARY_PATH** environment variable. For details, see **Configuring Environment Variables**.

  In the following example, **/opt/function/code** and **/opt/function/code/lib** indicate the project directories of the function code.

Figure 1-1 Setting environment variables



- If a library in a mounted file system is used, specify its directory in the **LD_LIBRARY_PATH** variable on the **Configuration** tab page.

2 Initializer

Overview

An initializer is a logic entry for initializing functions. For a function with an initializer, FunctionGraph invokes the initializer to initialize the function and then invokes the handler to process function requests. For a function without an initializer, FunctionGraph only invokes the handler to process function requests.

Applicable Scenario

FunctionGraph executes a function in the following steps:

1. Allocate container resources to the function.
2. Download function code.
3. Use the runtime to load the function code.
4. Initialize the function.
5. Process the function request and return the result.

Steps **1**, **2**, and **3** are performed during a systematic cold start, ensuring a stable latency through proper resource scheduling and process optimization. Step **4** is performed during an application-layer cold start in complex scenarios, such as loading large models for deep learning, building database connection pools, and loading function dependencies.

To reduce the latency caused by an application-layer cold start, FunctionGraph provides the initializer to identify function initialization logic for proper resource scheduling.

Benefits of the Initializer

- Isolate function initialization and request processing to enable clearer program logic and better structured and higher-performance code.
- Ensure a 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.

- If there are continuous requests and the function is not updated, the system may still reclaim or update existing containers. Although no code starts on the platform side, there are cold starts on the service side. The initializer can be used to ensure that requests can be processed properly.

Features of the Initializer

The initializer of each runtime has the following features:

- No custom parameters

  The initializer does not support custom parameters and only uses the variables in **context** for logic processing.

- No return values

  No values will be returned for initializer invocation.

- Initialization timeout

  You can set an initialization timeout (≤ 300s) different from the timeout for invoking the handler.

- Execution duration

  Function instances are processes that execute function logic in a container and automatically scale if the number of requests changes. When a new function instance is generated, the system invokes the initializer and then executes the handler logic if the invocation is successful.

- One-time execution

  After each function instance starts, the initializer can only be executed once. If an instance fails to execute the initializer, the instance is abandoned and another instance starts to execute the initializer. A maximum of three attempts are allowed. If the initializer is executed successfully, the instance will only process requests upon invocation and will no longer execute the initializer again within its lifecycle.

- Naming rule

  For all runtimes except Java, the initializer can be named in the format of *[File name].[Initializer name]*, which is similar with the format of a handler name. For Java, a class needs to be defined to implement the predefined initializer.

- Billing

  The initializer execution duration will be billed at the same rate as the function execution duration.

3 Node.js

3.1 Developing an Event Function

Function Syntax

NOTE

You are advised to use Node.js 12.13.

- Node.js 6.10

  Use the following syntax when creating a handler function in Node.js 6.10:

  export.handler = function(event, context, callback)

  - **handler**: name of the function that FunctionGraph invokes to execute your code. The name must be consistent with that you define when creating a function.
  - **event**: event parameter defined for the function. The parameter is in JSON format.
  - **context**: runtime information provided for executing the function. For details, see **SDK APIs**.
  - **callback**: used to return the defined **err** and **message** information to the frontend. The general syntax is **callback(err, message)**. You can define the error or message content, for example, a character string.
  - Function handler: **index.handler**.

    The function handler is in the format of *[File name].[Function name]*. For example, if you set the handler to **index.handler** in your function, FunctionGraph will load the handler function defined in the **index.js** file.

- Node.js 8.10, Node.js 10.16, Node.js 12.13, Node.js 14.18, Node.js 16.17, and Node.js 18.15

  Node.js 8.10, Node.js 10.16, Node.js 12.13, Node.js 14.18, Node.js 16.17, and Node.js 18.15 are compatible with the APIs of Node.js 6.10, and supports an **async** handler.

  exports.handler = async (event, context, callback [optional]) => { return data;}

  Responses are output through **return**.

Node.js Initializer

FunctionGraph supports the following Node.js runtimes:

- Node.js6.10 (runtime = Node.js6)
- Node.js8.10 (runtime = Node.js8)
- Nodejs10.16(runtime = Node.js10)
- Nodejs12.13(runtime = Node.js12)
- Node.js16.17(runtime = Node.js16)
- Node.js18.15(runtime = Node.js18)

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **index.initializer**, FunctionGraph loads the initializer function defined in the **index.js** file.

To use Node.js to build initialization logic, define a Node.js function as the initializer. The following is a simple initializer:

```
exports.initializer = function(context, callback) {
    callback(null, '');
    };
```

- Function Name

  The function name **exports.initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **index.initializer**, FunctionGraph loads the initializer function defined in the **index.js** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

- callback

  The **callback** parameter is used to return the invocation result. The signature of this parameter is **function(err, data)**, which is the same as that of the common **callback** parameter used in Node.js. If the value of **err** is not null, the function will return **HandledInitializationError**. The value of **data** is invalid because no value will be returned for function initialization. You can set the **data** parameter to null by referring to the previous example.

SDK APIs

Table 3-1 describes the context methods provided by FunctionGraph.

Results returned by using the getToken(), getAccessKey(), and getSecretKey() methods contain sensitive information. Exercise caution when using these methods.

Table 3-1 Context methods

Method	Description
getRequestID()	Obtains a request ID.
getRemainingTimeInMilliSeconds ()	Obtains the remaining running time of a function.
getAccessKey()	Obtains the AK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.**
getSecretKey()	Obtains the SK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.**
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getFunctionName()	Obtains the name of a function.
getRunningTimeInSeconds ()	Obtains the timeout of a function.
getVersion()	Obtains the version of a function.
getMemorySize()	Obtains the allocated memory.
getCPUNumber()	CPU usage of a function.

Method	Description
getPackage()	Obtains a function group, that is, an app.
getToken()	Obtains the token (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getLogger()	Obtains the **logger** method provided by the context and returns a log output class. Logs are output in the format of *Time-Request ID-Content* by using the **info** method. For example, use the **info** method to output logs: logg = context.getLogger() logg.info("hello")
getAlias	Obtains function alias.

Developing a Node.js Function

The following procedure uses a local function as an example. You can also create a new one on the console.

Constraints:

- In this example, the function project files are saved under the **~/Code/** directory. Select and package all files under the directory to ensure that the **index.js** file, the handler of your FunctionGraph function, is under the root directory when the **fss_examples_nodejs.zip** file is decompressed.
- If the first parameter returned by **callback** is not **null**, the function execution fails and the HTTP error message defined in the second parameter is returned.
- When an APIG trigger is used, the response must be in the output format used in this example. The body only supports the following values: For details about the constraints, see **Base64 Decoding and Response Structure**.
  - **null**: The HTTP response body is empty.
  - **[]byte**: The content in this byte array is the body of an HTTP response.
  - **string**: The content in this string is the body of an HTTP response.

Step 1 Create a function project.

1. Write function code with a sync handler.

   Open the text editor, compile a function, and save the code file as **index.js**. The following is a sync handler:

```
exports.handler = function (event, context, callback) {
    const error = null;
    const output = {
        'statusCode': 200,
```

```
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }
    callback(error, output);
}
```

2. Write function code with an async handler and runtime 8.10 or later.

```
exports.handler = async (event, context) => {
    const output =
    {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }
    return output;
}
```

If your Node.js function contains an asynchronous task, use **Promise** to execute the task in the current invocation. You can directly return the declared **Promise** or await to execute it. The asynchronous task can be executed only before the function responds to requests.

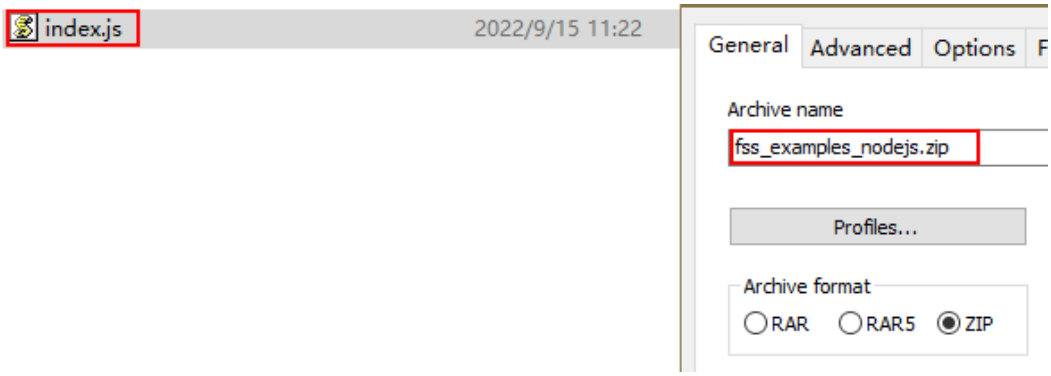
```
exports.handler =  async(event, context ) => {
    const output =
    {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }

    const promise = new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve(output)
        }, 2000)
    })
    return promise
    // anthor way
    // res = await promise;
    // return res
}
```

If you want the function to first respond and then execute the task, use an SDK or directly call the **asynchronous invocation** API. When using an APIG trigger, click the trigger name to go to the APIG console, and call the API in asynchronous mode.

Step 2 Package the project files.

This step uses an async handler as an example. After creating the function project, you get the following directory. Select all files under the directory and package them into the **fss_examples_nodejs.zip** file.

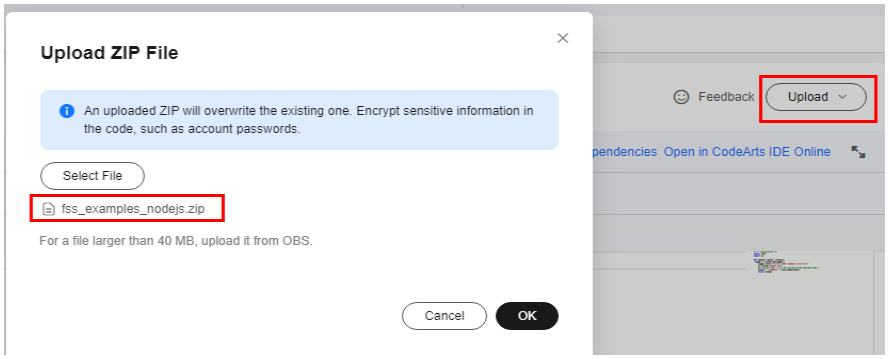
Figure 3-1 Packaging the project files



Step 3 Create a FunctionGraph function and upload the code package.

Log in to the FunctionGraph console, create a Node.js function, and upload the **fss_examples_nodejs.zip** file, as shown in **Figure 3-2**.

Figure 3-2 Uploading the code package

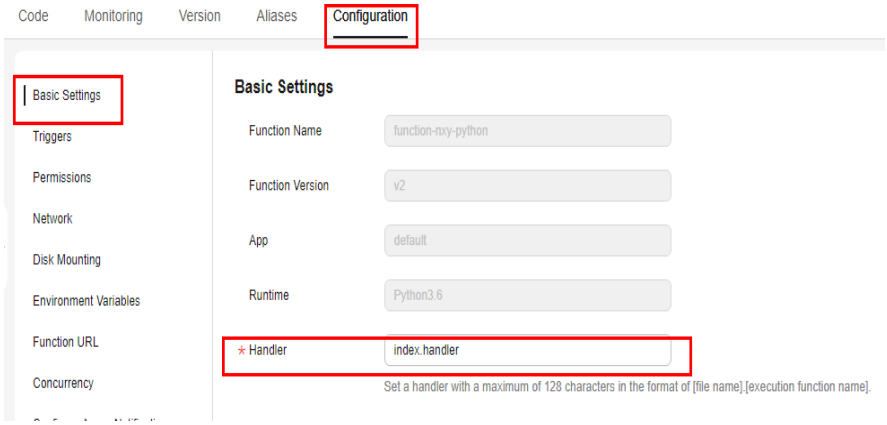


In the function configuration, the **index** of the **handler** must be consistent with the name of the function file created in **Step 1**, because the file name will help to locate the function file.

The **handler** is a function name, which must be the same as that in the **index.js** file created in **Step 1**.

In the navigation pane on the left of the FunctionGraph console, choose **Functions** > **Function List**. Click the name of the function to be set. On the function details page that is displayed, choose **Configuration** > **Basic Settings** and set the **Handler** parameter, as shown in **Figure 3-3**. The parameter value is in the format of **index.handler**. The values of **index** and **handler** can be customized.

Figure 3-3 Handler parameter

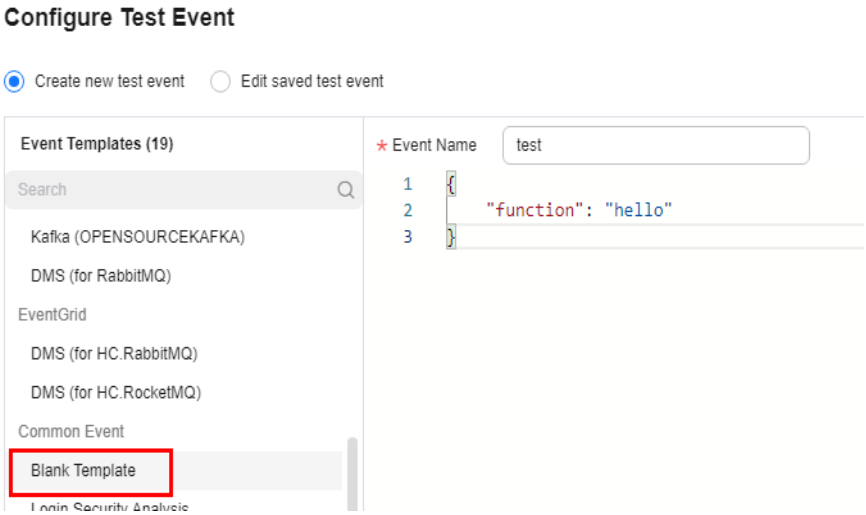


Step 4 Test the function.

1. Create a test event.

   On the function details page that is displayed, click **Configure Test Event**. Configure the test event information, as shown in **Figure 3-4**, and then click **Create**.

   Figure 3-4 Configuring a test event

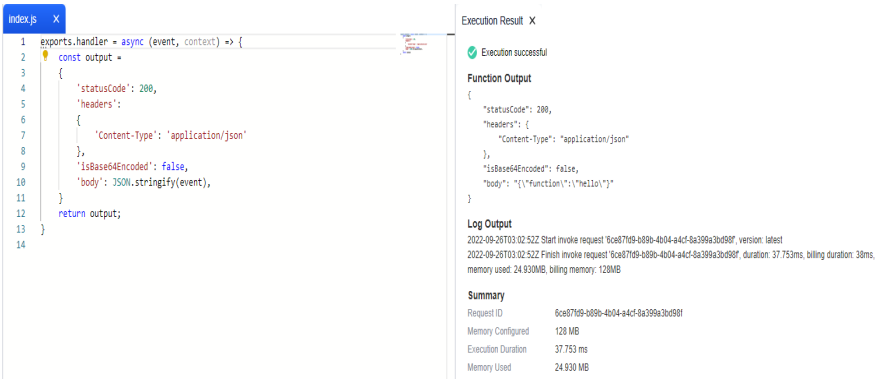
   

2. On the function details page, select the configured test event, and click **Test**.

Step 5 View the function execution result.

The function execution result consists of three parts: function output (returned by **callback**), summary, and logs (output by using the **console.log** or **getLogger()** method), as shown in **Figure 3-5**.

Figure 3-5 Test result



----End

Execution Result

The execution result consists of the function output, summary, and log output.

Table 3-2 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **errorType** is returned. The format is as follows:
		{
		"errorMessage": "",
		"errorType":"",
		}
		errorMessage: Error message returned by the runtime.
		errorType: Error type.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

3.2 Developing an HTTP Function

Deploy the Koa framework by using an HTTP function. For details, see **Creating an HTTP Function**.

Constraints

- HTTP functions can only use APIG or APIC triggers. According to the forwarding protocol between FunctionGraph and APIG/APIC, a valid HTTP function response must contain **body(String)**, **statusCode(int)**, **headers(Map)**, and **isBase64Encoded(boolean)**. By default, the response is encoded using Base64. The default value of **isBase64Encoded** is **true**. The same applies to other frameworks. For details about the constraints, see **Base64 Decoding and Response Structure**.
- By default, port 8000 is enabled for HTTP functions.
- **Table 3-1** describes the context methods provided by FunctionGraph.

Prerequisites

1. You have prepared a bootstrap file as the startup file of the HTTP function. Example:
   ```
   /opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node $RUNTIME_CODE_ROOT/index.js
   ```
   - **/opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node**: path of the Node.js compilation environment.
   - *$RUNTIME_CODE_ROOT*: system variable that represents the **/opt/function/code** path for storing project code in a container.
   - **index.js**: project file. You can also define a custom name.

   Table 3-3 lists the supported Node.js versions and the corresponding paths.

   Table 3-3 Node.js paths

Language	Path
Node.js 6	/opt/function/runtime/nodejs6.10/rtsp/nodejs/bin/node
Node.js 8	/opt/function/runtime/nodejs8.10/rtsp/nodejs/bin/node
Node.js 10	/opt/function/runtime/nodejs10.16/rtsp/nodejs/bin/node
Node.js 12	/opt/function/runtime/nodejs12.13/rtsp/nodejs/bin/node
Node.js 14	/opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node

2. Install the Node.js environment on a Linux host and prepare the Node.js project file.

   Koa web application

   a. Create a project folder.
      ```
      mkdir koa-example && cd koa-example
      ```
   b. Initialize the Node.js project and download the Koa framework. The **node_modules** folder and the **package.json** and **package-lock.json** files are created in the folder.
      ```
      npm init -y
      npm i koa
      ```
   c. Create the **index.js** file to reference the Koa framework. For details about how to use this framework, see **Koa's guide**.

      Sample code:
      ```
      const Koa = require("koa");
      const app = new Koa();
      ```



```
const main = (ctx) = >{
    if (ctx.request.path == ("/koa")) {
        ctx.response.type = " application/json";
        ctx.response.body = "Hello World, user!";
        ctx.response.status = 200;
    } else {
        ctx.response.type = " application/json";
        ctx.response.body = 'Hello World!';
        ctx.response.status = 200;
    }
};
app.use(main);
app.listen(8000, '127.0.0.1');
console.log('Node.js web server at port 8000 is running..')
```

d. Create a bootstrap file.

```
/opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node $RUNTIME_CODE_ROOT/index.js
```

3. Compress the project files and the bootstrap file into a ZIP package. The Koa framework is used as an example.

```
[root@                koa-example]# ls
bootstrap  index.js  koa.zip  node_modules  package.json  package-lock.json
```

3.3 Node.js Template

```
exports.handler = async (event, context) => {
    const output =
    {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }
    return output;
}
```

3.4 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

Constraints

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

Creating a Dependency for a Node.js Function

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

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

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


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

- Linux OS

  Run the following command to generate a ZIP package.

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

  The required dependency is generated.

- Windows OS

  Compress **node_modules** into a ZIP file.

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

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

NOTE

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

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

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

4 Python

4.1 Developing an Event Function

Function Syntax

NOTE

You are advised to use Python 3.6.

FunctionGraph supports Python 2.7, Python 3.6, Python 3.9, and Python 3.10.

Syntax for creating a handler function in Python:

def handler (event, context)

- **handler**: name of the function that FunctionGraph invokes to execute your code. The name must be consistent with that you define when creating a function.
- **event**: event parameter defined for the function. The parameter is in JSON format.
- **Context**: runtime information provided for executing the function. For details, see **SDK APIs**.

Python Initializer

FunctionGraph supports the following Python runtimes:

- Python 2.7 (runtime = python2.7)
- Python 3.6 (runtime = python3)
- Python 3.9 (runtime = python3)
- Python 3.10 (runtime = python3)

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.py** file.

To use Python to build initialization logic, define a Python function as the initializer. The following is a simple initializer (Python 2.7 is used as an example):

```
def my_initializer(context):
    print 'hello world!'
```

- Function Name

  The function name **my_initializer** must be the initializer function name specified for a function. For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.py** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

SDK APIs

Table 4-1 describes the context methods provided by FunctionGraph.

Results returned by using the **getToken()**, **getAccessKey()**, and **getSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

Table 4-1 Context methods

Method	Description
getRequestID()	Obtains a request ID.
getRemainingTimeInMilliSeconds ()	Obtains the remaining running time of a function.
getAccessKey()	Obtains the AK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.**
getSecretKey()	Obtains the SK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.**
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.

Method	Description
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getFunctionName()	Obtains the name of a function.
getRunningTimeInSeconds ()	Obtains the timeout of a function.
getVersion()	Obtains the version of a function.
getMemorySize()	Obtains the allocated memory.
getCPUNumber()	CPU usage of a function.
getPackage()	Obtains a function group, that is, an app.
getToken()	Obtains the token (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getLogger()	Obtains the **logger** method provided by the context and returns a log output class. Logs are output in the format of *Time-Request ID-Content* by using the **info** method. For example, use the **info** method to output logs: log = context.getLogger() log.info("test")
getAlias	Obtains function alias.

Developing a Python Function

Perform the following steps to develop a Python function:

Constraints:

- FunctionGraph can return only the following types of values:
  - **None**: The HTTP response body is empty.

  - **String**: The content in this string is the body of an HTTP response.
  - **Other**: For a value rather than **None** or **String**, FunctionGraph encodes the value in JSON, and uses the encoded object as the body of an HTTP response. The **Content-Type** header of the HTTP response is set to **application/json**.
- For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.
- In this example, the function project files are saved under the **~/code/** directory. Select and package all files under the directory to ensure that the **index.py** file, the handler of your FunctionGraph function, is under the **root** directory when the **fss_examples_python2.7.zip** file is 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 will be reported indicating a module loading failure.

NOTE

Python 2.7 is used an example.

Step 1 Create a function project.

1. Write code for printing text **helloworld**.

   Open the text editor, compile a HelloWorld function, and save the code file as **helloworld.py**. The code is as follows:

   ```
   def printhello():
       print 'hello world!'
   ```

2. Define a FunctionGraph function.

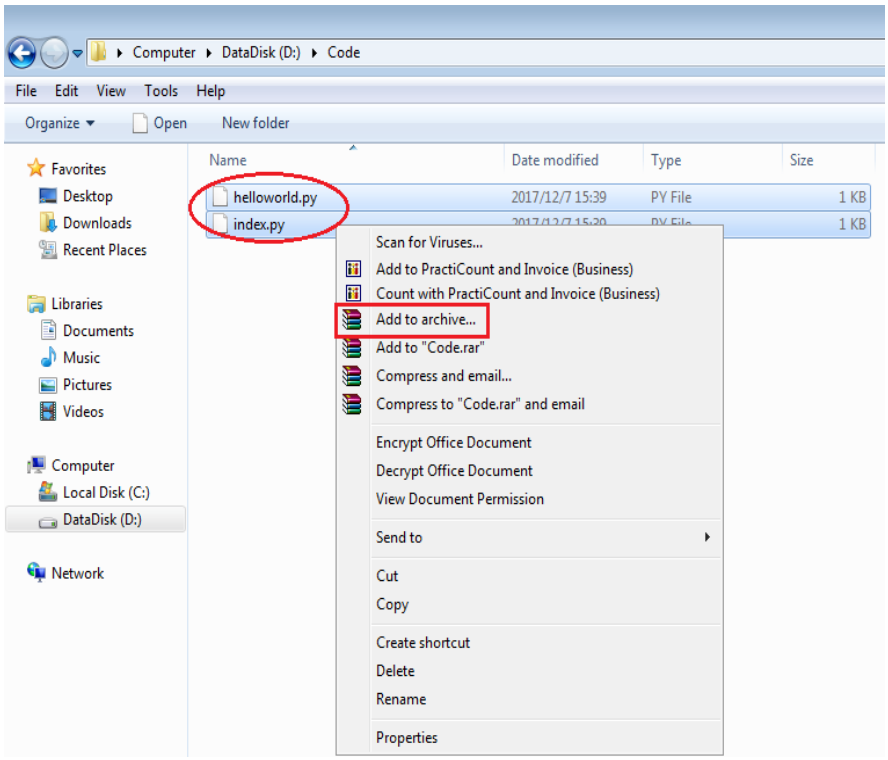
   Open the text editor, define a function, and save the function file as **index.py** under the same directory as the **helloworld.py** file. The function code is as follows:

   ```
   import json
   import helloworld

   def handler (event, context):
       output =json.dumps(event)
       helloworld.printhello()
       return output
   ```

Step 2 Package the project files.

After creating the function project, you get the following directory. Select all files under the directory and package them into the **fss_examples_python2.7.zip** file, as shown in **Figure 4-1**.

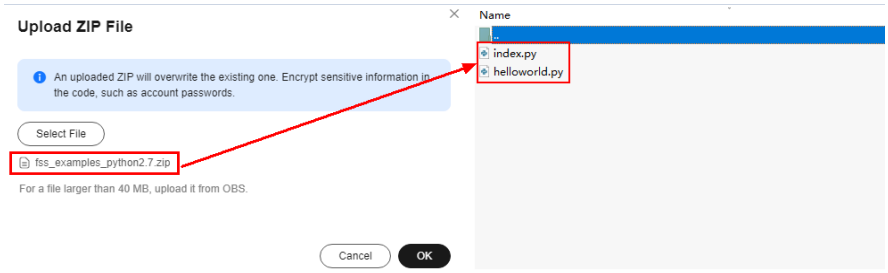
Figure 4-1 Packaging the project files



Step 3 Create a FunctionGraph function and upload the code package.

Log in to the FunctionGraph console, create a Python function, and upload the **fss_examples_python2.7.zip** file, as shown in **Figure 4-2**.

Figure 4-2 Uploading the code package



The **index** of the **handler** must be consistent with the name of the file created in **Step 1 (2)**, because the file name will help to locate the function file.

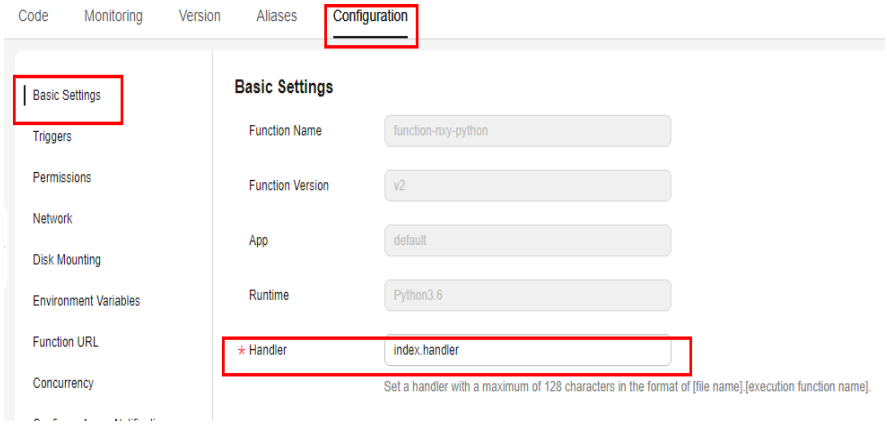
The **handler** is the function name, which must be the same as that in the **index.py** file created in **Step 1 (2)**.

After you upload the **fss_examples_python2.7.zip** file to OBS, when the function is triggered, FunctionGraph decompresses the file to locate the function file through **index** and locate the function defined in the **index.py** file through **handler**, and then executes the function.

In the navigation pane on the left of the FunctionGraph console, choose **Functions** > **Function List**. Click the name of the function to be set. On the function details page that is displayed, choose **Configuration** > **Basic Settings**

and set the **Handler** parameter, as shown in Figure 4-3. The parameter value is in the format of **index.handler**. The values of **index** and **handler** can be customized.

Figure 4-3 Handler parameter

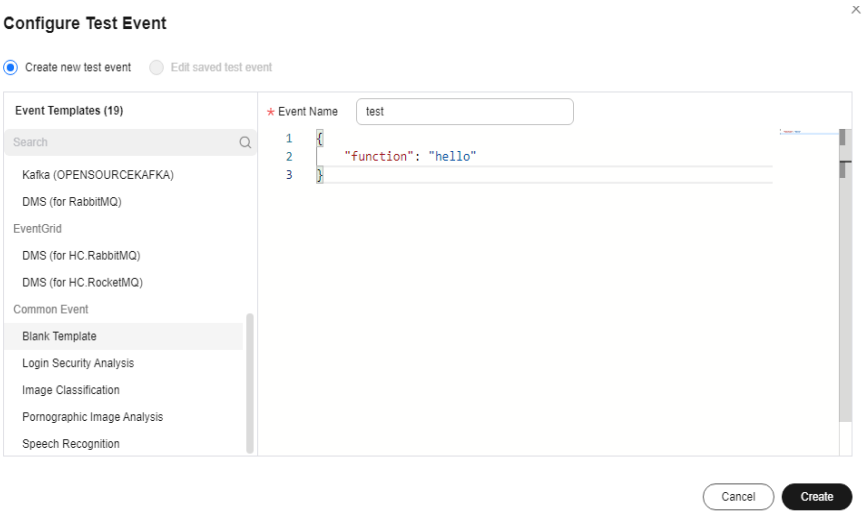


Step 4 Test the function.

1. Create a test event.

   On the function details page that is displayed, click **Configure Test Event**. Configure the test event information, as shown in Figure 4-4, and then click **Create**.

   Figure 4-4 Configuring a test event



2. On the function details page, select the configured test event, and click **Test**.

Step 5 View the function execution result.

The function execution result consists of three parts: function output (returned by **callback**), summary, and logs (output by using the **print()** method), as shown in Figure 4-5.

Figure 4-5 Test result

----End

Execution Result

The execution result consists of the function output, summary, and log output.

Table 4-2 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage**, **errorType**, and **stackTrace** is returned. The format is as follows:
		`{`
		`"errorMessage": "",`
		`"errorType": "",`
		`"stackTrace": []`
		`}`
		errorMessage: Error message returned by the runtime.
		errorType: Error type.
		stackTrace: Stack error information returned by the runtime.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

4.2 Python Template

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



```
def handler (event, context):
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
```

4.3 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

Constraints

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

Creating a Dependency for a Python Function

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

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

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

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

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

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

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

The required dependency is generated.

NOTE

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

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


5 Java

5.1 Developing an Event Function

5.1.1 Developing Functions in Java (Using Eclipse)

Function Syntax

The following is the syntax for creating a handler function in Java:

Scope Return parameter Function name (User-defined parameter, Context)

- *Scope*: It must be defined as **public** for the function that FunctionGraph invokes to execute your code.
- *Return parameter*: user-defined output, which is converted into a character string and returned as an HTTP response. The HTTP response is a JSON string.
- *Function name*: user-defined function name.
- *User-defined parameter*: FunctionGraph supports only one user-defined parameter. For complex parameters, define them as an object and provide data through JSON strings. When invoking a function, FunctionGraph parses the JSON strings as an object.
- *Context*: runtime information provided for executing the function. For details, see **SDK APIs**.

When creating a function in Java, define a handler in the format of *[Package name].[Class name].[Function name]*.

Constraints

Results returned by using the **getToken()**, **getAccessKey()**, and **getSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

Java Initializer

FunctionGraph supports the following Java runtime:

- Java 8
- Java 11

Initializer syntax:

[Package name].[Class name].[Execution function name]

For example, if the initializer is named **com.huawei.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com.huawei** file.

To use Java to build initialization logic, define a Java function as the initializer. The following is a simple initializer:

```
public void my_initializer(Context context)
{
RuntimeLogger log = context.getLogger();
log.log(String.format("ak:%s", context.getAccessKey()));
}
```

- Function Name

  The function name **my_initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **com.huawei.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com.huawei** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

SDK APIs

The **Java SDK** (verification file: **fss-java-sdk-2.0.5.sha256**) provides context, event, and logging APIs.

- Event APIs

  Event structure definitions are added to the Java SDK. Currently, DMS, DIS, SMN, timer, APIG, and Kafka triggers are supported. The definitions make coding much simpler when triggers are required.

  a. **APIG trigger**

     i. APIGTriggerEvent methods

Table 5-1 APIGTriggerEvent methods

Method	Description
isBase64Encoded()	Checks whether the body of an event is encoded using Base64.
getHttpMethod()	Obtains the HTTP request method.
getPath()	Obtains the HTTP request path.
getBody()	Obtains the HTTP request body.

Method	Description
getPathParameters()	Obtains all path parameters.
getRequestContext()	Obtains APIG configuration information. (The **APIGRequestContext object** is returned.)
getHeaders()	Obtains the HTTP request header.
getQueryStringParameters()	Obtains query parameters. **The value of a query parameter cannot be an array. To support an array, customize the corresponding event structure.**
getRawBody()	Obtains the content before Base64 encoding.
getUserData()	Obtains the user data set in the APIG custom authorizer.

Table 5-2 APIGRequestContext methods

Method	Description
getApiId()	Obtains the API ID.
getRequestId()	Obtains the request ID of an API request.
getStage()	Obtains the name of the environment in which an API has been published.
getSourceIp()	Obtains the source IP address in the APIG custom authorizer.

ii. APIGTriggerResponse methods

Table 5-3 APIGTriggerResponse construction methods

Method	Description
APIGTriggerResponse()	Set the following parameters: • **headers**: null • **statusCode**: 200 • **body**: "" • **isBase64Encoded**: false
APIGTriggerResponse(statusCode, headers, body)	Set the value of **isBase64Encoded** to **false**, and use the input values of other parameters.
APIGTriggerResponse(statusCode, headers, isBase64Encoded, body)	Set the parameters in sequence.

Table 5-4 APIGTriggerResponse methods

Method	Description
setBody(String body)	Sets the message body.
setHeaders(Map<String,String> headers)	Sets the HTTP response header to be returned.
setStatusCode(int statusCode)	Sets the HTTP status code.
setBase64Encoded(boolean isBase64Encoded)	Configures Base64 encoding for the response body.
setBase64EncodedBody(String body)	Encodes the input with Base64 and configures it in the body.
addHeader(String key, String value)	Adds a group of HTTP headers.
removeHeader(String key)	Removes the specified header.
addHeaders(Map<String,String> headers)	Adds multiple headers.

NOTE

APIGTriggerResponse methods have the **headers** attribute, which can be initialized using the setHeaders method or a constructor function with the **headers** parameter.

b. **DIS trigger**

Table 5-5 DISTriggerEvent methods

Method	Description
getShardID()	Obtains the partition ID.
getMessage()	Obtains the DIS message body. (**DISMessage structure**)
getTag()	Obtains the version of a function.
getStreamName()	Obtains the stream name.

Table 5-6 DISMessage methods

Method	Description
getNextPatitionCursor()	Obtains the next partition cursor.
getRecords()	Obtains message records. (**DISRecord structure**)
getMillisBehindLatest()	Reserved (Currently, **0** is returned.)

Table 5-7 DISRecord methods

Method	Description
getPartitionKey()	Obtains the data partition.
getData()	Obtain data.
getRawData()	Obtains UTF-8 data strings decoded using Base64.
getSequenceNumber()	Obtains the sequence number (ID of each record).

c. **DMS trigger**

Table 5-8 DMSTriggerEvent methods

Method	Description
getQueueId()	Obtains the queue ID.
getRegion()	Obtains the region name.
getEventType()	Obtains the event type. ("MessageCreated" is returned.)
getConsumerGroupId()	Obtains the consumer group ID.

Method	Description
getMessages()	Obtains DMS messages. (**DMSMessage structure**)

Table 5-9 DMSMessage methods

Method	Description
getBody()	Obtains the DMS message body.
getAttributes()	Obtains the message attribute set.

d. **SMN trigger**

Table 5-10 SMNTriggerEvent method

Method	Description
getRecord()	Obtains the message records. (**SMNRecord structure**)

Table 5-11 SMNRecord methods

Method	Description
getEventVersion()	Obtains event version. (The current version is 1.0.)
getEventSubscriptionUrn()	Obtains the subscription Uniform Resource Name (URN).
getEventSource()	Obtains the event source.
getSmn()	Obtains the message body (**SMNBody structure**).

Table 5-12 SMNBody methods

Method	Description
getTopicUrn()	Obtains the topic URN.
getTimeStamp()	Obtains the timestamp of a message.
getMessageAtrributes()	Obtains the message attribute set.
getMessage()	Obtains the message body.

Method	Description
getType()	Obtains the message type.
getMessageId()	Obtains the message ID.
getSubject()	Obtains the message topic.

e. **Timer trigger**

Table 5-13 TimerTriggerEvent methods

Method	Description
getVersion()	Obtains the version name. (The current version is 1.0.)
getTime()	Obtains the current time.
getTriggerType()	Obtains trigger type. (Timer)
getTriggerName()	Obtains the trigger name.
getUserEvent()	Obtains the additional information of the trigger.

f. **Kafka trigger**

Table 5-14 Kafka trigger

Method	Description
getEventVersion	Obtains event version.
getRegion	Obtains region information.
getEventTime	Obtains event time.
getTriggerType	Obtains trigger type.
getInstanceId	Obtains the instance ID.
getRecords	Obtains record.

NOTE

1. When using an APIG trigger, set the first parameter of the handler function (for example, **handler**) to **handler(APIGTriggerEvent event, Context context)**.
2. The preceding TriggerEvent methods have corresponding **set** methods, which are recommended for local debugging. DIS and LTS triggers have getRawData() methods, but do not have setRawData() methods.

- Context APIs

The context APIs are used to obtain the context, such as agency AK/SK, current request ID, allocated memory space, and number of CPUs, required for executing a function.

Table 5-15 describes the context APIs provided by FunctionGraph.

Table 5-15 Context methods

Method	Description
getRequestID( )	Obtains a request ID.
getRemainingTimeInMilliSeconds ( )	Obtains the remaining running time of a function.
getAccessKey( )	Obtains the AK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.**
getSecretKey( )	Obtains the SK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.**
getSecurityAccessKey( )	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey( )	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken( )	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.

Method	Description
getFunctionName( )	Obtains the name of a function.
getRunningTimeInSeconds ( )	Obtains the timeout of a function.
getVersion( )	Obtains the version of a function.
getMemorySize( )	Obtains the allocated memory.
getCPUNumber( )	CPU usage of a function.
getPackage( )	Obtains a function group, that is, an app.
getToken( )	Obtains the token (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getLogger( )	Obtains the **logger** method provided by the context. By default, information such as the time and request ID is output.
getAlias	Obtains function alias.

- Logging API

  Table 5-16 describes the logging API provided in the Java SDK.

  Table 5-16 Logging API

Method	Description
RuntimeLogger()	Records user input logs using the method **log(String string)**.

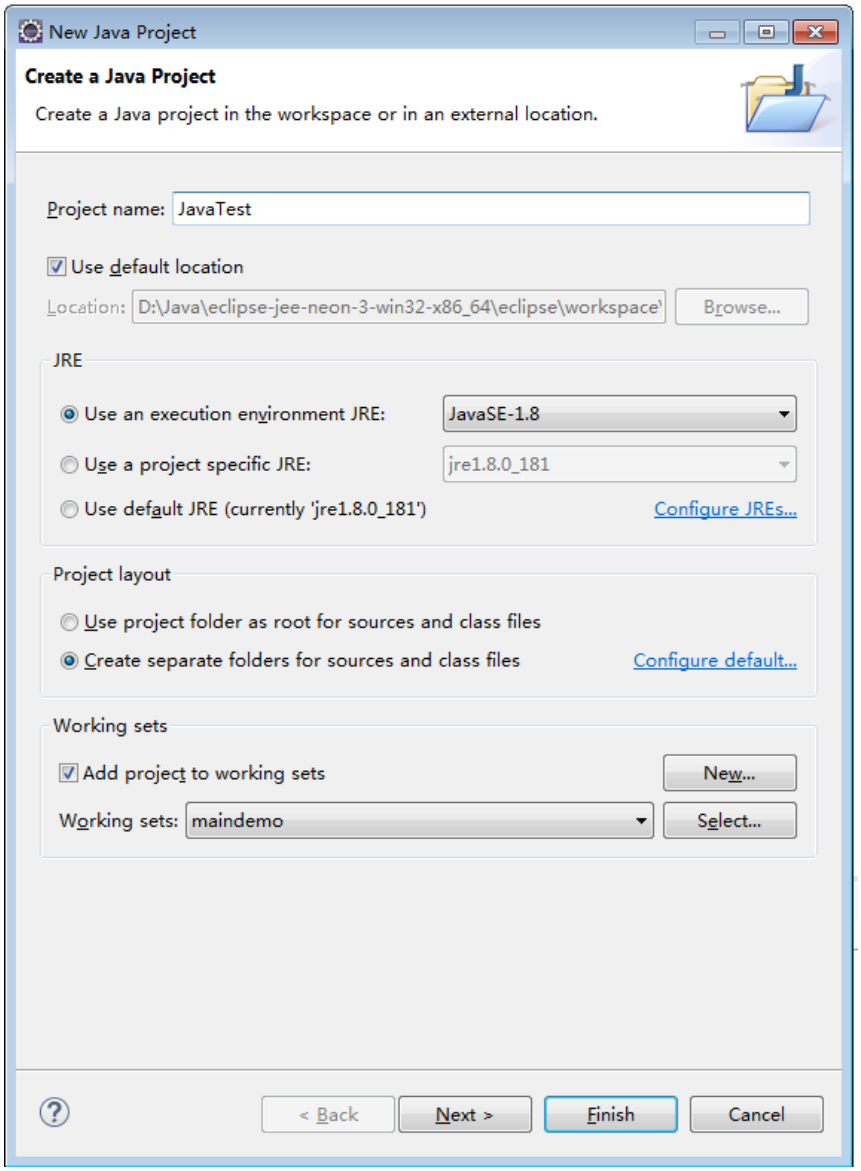
Developing a Java Function

Perform the following procedure to develop a Java function:

Step 1 Create a function project.

1. Configure Eclipse and create a Java project named JavaTest, as shown in **Figure 5-1**.

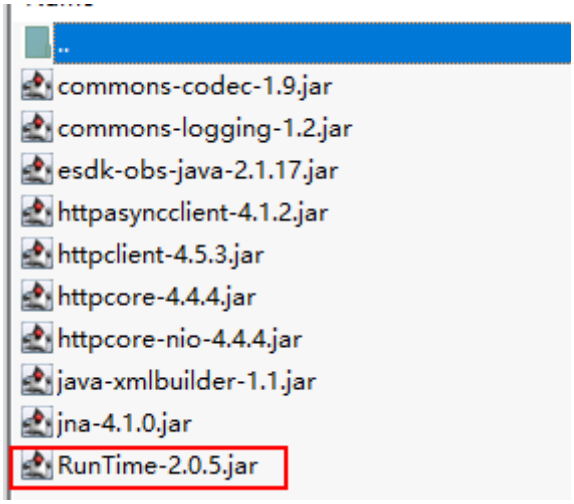
Figure 5-1 Creating a project



2. Add a dependency to the project.

   Download the **Java SDK** to a local development environment, and decompress the SDK package, as shown in **Figure 5-2**.

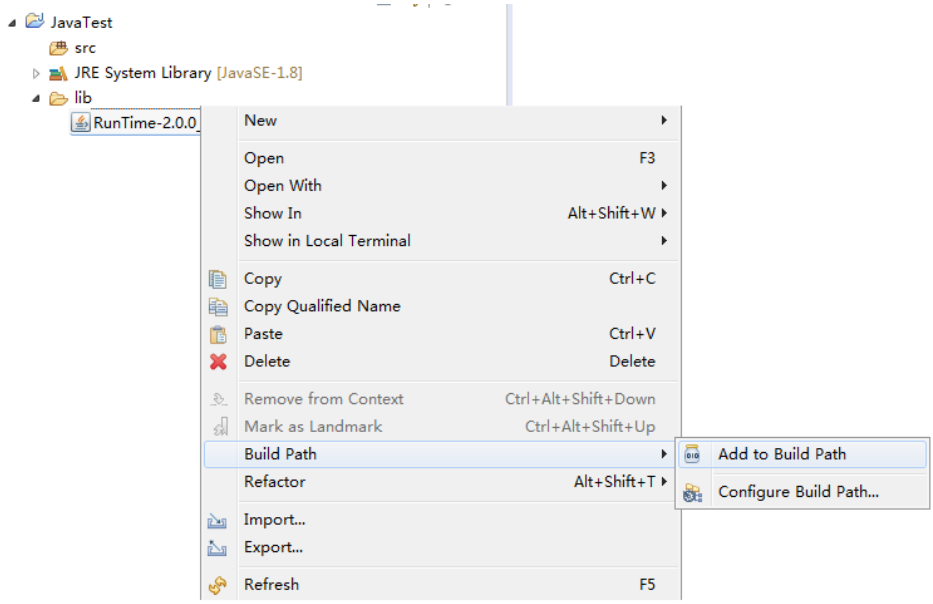
Figure 5-2 Decompressing the downloaded SDK



3. Configure the dependency.

   Create a folder named **lib** in the project directory, copy the **Runtime-2.0.5.jar** file in the SDK package to the **lib** folder, and add the JAR file as a dependency of the project, as shown in **Figure 5-3**.

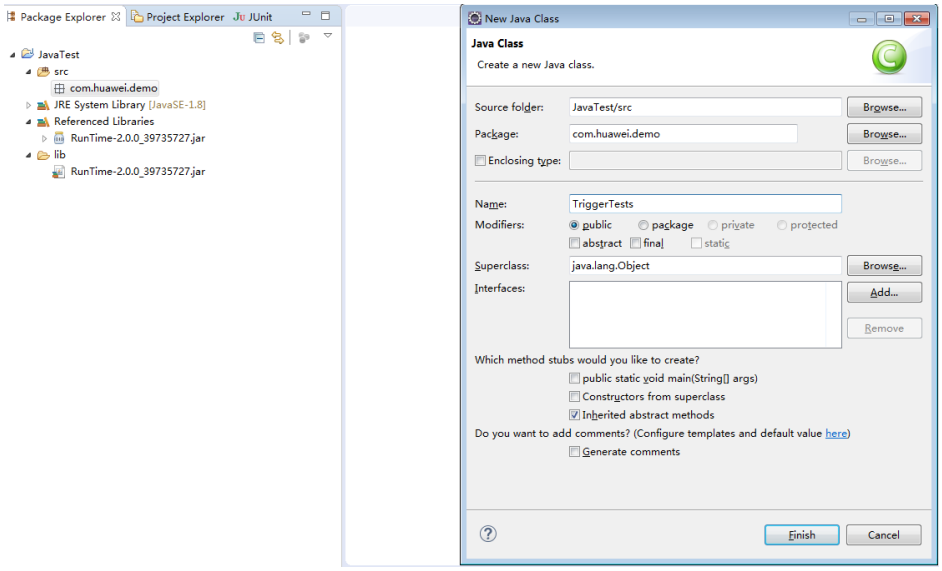
Figure 5-3 Configuring the dependency



Step 2 Create a function.

1. Create a package named **com.huawei.demo**, and then create a class named **TriggerTests** under the package, as shown in **Figure 5-4**.

Figure 5-4 Creating a package named com.huawei.demo



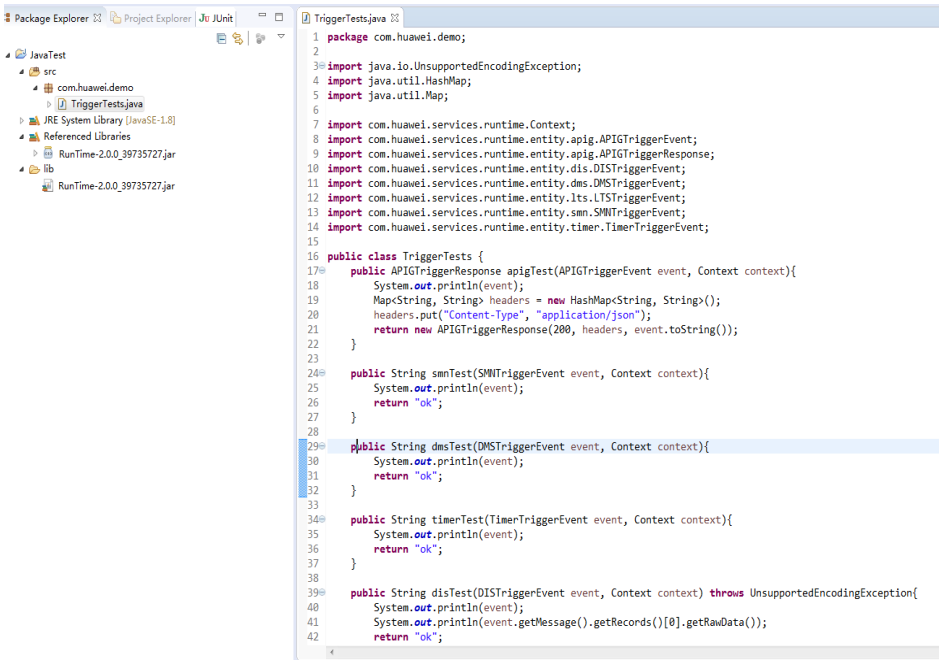
2. Define the function handler in **TriggerTests.java**, as shown in **Figure 5-5**.

```
package com.huawei.demo;

import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.Map;

import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;

public class TriggerTests {
    public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context){
        System.out.println(event);
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "application/json");
        return new APIGTriggerResponse(200, headers, event.toString());
    }

    public String smnTest(SMNTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String dmsTest(DMSTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String timerTest(TimerTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String disTest(DISTriggerEvent event, Context context) throws
UnsupportedEncodingException{
        System.out.println(event);
        System.out.println(event.getMessage().getRecords()[0].getRawData());
        return "ok";
    }
```



```
}
```

Figure 5-5 Defining the function handler

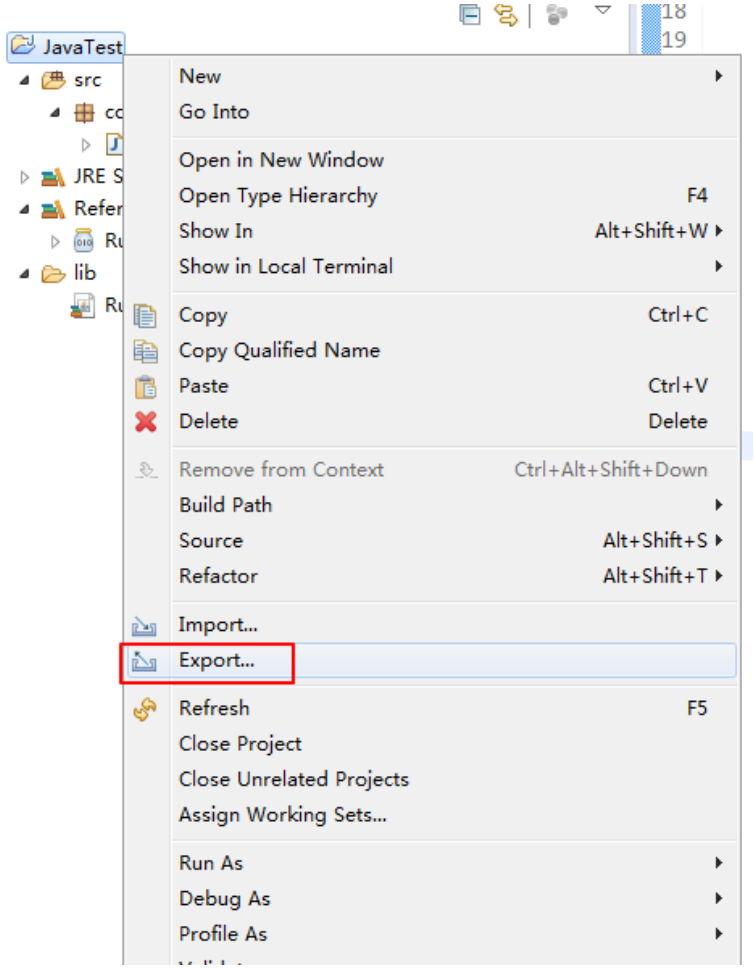


The example code contains multiple handler functions, which use different types of triggers. For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 3 Package the project files.

1. Right-click the **JavaTest** project and choose **Export**, as shown in **Figure 5-6**.

Figure 5-6 Packaging the project files



2. Export the project to a directory as a JAR file, as shown in **Figure 5-7** and **Figure 5-8**.

Figure 5-7 Selecting a format

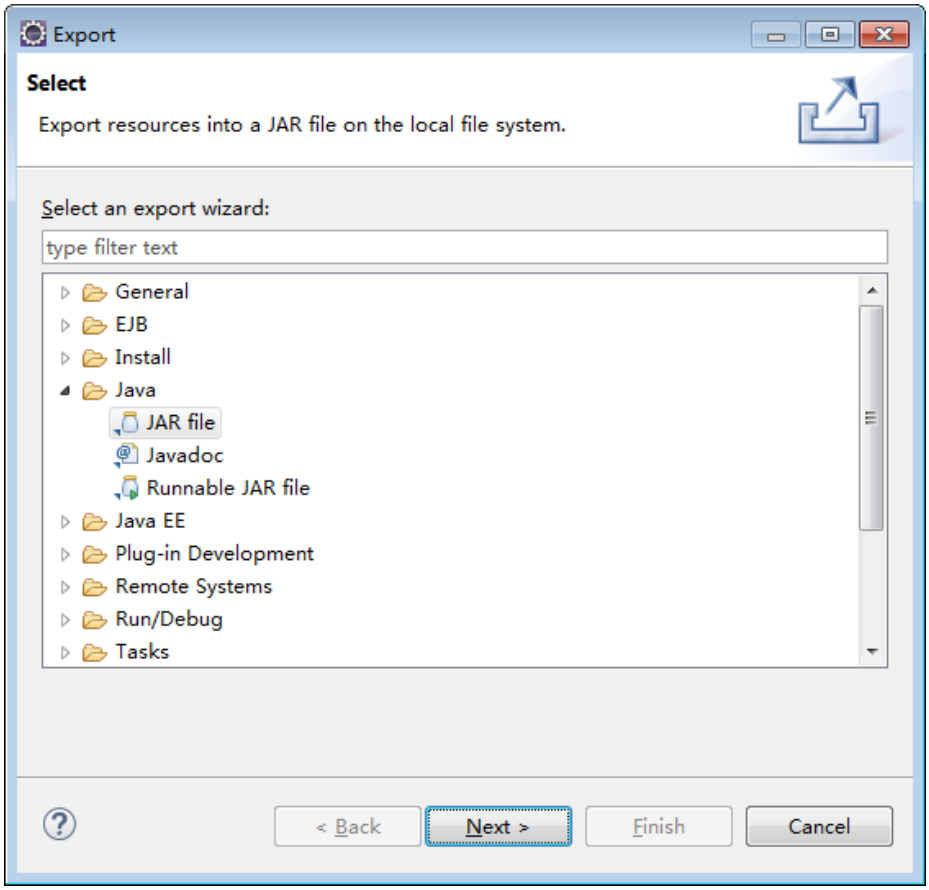
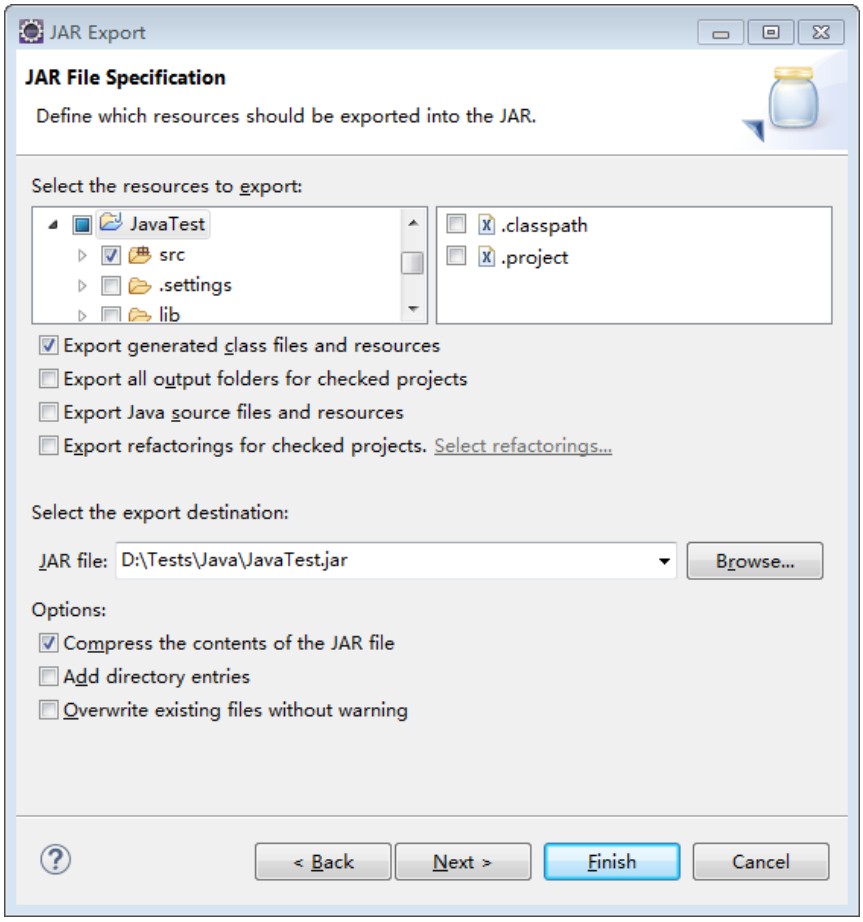


Figure 5-8 Specifying the destination path



Step 4 Log in to the FunctionGraph console, create a Java function, and upload the code package.

Step 5 Test the function.

1. Create a test event.

   Select **apig-event-template** and click **Create**.

2. Click **Test**.

   The function execution result consists of three parts: function output (returned by **callback**), summary, and logs (output by using the **console.log** or **getLogger()** method).

3. Create an APIG trigger.

4. Access the API URL, as shown in **Figure 5-9**.

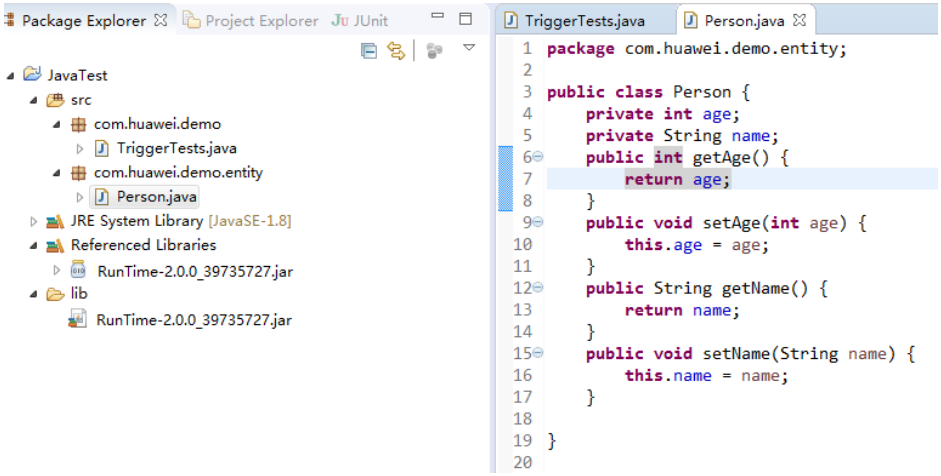
Figure 5-9 Accessing the API URL

Change the handler to **com.huawei.demo.TriggerTests.smnTest** and change the event template to **smn-event-template**.

Step 6 Use a user-defined parameter in the Java code.

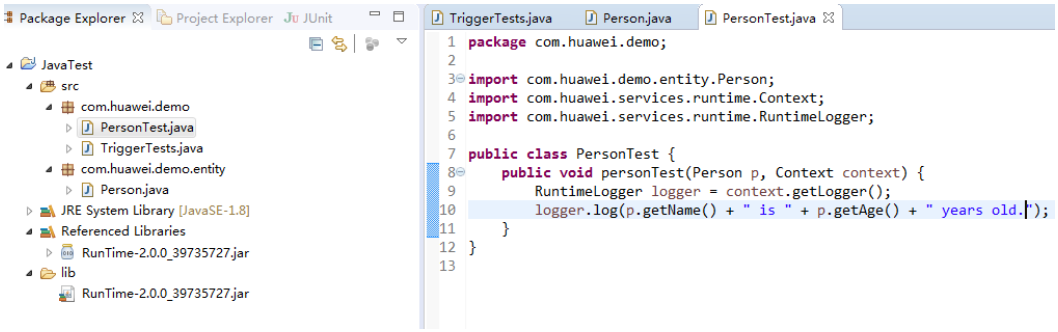
Create a Person class in the project, as shown in **Figure 5-10**.

Figure 5-10 Creating a Person class



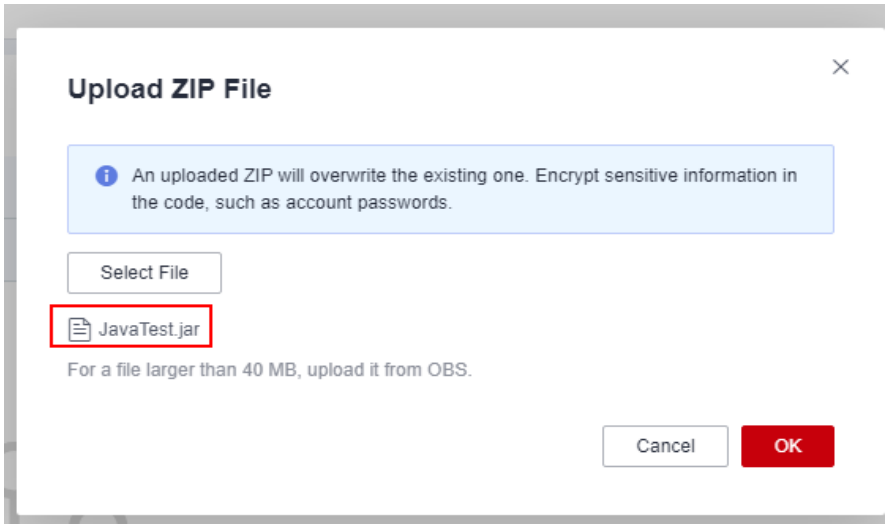
Create a class named **PersonTest.java**, and add a handler function to the class, as shown in **Figure 5-11**.

Figure 5-11 Creating a class named PersonTest.java



Upload the exported JAR file on the function details page, change the handler to **com.huawei.demo.PersonTest.personTest**, and save the change.

Figure 5-12 Uploading a JAR file



In the **Configure Test Event** dialog box, select **blank-template**, and enter a test event.

Click **Create**, and then click **Test**.

----End

Execution Result

The execution result consists of the function output, summary, and log output.

Table 5-17 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **stackTrace** is returned. The format is as follows:
		{
		"errorMessage": "",
		"stackTrace": []
		}
		errorMessage: Error message returned by the runtime.
		stackTrace: Stack error information returned by the runtime.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.

Parameter	Successful Execution	Failed Execution
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

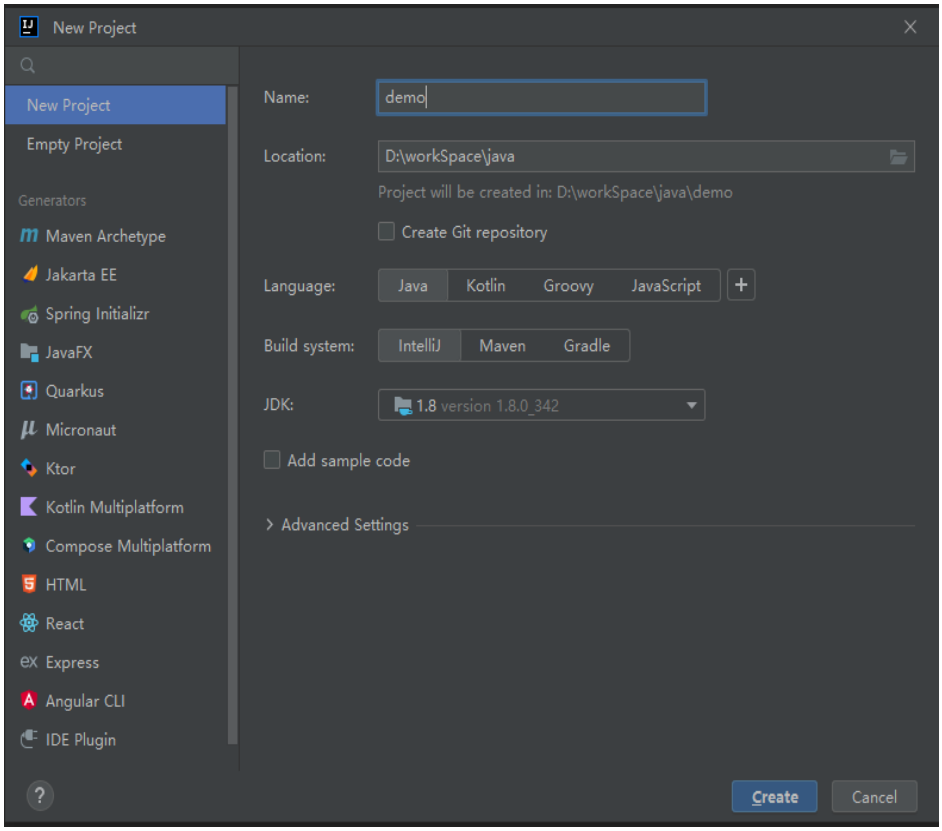
5.1.2 Developing Functions in Java (Using an IDEA Java Project)

Perform the following procedure to develop a Java function:

Step 1 Create a function project.

1. Configure the IDEA and create a Java project named **JavaTest**, as shown in Figure 5-13.

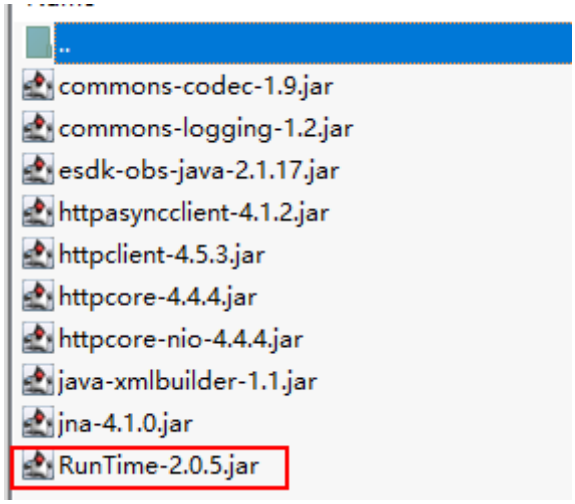
   Figure 5-13 Creating a project

2. Add dependencies to the project.

   Download the **Java SDK** to a local development environment, and decompress the SDK package, as shown in Figure 5-14.

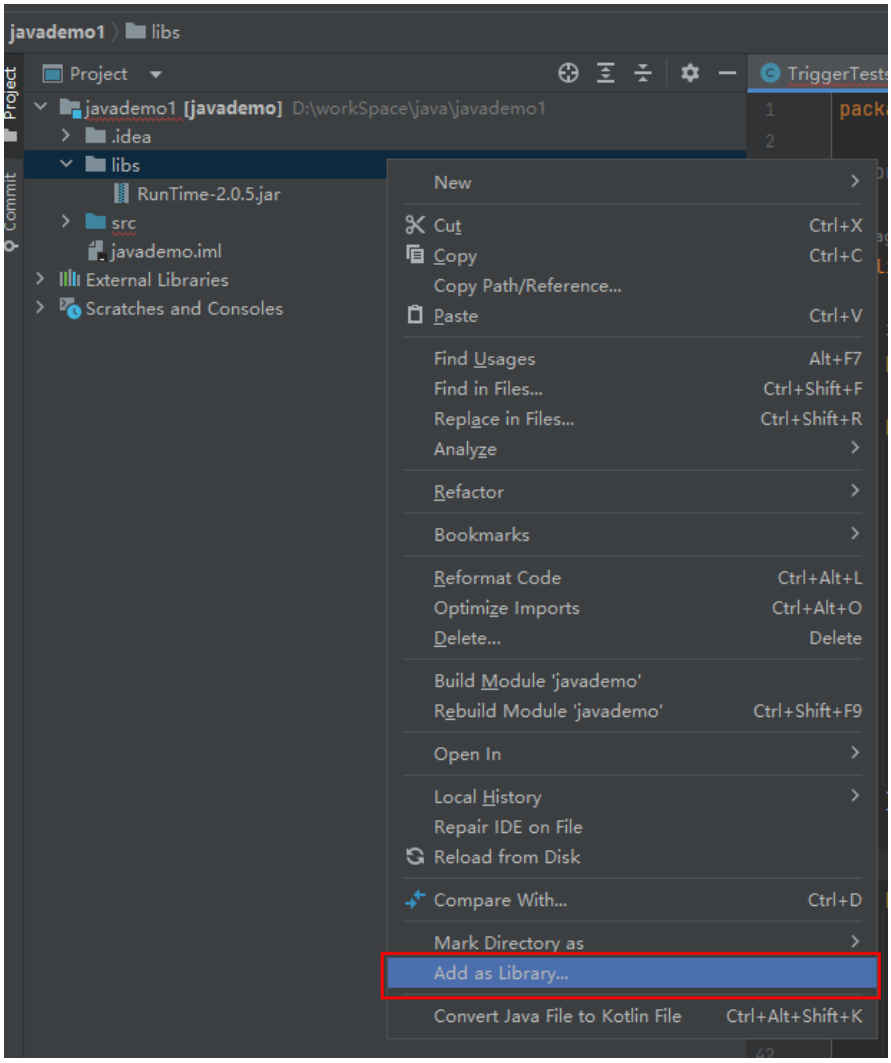
Figure 5-14 Decompressing the downloaded SDK



3. Configure dependencies.

   Create a folder named **lib** in the project directory, copy the **Runtime2.0.5.jar** file and other required dependencies to the **lib** folder, and add the JAR files as the dependencies of the project, as shown in **Figure 5-15**.

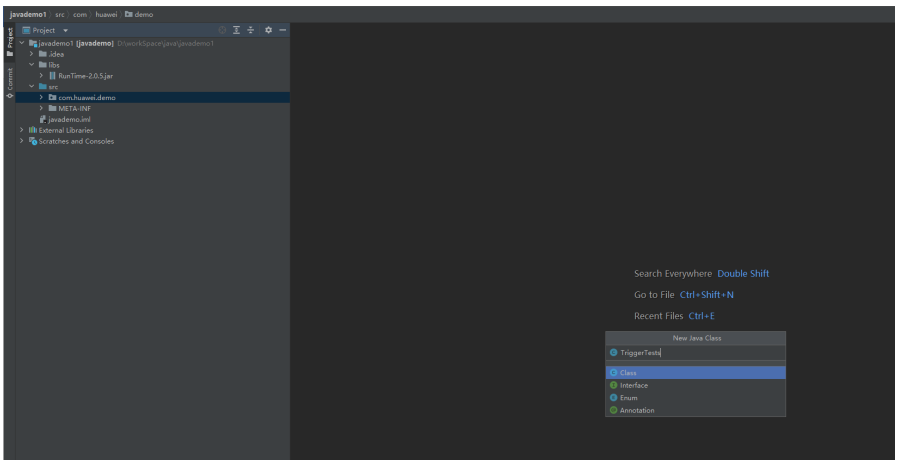
Figure 5-15 Configuring dependencies



Step 2 Create a function.

1. Create a package named **com.huawei.demo**, and then create a class named **TriggerTests** under the package, as shown in **Figure 5-16**.

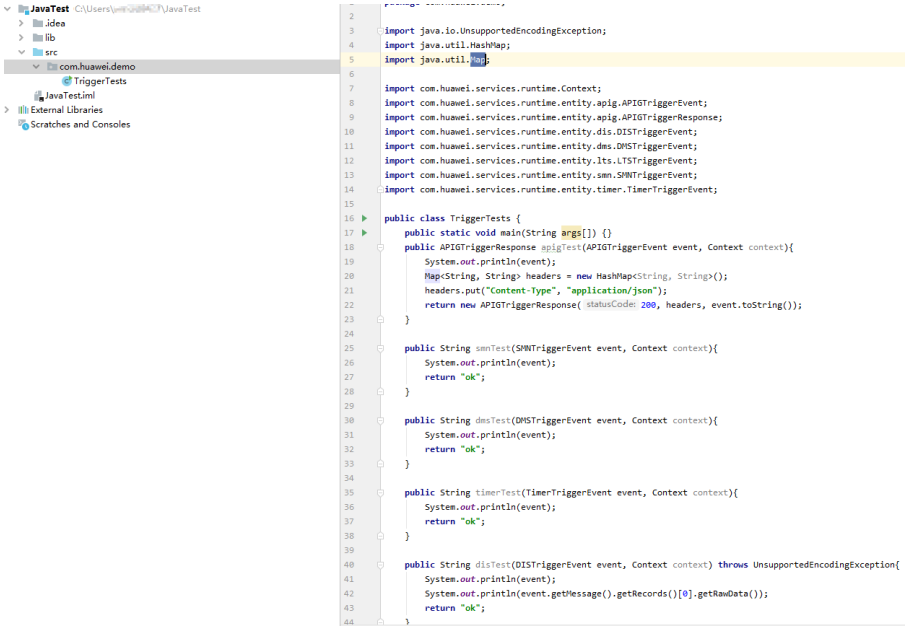
Figure 5-16 Creating the TriggerTests class



2. Define the function handler in **TriggerTests.java**, as shown in **Figure 5-17**. **A common Java project needs to be compiled using artifacts, and a main function needs to be defined.**

```
package com.huawei.demo;

import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.Map;

import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.lts.LTSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;

public class TriggerTests {
    public static void main(String args[]) {}
    public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context){
        System.out.println(event);
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "application/json");
        return new APIGTriggerResponse(200, headers, event.toString());
    }

    public String smnTest(SMNTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String dmsTest(DMSTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String timerTest(TimerTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String disTest(DISTriggerEvent event, Context context) throws
UnsupportedEncodingException{
        System.out.println(event);
        System.out.println(event.getMessage().getRecords()[0].getRawData());
        return "ok";
    }

    public String ltsTest(LTSTriggerEvent event, Context context) throws
UnsupportedEncodingException {
        System.out.println(event);
        event.getLts().getData();
        System.out.println("raw data: " + event.getLts().getRawData());
        return "ok";
    }
}
```


Figure 5-17 Defining the function handler

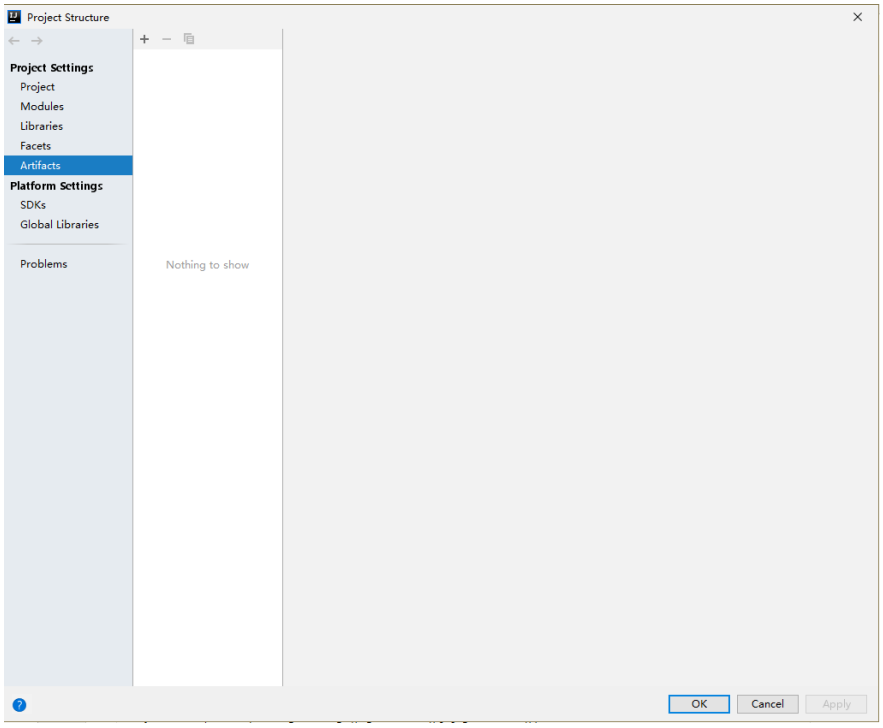


The example code contains multiple handler functions, which use different types of triggers. **For details about the constraints for the APIG event source, see Base64 Decoding and Response Structure.**

Step 3 Pack the project file.

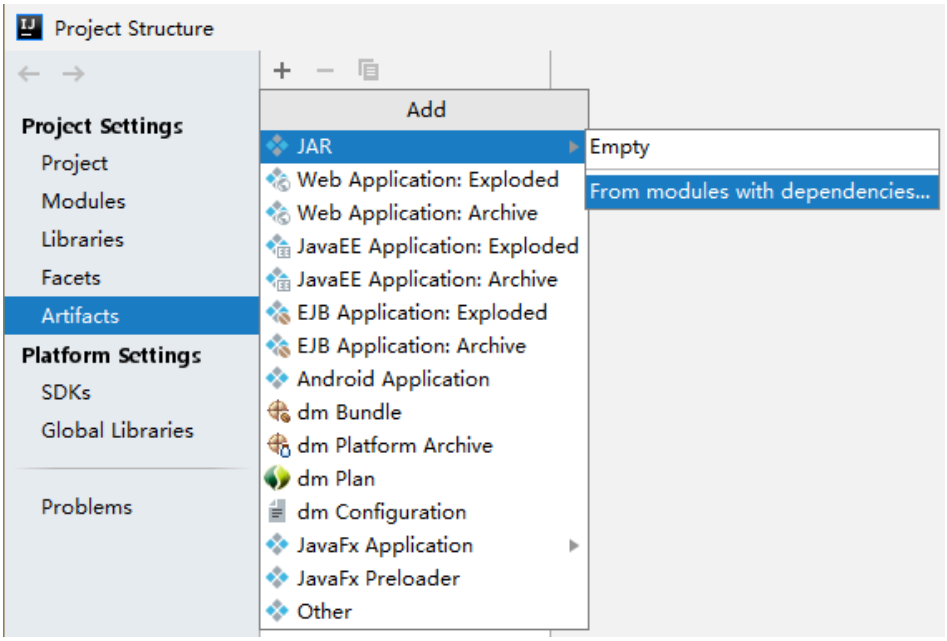
1. Choose **File** > **Project Structure**. The **Project Structure** page is displayed, as shown in **Figure 5-18**.

Figure 5-18 Going to the Project Structure page



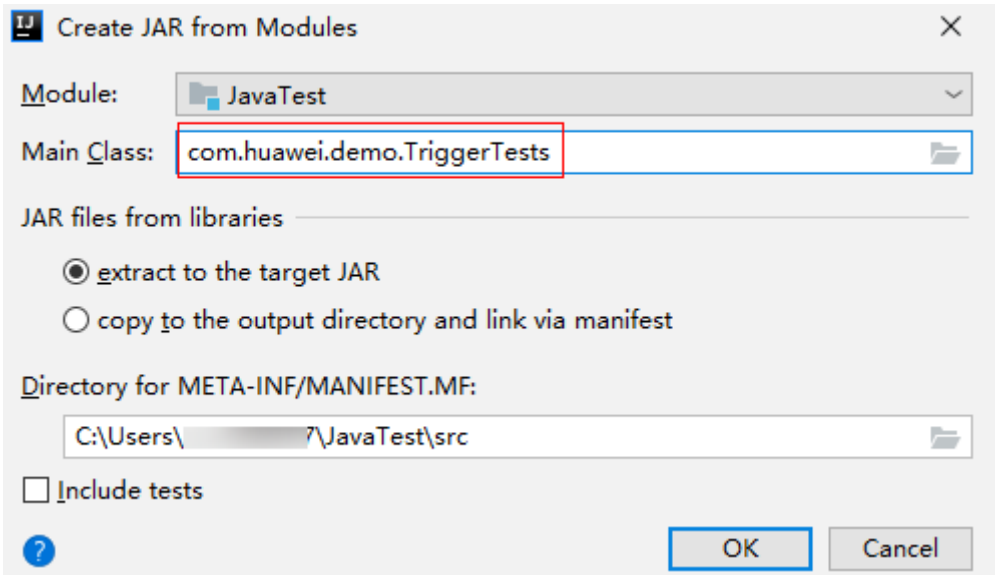
2. Choose **Artifacts** and click **+** to add artifacts, as shown in **Figure 5-19**.

Figure 5-19 Adding artifacts



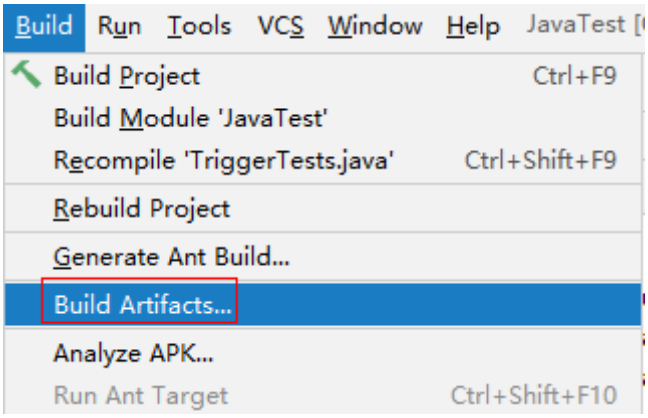
3. Add a main class, as shown in **Figure 5-20**.

Figure 5-20 Adding a main class



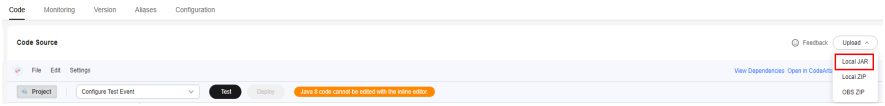
4. Choose **Build** > **Build Artifacts** to compile the JAR file, as shown in **Figure 5-21**.

Figure 5-21 Choosing Build Artifacts to compile the JAR file



Step 4 Log in to the FunctionGraph console, create a Java function, and upload the JAR file, as shown in **Figure 5-22**.

Figure 5-22 Uploading a JAR file



Step 5 Test the function.

1. Create a test event.

   Select **timer-event-template** from the **Event Template** drop-down list and click **Create**.

2. Click **Test**.

   The function execution result consists of three parts: function output (returned by **callback**), summary, and log (output by using the **System.out.println()** method).

----End

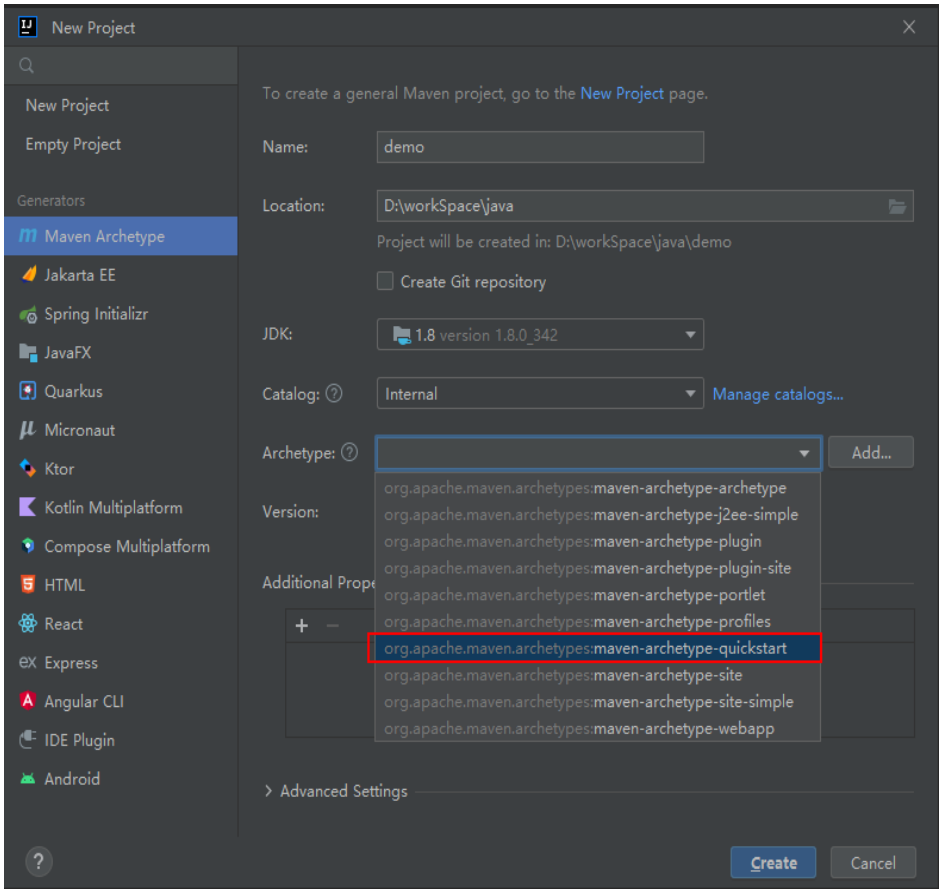
5.1.3 Developing Functions in Java (Using an IDEA Maven Project)

Perform the following procedure to develop a Java function:

Step 1 Create a function project.

1. Configure the IDEA and create a Maven project, as shown in **Figure 5-23**.

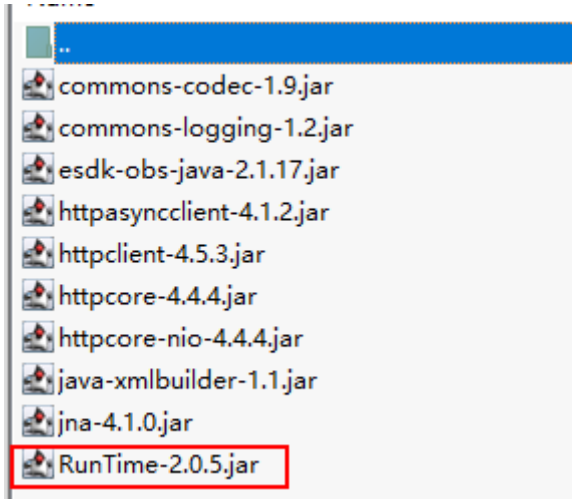
Figure 5-23 Creating a project



2. Add dependencies to the project.

   Download the **Java SDK** to a local development environment, and decompress the SDK package, as shown in **Figure 5-24**.

Figure 5-24 Decompressing the downloaded SDK



3. Copy **Runtime-2.0.5.jar** in the package to a local directory, for example, **d:\\runtime**. Then, run the following command in the CLI to install the runtime to the local Maven repository:

   ```
   mvn install:install-file -Dfile=d:\runtime\RunTime-2.0.5.jar -DgroupId=RunTime -DartifactId=RunTime -Dversion=2.0.5 -Dpackaging=jar
   ```


4. Configure the dependency in the **pom.xml** file.

```
<dependency>
<groupId>Runtime</groupId>
<artifactId>Runtime</artifactId>
<version>2.0.5</version>
</dependency>
```

5. Configure other dependencies. The following uses the OBS dependency as an example.

```
<dependency>
<groupId>com.huaweicloud</groupId>
<artifactId>esdk-obs-java</artifactId>
<version>3.21.4</version>
</dependency>
```

6. Add a plug-in to **pom.xml** to pack the code and dependencies together.

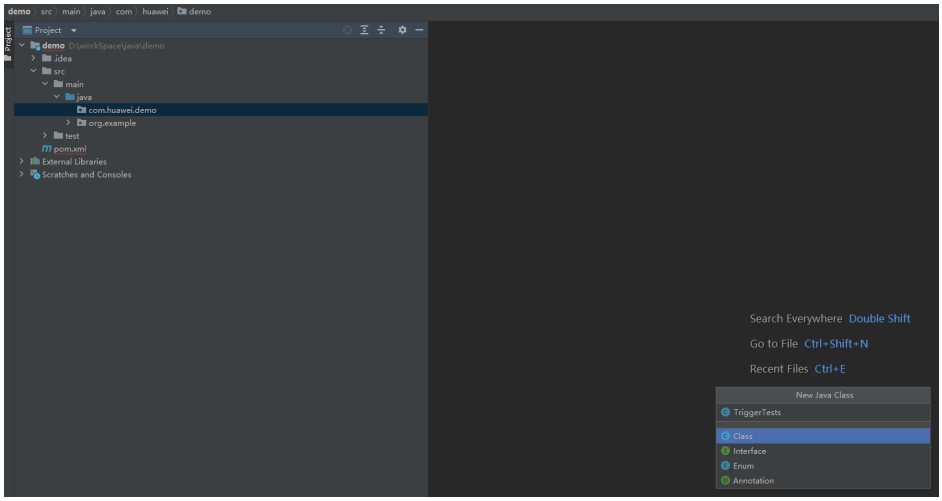
```
<build>
 <plugins>
  <plugin>
   <artifactId>maven-assembly-plugin</artifactId>
   <configuration>
    <descriptorRefs>
     <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
    <archive>
     <manifest>
      <mainClass>com.huawei.demo.TriggerTests</mainClass>
     </manifest>
    </archive>
    <finalName>${project.name}</finalName>
   </configuration>
   <executions>
    <execution>
     <id>make-assembly</id>
     <phase>package</phase>
     <goals><goal>single</goal>
     </goals>
    </execution>
   </executions>
  </plugin>
 </plugins>
</build>
```

Replace *mainClass* with the actual class.

Step 2 Create a function.

1. Create a package named **com.huawei.demo**, and then create a class named **TriggerTests** under the package, as shown in **Figure 5-25**.

Figure 5-25 Creating the TriggerTests class



2. Define the function handler in **TriggerTests.java**.

```
package com.huawei.demo;

import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.Map;

import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.lts.LTSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;

public class TriggerTests {
    public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context){
        System.out.println(event);
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "application/json");
        return new APIGTriggerResponse(200, headers, event.toString());
    }

    public String smnTest(SMNTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String dmsTest(DMSTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String timerTest(TimerTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    public String disTest(DISTriggerEvent event, Context context) throws
UnsupportedEncodingException{
        System.out.println(event);
        System.out.println(event.getMessage().getRecords()[0].getRawData());
        return "ok";
    }

    public String ltsTest(LTSTriggerEvent event, Context context) throws
UnsupportedEncodingException {
```



```
        System.out.println(event);
        event.getLts().getData();
        System.out.println("raw data: " + event.getLts().getRawData());
        return "ok";
    }
}
```

For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 3 Compile and pack the project file.

Run the following command to compile and pack the project file:

```
mvn package assembly:single
```

After compilation is complete, the **demo-jar-with-dependencies.jar** file is generated in the target directory.

Step 4 Log in to the FunctionGraph console, create a Java function, and upload the JAR file.

Step 5 Test the function.

1. Create a test event.

   Select **timer-event-template** from the **Event Template** drop-down list and click **Create**.

2. Click **Test**.

   The function execution result consists of three parts: function output (returned by **callback**), summary, and log (output by using the **System.out.println()** method).

----End

5.2 Java Template

```
package com.huawei.demo;
import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.lts.LTSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.Map;
public class TriggerTests {
  public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context) {
    System.out.println(event);
    Map<String, String> headers = new HashMap<>();
    headers.put("Content-Type", "application/json");
    return new APIGTriggerResponse(200, headers, event.toString());
  }
  public String smnTest(SMNTriggerEvent event, Context context) {
    System.out.println(event);
    return "ok";
  }
  public String dmsTest(DMSTriggerEvent event, Context context) {
    System.out.println(event);
    return "ok";
  }
```

```
  public String timerTest(TimerTriggerEvent event, Context context) {
    System.out.println(event);
    return "ok";
  }
  public String disTest(DISTriggerEvent event, Context context) throws UnsupportedEncodingException {
    System.out.println(event);
    System.out.println(event.getMessage().getRecords()[0].getRawData());
    return "ok";
  }
  public String ltsTest(LTSTriggerEvent event, Context context) throws UnsupportedEncodingException {
    System.out.println(event);
    System.out.println("raw data: " + event.getLts().getRawData());
    return "ok";
  }
}
```

5.3 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

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

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

6 Go

6.1 Developing an Event Function

Function Syntax

Syntax for creating a handler function in Go:

func Handler (payload []byte, ctx context.RuntimeContext)

- **Handler**: name of the handler function.
- **payload**: event parameter defined for the function. The parameter is in JSON format.
- **ctx**: runtime information provided for executing the function. For details, see **SDK APIs**.

Constraints

Results returned by using the **GetToken()**, **GetAccessKey()**, and **GetSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

SDK APIs

The Go SDK provides event, context, and logging APIs. **Download the Go SDK** (**Go SDK.sha256**).

- Event APIs

  Event structure definitions are added to the Go SDK. Currently, DIS, DDS, SMN, Timer, and APIG triggers are supported. The definitions make coding much simpler when triggers are required.

  a. **APIG Trigger Field Description**

     i. APIGTriggerEvent fields

Table 6-1 APIGTriggerEvent fields

Field Name	Description
IsBase64Encoded	Whether the body of an event is encoded using Base64.
HttpMethod	HTTP request method.
Path	HTTP request path.
Body	HTTP request body.
PathParameters	All path parameters.
RequestContext	API Gateway configurations (**APIGRequestContext object**).
Headers	HTTP request header.
QueryStringParameters	Query parameters.
UserData	User data set in the APIG custom authorizer.

Table 6-2 APIGRequestContext fields

Field Name	Description
ApiId	API ID.
RequestId	API request ID.
Stage	Name of the environment in which an API has been published.

ii. APIGTriggerResponse fields

Table 6-3 APIGTriggerResponse fields

Field Name	Description
Body	Message body.
Headers	HTTP response header to be returned.
StatusCode	HTTP status code. Type: **int**.
IsBase64Encoded	Whether the body has been encoded using Base64. Type: **bool**.

> NOTE
>
> APIGTriggerEvent provides the GetRawBody() method to obtain the body decoded using Base64. APIGTriggerResponse provides the SetBase64EncodedBody() method to set the body encoded using Base64.

b. **DIS Trigger Field Description**

Table 6-4 DISTriggerEvent fields

Field Name	Description
ShardID	Partition ID.
Message	DIS message body (**DISMessage structure**).
Tag	Function version.
StreamName	Stream name.

Table 6-5 DISMessage fields

Field Name	Description
NextPartitionCursor	Next partition cursor.
Records	Message records (**DISRecord structure**).
MillisBehindLatest	Reserved parameter.

Table 6-6 DISRecord fields

Field Name	Description
PartitionKey	Data partition.
Data	Data.
SequenceNumber	Sequence number (ID of each record).

c. **Kafka Trigger Field Description**

Table 6-7 KAFKATriggerEvent fields

Field Name	Description
InstanceId	Instance ID.
Records	Message records (**Table 6-8**).

Field Name	Description
TriggerType	Trigger type (Kafka).
Region	region
EventTime	Time when an event occurred (seconds).
EventVersion	Event version.

Table 6-8 KAFKARecord parameters

Field Name	Description
Messages	DMS message body.
TopicId	Topic ID.

d. **SMN Trigger Field Description**

Table 6-9 SMNTriggerEvent fields

Field Name	Description
Record	Message records (**SMNRecord structure**).

Table 6-10 SMNRecord fields

Field Name	Description
EventVersion	Event version. (Currently, the version is **1.0**.)
EventSubscriptionUrn	Subscription Uniform Resource Name (URN).
EventSource	Event source.
Smn	Message body (**SMNBody structure**).

Table 6-11 SMNBody fields

Field Name	Description
TopicUrn	Topic URN.

Field Name	Description
TimeStamp	Message timestamp.
MessageAtrributes	Message attribute set.
Message	Message body.
Type	Message format.
MessageId	Message ID.
Subject	Message topic.

e. **Timer Trigger Field Description**

Table 6-12 TimerTriggerEvent fields

Field Name	Description
Version	Version. (Currently, the version is **v1.0**.)
Time	Current time.
TriggerType	Trigger type (Timer).
TriggerName	Trigger name.
UserEvent	Additional information about the trigger.

NOTE

1. When using an APIG trigger, set the first parameter of the handler function (for example, **handler**) to **handler(APIGTriggerEvent event, Context context)**. For details about the constraints, see **Base64 Decoding and Response Structure**.
2. The preceding TriggerEvent methods have corresponding **set** methods, which are recommended for local debugging. DIS and LTS triggers have getRawData() methods, but do not have setRawData() methods.

- Context APIs

  The context APIs are used to obtain the context, such as agency AK/SK, current request ID, allocated memory space, and number of CPUs, required for executing a function.

  Table 6-13 describes the context APIs provided by FunctionGraph.

Table 6-13 Context methods

Method	Description
getRequestID( )	Obtains a request ID.
getRemainingTimeInMilligetRun-ningTimeInSecondsSeconds ( )	Obtains the remaining running time of a function.

Method	Description
getAccessKey( )	Obtains the AK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.**
getSecretKey( )	Obtains the SK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.**
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getFunctionName( )	Obtains the name of a function.
getRunningTimeInSeconds ( )	Obtains the timeout of a function.
getVersion( )	Obtains the version of a function.
getMemorySize( )	Obtains the allocated memory.
getCPUNumber( )	CPU usage of a function.
getPackage( )	Obtains a function group, that is, an app.

Method	Description
getToken( )	Obtains the token (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getLogger( )	Obtains the **logger** method provided by the context. By default, information such as the time and request ID is output.
getAlias	Obtains function alias.

- **Table 6-14** describes the logging API provided in the Go SDK.

  Table 6-14 Logging API

Method	Description
RuntimeLogger()	• Records user input logs by using the method **Logf(format string, args ...interface{})**. • This method outputs logs in the format of *Time-Request ID-Output*, for example, **2017-10-25T09:10:03.328Z 473d369d-101a-445e-a7a8-315cca788f86 test log output**.

Developing a Go Function

Log in to the Linux server where the Go 1.x SDK has been installed. (Currently, Ubuntu 14.04, Ubuntu 16.04, SUSE 11.3, SUSE 12.0, and SUSE 12.1 are supported.)

- If the Go version (1.11.1 or later) supports go mod, perform the following steps to compile and package code:

Step 1 Create a temporary directory, for example, **/home/fssgo**, decompress the **Go SDK** of FunctionGraph to the created directory, and enable the go module function.

$ mkdir -p /home/fssgo

$ unzip functiongraph-go-runtime-sdk-1.0.1.zip -d /home/fssgo

$ export GO111MODULE="on"

Step 2 Generate the **go.mod** file in the **/home/fssgo** directory. Assume that the module name is **test**:

$ go mod init test

Step 3 Edit the **go.mod** file in the **/home/fssgo** directory as required (that is, add the content in bold).

```
module test

go 1.14

require (
    huaweicloud.com/go-runtime v0.0.0-00010101000000-000000000000
)

replace (
    huaweicloud.com/go-runtime => ./go-runtime
)
```

Step 4 Create a function file under the **/home/fssgo** directory and implement the following interface:

func Handler(payload []byte, ctx context.RuntimeContext) (interface{}, error)

In this interface, **payload** is the body of a client request, and **ctx** is the runtime context object provided for executing a function. For more information about the methods, see **Table 6-13**. The following uses **test.go** as an example.

```
package main

import (
    "fmt"
    "huaweicloud.com/go-runtime/go-api/context"
    "huaweicloud.com/go-runtime/pkg/runtime"
    "huaweicloud.com/go-runtime/events/apig"
    "huaweicloud.com/go-runtime/events/cts"
    "huaweicloud.com/go-runtime/events/dds"
    "huaweicloud.com/go-runtime/events/dis"
    "huaweicloud.com/go-runtime/events/kafka"
    "huaweicloud.com/go-runtime/events/lts"
    "huaweicloud.com/go-runtime/events/smn"
    "huaweicloud.com/go-runtime/events/timer"
    "encoding/json"
)

func ApigTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var apigEvent apig.APIGTriggerEvent
    err := json.Unmarshal(payload, &apigEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", apigEvent.String())
    apigResp := apig.APIGTriggerResponse{
        Body: apigEvent.String(),
        Headers: map[string]string {
            "content-type": "application/json",
        },
        StatusCode: 200,
    }
    return apigResp, nil
}

func CtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ctsEvent cts.CTSTriggerEvent
    err := json.Unmarshal(payload, &ctsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ctsEvent.String())
    return "ok", nil
}
```



```
func DdsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ddsEvent dds.DDSTriggerEvent
    err := json.Unmarshal(payload, &ddsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ddsEvent.String())
    return "ok", nil
}

func DisTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var disEvent dis.DISTriggerEvent
    err := json.Unmarshal(payload, &disEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", disEvent.String())
    return "ok", nil
}

func KafkaTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var kafkaEvent kafka.KAFKATriggerEvent
    err := json.Unmarshal(payload, &kafkaEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", kafkaEvent.String())
    return "ok", nil
}

func LtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ltsEvent lts.LTSTriggerEvent
    err := json.Unmarshal(payload, &ltsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ltsEvent.String())
    return "ok", nil
}

func SmnTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var smnEvent smn.SMNTriggerEvent
    err := json.Unmarshal(payload, &smnEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", smnEvent.String())
    return "ok", nil
}

func TimerTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var timerEvent timer.TimerTriggerEvent
    err := json.Unmarshal(payload, &timerEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    return timerEvent.String(), nil
}

func main() {
    runtime.Register(ApigTest)
}
```


Constraints:

1. If the **error** parameter returned by a function is not **nil**, the function execution fails.
2. If the **error** parameter returned by a function is **nil**, FunctionGraph supports only the following types of values:

   nil: The HTTP response body is empty.

   []byte: The content in this byte array is the body of an HTTP response.

   string: The content in this string is the body of an HTTP response.

   Other: FunctionGraph returns a value for JSON encoding, and uses the encoded object as the body of an HTTP response. The **Content-Type** header of the HTTP response is set to **application/json**.
3. **The preceding example uses an APIG trigger as an example. For other trigger types, you need to modify the content of the main function. For example, change the CTS trigger to runtime.Register(CtsTest). Currently, only one entry can be registered.**
4. For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 5 Compile and package the function code.

After completing the function code, compile and package it as follows:

1. Compile the code.

   $ cd /home/fssgo

   $ go build -o handler test.go

   NOTE

   The handler can be customized, which is used as the function entry.
2. Package the code.

   $ zip fss_examples_go1.x.zip handler

----End

- If the Go version (earlier than 1.11.1) does not support go mod, perform the following steps to compile and package code:

Step 1 Create a temporary directory, for example, **/home/fssgo/src/huaweicloud.com**, and decompress the **Go SDK** to the created directory.

$ mkdir -p /home/fssgo/src/huaweicloud.com

$ unzip functiongraph-go-runtime-sdk-1.0.1.zip -d /home/fssgo/src/huaweicloud.com

Step 2 Create a function file under the **/home/fssgo/src** directory and implement the following interface:

func Handler(payload []byte, ctx context.RuntimeContext) (interface{}, error)

In this interface, **payload** is the body of a client request, and **ctx** is the runtime context object provided for executing a function. For more information about the methods, see the SDK APIs. The following uses **test.go** as an example.


```
package main

import (
    "fmt"
    "huaweicloud.com/go-runtime/go-api/context"
    "huaweicloud.com/go-runtime/pkg/runtime"
    "huaweicloud.com/go-runtime/events/apig"
    "huaweicloud.com/go-runtime/events/cts"
    "huaweicloud.com/go-runtime/events/dds"
    "huaweicloud.com/go-runtime/events/dis"
    "huaweicloud.com/go-runtime/events/kafka"
    "huaweicloud.com/go-runtime/events/lts"
    "huaweicloud.com/go-runtime/events/smn"
    "huaweicloud.com/go-runtime/events/timer"
    "encoding/json"
)

func ApigTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var apigEvent apig.APIGTriggerEvent
    err := json.Unmarshal(payload, &apigEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", apigEvent.String())
    apigResp := apig.APIGTriggerResponse{
        Body: apigEvent.String(),
        Headers: map[string]string {
            "content-type": "application/json",
        },
        StatusCode: 200,
    }
    return apigResp, nil
}

func CtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ctsEvent cts.CTSTriggerEvent
    err := json.Unmarshal(payload, &ctsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ctsEvent.String())
    return "ok", nil
}

func DdsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ddsEvent dds.DDSTriggerEvent
    err := json.Unmarshal(payload, &ddsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ddsEvent.String())
    return "ok", nil
}

func DisTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var disEvent dis.DISTriggerEvent
    err := json.Unmarshal(payload, &disEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", disEvent.String())
    return "ok", nil
}

func KafkaTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
```



```
    var kafkaEvent kafka.KAFKATriggerEvent
    err := json.Unmarshal(payload, &kafkaEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", kafkaEvent.String())
    return "ok", nil
}

func LtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ltsEvent lts.LTSTriggerEvent
    err := json.Unmarshal(payload, &ltsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ltsEvent.String())
    return "ok", nil
}

func SmnTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var smnEvent smn.SMNTriggerEvent
    err := json.Unmarshal(payload, &smnEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", smnEvent.String())
    return "ok", nil
}

func TimerTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var timerEvent timer.TimerTriggerEvent
    err := json.Unmarshal(payload, &timerEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    return timerEvent.String(), nil
}

func main() {
    runtime.Register(ApigTest)
}
```

Constraints:

1. If the **error** parameter returned by a function is not **nil**, the function execution fails.
2. If the **error** parameter returned by a function is **nil**, FunctionGraph supports only the following types of values:

   nil: The HTTP response body is empty.

   []byte: The content in this byte array is the body of an HTTP response.

   string: The content in this string is the body of an HTTP response.

   Other: FunctionGraph returns a value for JSON encoding, and uses the encoded object as the body of an HTTP response. The **Content-Type** header of the HTTP response is set to **application/json**.
3. **The preceding example uses an APIG trigger as an example. For other trigger types, you need to modify the content of the main function. For example, change the CTS trigger to runtime.Register(CtsTest). Currently, only one entry can be registered.**

4. For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 3 Compile and package the function code.

After completing the function code, compile and package it as follows:

1. Set environment variables **GOROOT** and **GOPATH**.

   $ export GOROOT=/usr/local/go (Assume that the Go SDK is installed under the /usr/local/go directory.)

   $ export PATH=$GOROOT/bin:$PATH

   $ export GOPATH=/home/fssgo

2. Compile the function code.

   $ cd /home/fssgo

   $ go build -o handler test.go

   NOTE

   The handler can be customized, which is used as the function entry.

3. Package the code.

   $ zip fss_examples_go1.x.zip handler

- Creating a function

  Log in to the FunctionGraph console, create a Go 1.x function, and upload the **fss_examples_go1.x.zip** file.

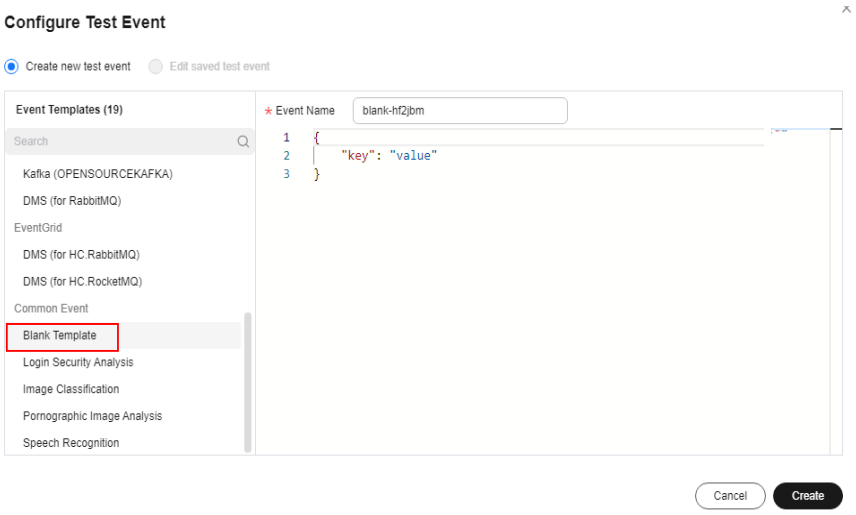
  If you edit code in Go, package the compiled file into a ZIP file, and ensure that the name of the compiled file is consistent with the handler plugin name. For example, if the name of the binary file is **handler**, set the handler plugin name to **handler**. The handler must be consistent with that defined in **Step 3.2**.

- Testing the function

  a. Create a test event.

  On the function details page that is displayed, click **Configure Test Event**. Configure the test event information, as shown in **Figure 6-1**, and then click **Create**.

  Figure 6-1 Configuring a test event



b. On the function details page, select the configured test event, and click **Test**.

- Executing the function

  The function execution result consists of three parts: function output (returned by **callback**), summary, and logs (output by using the **fmt.Println()** method).

----End

Execution Result

The execution result consists of the function output, summary, and log output.

Table 6-15 Description of the execution result

Parameter	Successful Execution	Failed Execution
Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **errorType** is returned. The format is as follows: `{ "errorMessage": "", "errorType":"", }` **errorMessage**: Error message returned by the runtime. **errorType**: Error type.
Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

7 C#

7.1 Developing an Event Function

7.1.1 C# Function Development

Function Syntax

FunctionGraph supports C#(.NET Core 2.1), C#(.NET Core 3.1), C#(.NET Core 6.0) and C# (.NET Core 8.0) (only available in **ME-Riyadh** and **TR-Istanbul**).

C# function syntax: *Scope Return parameter Function name (User-defined parameter, Context)*

- *Scope*: It must be defined as **public** for the function that FunctionGraph invokes to execute your code.
- *Return parameter*: user-defined output, which is converted into a character string and returned as an HTTP response.
- *Function name*: user-defined function name. The name must be consistent with that you define when creating a function.
- Event: event parameter defined for the function.
- **context**: runtime information provided for executing the function. For details, see the description of SDK APIs.

  The **HC.Serverless.Function.Common** library needs to be referenced when you deploy a project in FunctionGraph. For details about the **IFunctionContext** object, see the context description.

  When creating a C# function, you need to define a method as the handler of the function. The method can access the function by using specified IFunctionContext parameters. Example:

```
public Stream handlerName(Stream input,IFunctionContext context)
{
    // TODO
}
```

For a C# function, the handler must be named in the format of *[assembly]*::*[namespace].[class name]*::*[execution function name]*. Example:

CsharpDemo::CsharpDemo.Program::MyFunc. For details about handler configuration, see **handler parameter**.

NOTE

You are advised to use C# (.NET Core 3.1).

Function Handler

ASSEMBLY::NAMESPACE.CLASSNAME::METHODNAME

- **ASSEMBLY**: name of the .NET assembly file for your application, for example, **HelloCsharp**.
- **NAMESPACE** and **CLASSNAME**: names of the namespace and class to which the handler function belongs.
- **METHODNAME**: name of the handler function. Example:

  Set the handler to **HelloCsharp::Example.Hello::Handler** when you create a function.

SDK APIs

- Context APIs

  Table 7-1 describes the provided context attributes.

  Table 7-1 Context objects

Attribute	Description
String RequestId	Request ID.
String ProjectId	Project ID.
String PackageName	Name of the group to which the function belongs.
String FunctionName	Function name.
String FunctionVersion	Function version.
Int MemoryLimitInMb	Allocated memory.
Int CpuNumber	CPU usage of a function.
String Accesskey	Obtains the AK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the String AccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.**

Attribute	Description
String Secretkey	Obtains the SK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the String SecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.**
String SecurityAccessKey	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
String SecuritySecretKey	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
String SecurityToken	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
String Token	Obtains the token (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
Int RemainingTimeInMilli-Seconds	Remaining running time of a function.
String GetUserData(string key,string defvalue=" ")	Uses keys to obtain the values passed by environment variables.

- Logging APIs

  The following table describes the logging APIs provided in the C# SDK.

  Table 7-2 Logging APIs

Method	Description

Log(string message)	Creates a logger object by using context.
	var logger = context.Logger;
	logger.Log("hello CSharp runtime test(v1.0.2)");

Logf(string format, args ...interface{})	Creates a logger object by using context.
	var logger = context.Logger;
	var version = "v1.0.2"
	logger.Logf("hello CSharp runtime test({0})", version);

Developing a C# Function

Constraints:

- If you use sample package **fss_example_csharp2.0**, skip **Step 1** and **Step 2**, go to **Step 3**, and modify the function handler to **MyCsharpPro::src.Program::myFunc**.
- **For details about the constraints for the APIG event source, see Base64 Decoding and Response Structure.**

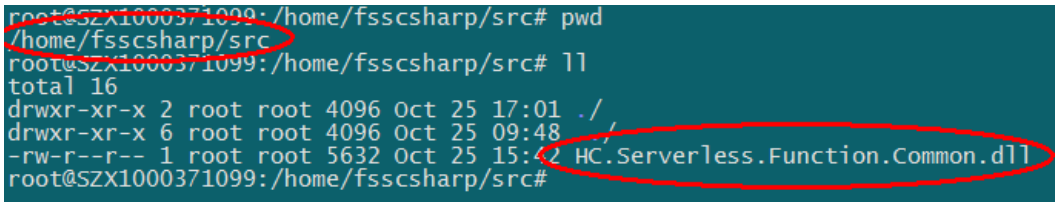
The following describes how to create a C# (.NET Core 2.0) function in Linux.

Step 1 Create a C# project.

Log in to a Linux server on which the .NET SDK and required running environment have been installed. Create the **/home/fsscsharp/src** directory, download the **.dll** file, and decompress it to this directory as shown in **Figure 7-1**.

This document uses the DLL function of the fssCsharp2.0-1.0.1 version as an example. The DLL functions of different versions are the same.

Figure 7-1 Decompressing the .dll package



Run the **dotnet --info** command to check whether the .NET environment has been installed. The command output is as follows:

```
root@SZX1000371099:/home/fsscsharp/src# dotnet --info
.NET Command Line Tools (2.1.202)

Product Information:
 Version:            2.1.202
 Commit SHA-1 hash:  281caedada

Runtime Environment:
 OS Name:     ubuntu
 OS Version:  14.04
 OS Platform: Linux
 RID:         ubuntu.14.04-x64
 Base Path:   /home/lusinking/dotnetdev/sdk/2.1.202/

Microsoft .NET Core Shared Framework Host

  Version  : 2.0.9
  Build    : 1632fa1589b0eee3277a8841ce1770e554ece037
```

Run the following command to create and initialize a console application project:

"dotnet new console -n project_name"

Example command:

```
dotnet new console -n MyCsharpPro
```

Create a handler function in the **Program.cs** code file in the **/home/fsscsharp/src/MyCsharpPro** directory. The input is the body of a client request, and the context

is a runtime context object provided by FunctionGraph. For more information about context objects, see the context API description. The code is as follows:

```
using HC.Serverless.Function.Common;
using System;
using System.IO;
using System.Text;

namespace src
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
        public Stream myFunc(Stream input,IFunctionContext context)
        {
            string payload = "";
            if (input != null && input.Length > 0)
            {
                byte[] buffer = new byte[input.Length];
                input.Read(buffer, 0, (int)(input.Length));
                payload = Encoding.UTF8.GetString(buffer);
            }
            var ms = new MemoryStream();
            using (var sw = new StreamWriter(ms))
            {
                sw.WriteLine("CSharp runtime test(v1.0.2)");
                sw.WriteLine("====================================");
                sw.WriteLine("Request Id:       {0}", context.RequestId);
                sw.WriteLine("Function Name:    {0}", context.FunctionName);
                sw.WriteLine("Function Version: {0}", context.FunctionVersion);
                sw.WriteLine("Project:          {0}", context.ProjectId);
                sw.WriteLine("Package:          {0}", context.PackageName);
                sw.WriteLine("Security Access Key:       {0}", context.SecurityAccessKey);
                sw.WriteLine("Security Secret Key:       {0}", context.SecuritySecretKey);
                sw.WriteLine("Security Token:            {0}", context.SecurityToken);
                sw.WriteLine("Token:            {0}", context.Token);
                sw.WriteLine("User data(ud-a):  {0}", context.GetUserData("ud-a"));
                sw.WriteLine("User data(ud-notexist): {0}", context.GetUserData("ud-notexist", ""));
                sw.WriteLine("User data(ud-notexist-default): {0}", context.GetUserData("ud-notexist", "default
value"));
                sw.WriteLine("====================================");

                var logger = context.Logger;
                logger.Logf("Hello CSharp runtime test(v1.0.2)");
                sw.WriteLine(payload);
            }
            return new MemoryStream(ms.ToArray());
        }
    }
}
```

Step 2 Compile the C# project.

Manually add the **dll** reference (relative path of **dll** in HinPath) provided by FunctionGraph to the project configuration file **MyCsharpPro.csproj**. The code is as follows:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="HC.Serverless.Function.Common, Version=1.0.0.0, Culture=neutral,
```



```
PublicKeyToken=null">
      <HintPath>../HC.Serverless.Function.Common.dll</HintPath>
    </Reference>
  </ItemGroup>
</Project>
```

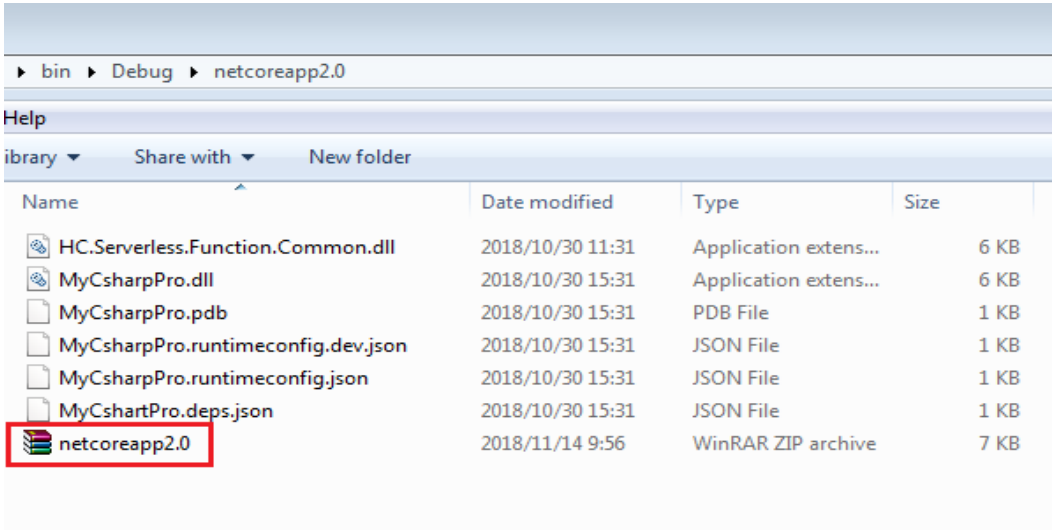
Run the **dotnet build** command to compile the project. The command output is as follows:

```
root@SZX1000371099:/home/fsscsharp/src/MyCsharpPro# vi MyCsharpPro.csproj
root@SZX1000371099:/home/fsscsharp/src/MyCsharpPro# dotnet build
Microsoft (R) Build Engine version 15.7.179.6572 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 54.28 ms for /home/fsscsharp/src/MyCsharpPro/MyCsharpPro.csproj.
  MyCsharpPro -> /home/fsscsharp/src/MyCsharpPro/bin/Debug/netcoreapp2.0/MyCsharpPro.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.47
```

Step 3 Deploy the C# project to FunctionGraph.

Use the SSH tool to copy and package the compiled files, as shown in **Figure 7-2**.

Figure 7-2 Packaging the compiled files



Create a function and upload the ZIP file.

Execute the function. The function execution result consists of three parts: function output (returned by **callback**), summary, and logs (output by using the **Console.WriteLine()** method).

----End

Execution Result

The execution result consists of the function output, summary, and log output.

Table 7-3 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **errorType** is returned. The format is as follows:
		{
		"errorMessage": "",
		"errorType": ""
		}
		errorMessage: Error message returned by the runtime.
		errorType: Error type.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

7.1.2 JSON Serialization and Deserialization

7.1.2.1 Using .NET Core CLI

C# supports JSON serialization and deserialization interfaces and provides the **HC.Serverless.Function.Common.JsonSerializer.dll** file.

The interfaces are as follows:

T Deserialize<T>(Stream ins): Deserializes data into objects of function programs.

Stream Serialize<T>(T value): Serializes data to the returned response payload.

The following shows how to create a project named **test** using .NET Core 2.1. The procedure is similar for C# (.NET Core 6.0, which is only available in **LA-Mexico City2**) and NET Core 3.1. The .NET SDK 2.1 has been installed in the execution environment.

Creating a Project

1. Run the following command to create the **/tmp/csharp/projects /tmp/csharp/release** directory:
   ```
   mkdir -p  /tmp/csharp/projects;mkdir -p  /tmp/csharp/release
   ```
2. Run the following command to go to the **/tmp/csharp/projects/** directory:
   ```
   cd /tmp/csharp/projects/
   ```


3. Create the project file **test.csproj** with the following content:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
    <RootNamespace>test</RootNamespace>
    <AssemblyName>test</AssemblyName>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="HC.Serverless.Function.Common">
      <HintPath>HC.Serverless.Function.Common.dll</HintPath>
    </Reference>
    <Reference Include="HC.Serverless.Function.Common.JsonSerializer">
      <HintPath> HC.Serverless.Function.Common.JsonSerializer.dll</HintPath>
    </Reference>
  </ItemGroup>
</Project>
```

Generating a Code Library

1. Download and upload the **.dll** file.

   Upload the **HC.Serverless.Function.Common.dll**, **HC.Serverless.Function.Common.JsonSerializer.dll**, and **Newtonsoft.Json.dll** files in the package to the **/tmp/csharp/projects/** directory.

2. Create the **Class1.cs** file in the **/tmp/csharp/projects/** directory. The code is as follows:

```
using HC.Serverless.Function.Common;
using System;
using System.IO;

namespace test
{
    public class Class1
    {
        public Stream ContextHandlerSerializer(Stream input, IFunctionContext context)
        {
            var logger = context.Logger;
            logger.Logf("CSharp runtime test(v1.0.2)");
            JsonSerializer test = new JsonSerializer();
            TestJson Testjson = test.Deserialize<TestJson>(input);
            if (Testjson != null)
            {
                logger.Logf("json Deserialize KetTest={0}", Testjson.KetTest);

            }
            else
            {
                return null;
            }

          return test.Serialize<TestJson>(Testjson);
        }

        public class TestJson
        {
            public string KetTest { get; set; }//Define the attribute of the serialization class as KetTest.

        }
    }
}
```

3. Run the following command to generate a code library:

```
/home/tools/dotnetcore-sdk/dotnet-sdk-2.1.302-linux-x64/dotnet build /tmp/csharp/projects/test.csproj -c Release -o /tmp/csharp/release
```


> **NOTE**
>
> .NET directory: /home/tools/dotnetcore-sdk/dotnet-sdk-2.1.302-linux-x64/dotnet

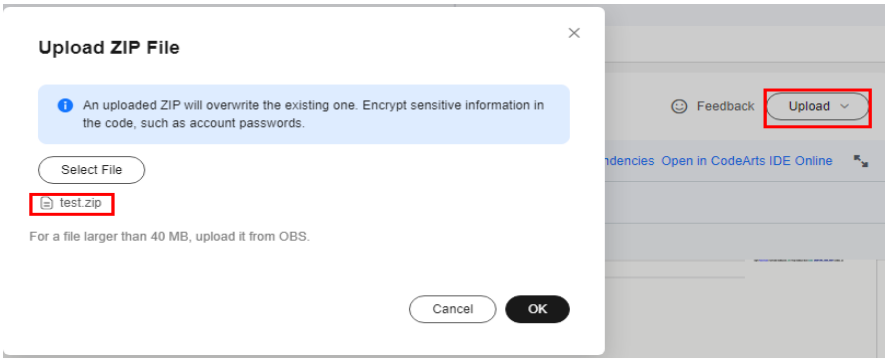
4. Run the following command to go to the **/tmp/csharp/release** directory:
   ```
   cd /tmp/csharp/release
   ```
5. View the compiled .dll files in the **/tmp/csharp/release** directory.
   ```
   -rw-r--r-- 1 root root 468480 Jan 21 16:40 Newtonsoft.Json.dll
   -rw-r--r-- 1 root root   5120 Jan 21 16:40 HC.Serverless.Function.Common.JsonSerializer.dll
   -rw-r--r-- 1 root root   5120 Jan 21 16:40 HC.Serverless.Function.Common.dll
   -rw-r--r-- 1 root root    232 Jan 21 17:10 test.pdb
   -rw-r--r-- 1 root root   3584 Jan 21 17:10 test.dll
   -rw-r--r-- 1 root root   1659 Jan 21 17:10 test.deps.json
   ```
6. Create the **test.runtimeconfig.json** file in the **/tmp/csharp/release** directory. The file content is as follows:
   ```
   {
     "runtimeOptions": {
       "framework": {
         "name": "Microsoft.NETCore.App",
         "version": "2.1.0"
       }
     }
   }
   ```

   > **NOTE**
   >
   > - The name of the ***.runtimeconfig.json** file is the name of an assembly.
   > - The **Version** parameter in the file indicates the version number of the target framework. If the framework is .NET Core 2.0, enter **2.0.0**. If the framework is .NET Core 2.1, enter **2.1.0**.
   > - For .NET Core 2.0, check whether **Newtonsoft.Json** is referenced in the ***.deps.json** file. If **Newtonsoft.Json** is not referenced, reference it manually.
   >
   >   1. Reference the following content in **targets**:
   >      ```
   >      "Newtonsoft.Json/9.0.0.0": {
   >          "runtime": {
   >            "Newtonsoft.Json.dll": {
   >              "assemblyVersion": "9.0.0.0",
   >              "fileVersion": "9.0.1.19813"
   >            }
   >          }
   >        }
   >      ```
   >   2. Reference the following content in **libraries**:
   >      ```
   >      "Newtonsoft.Json/9.0.0.0": {
   >          "type": "reference",
   >          "serviceable": false,
   >          "sha512": ""
   >        }
   >      ```
7. Run the following command to package the **test.zip** file in the **/tmp/csharp/release** directory:
   ```
   zip -r test.zip ./*
   ```

Test Example

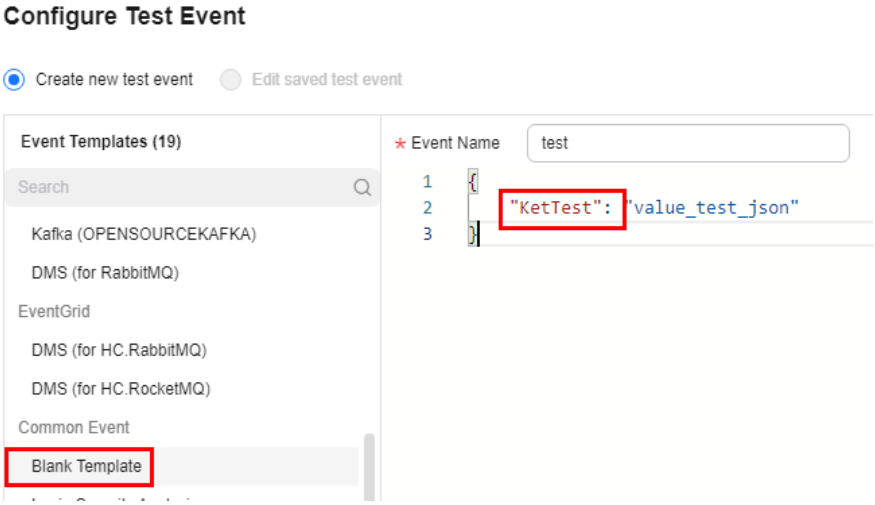
1. Create a C# (.NET 2.1) function on the FunctionGraph console and upload the **test.zip** file, as shown in **Figure 7-3**.

Figure 7-3 Uploading the code package



2. Configure a test event, as shown in **Figure 7-4**. The key must be set to **KetTest**, and the value can be customized. (**The test string must be in JSON format.**)

Figure 7-4 Configuring a test event



NOTE

The attribute of the serialization class must be defined as **KetTest**.

3. Click **Test** and view the execution result.

7.1.2.2 Using Visual Studio

C# supports JSON serialization and deserialization interfaces and provides the **HC.Serverless.Function.Common.JsonSerializer.dll** file.

The interfaces are as follows:

T Deserialize<T>(Stream ins): Deserializes data into objects of function programs.

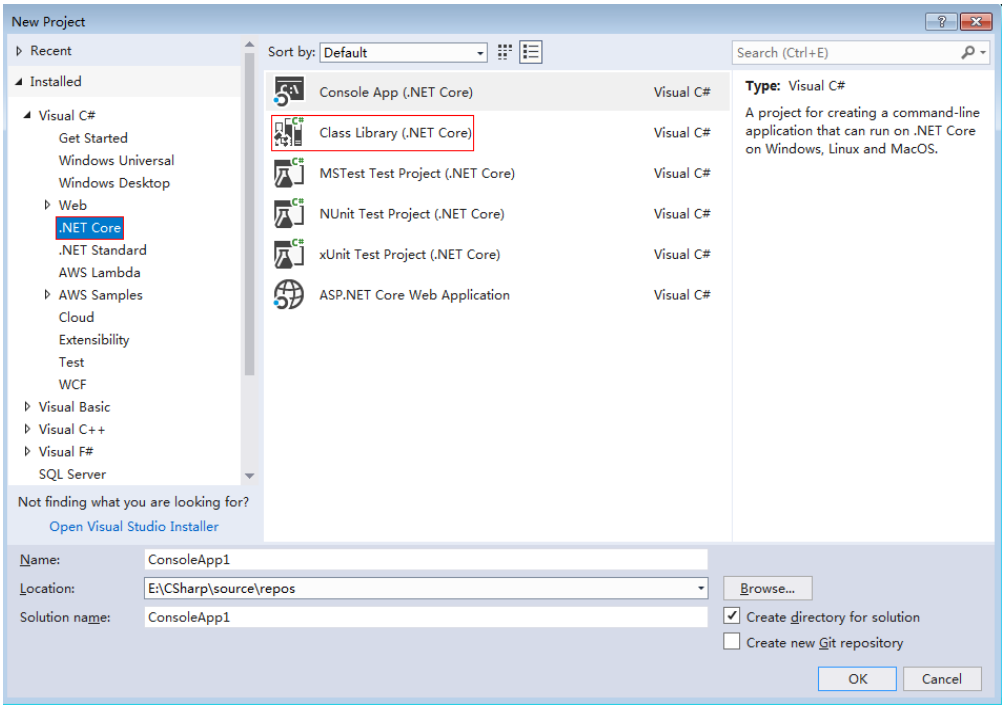
Stream Serialize<T>(T value): Serializes data to the returned response payload.

The following shows how to create a project named **test** using .NET Core 2.0 in Visual Studio 2017. The procedure is similar for .NET Core 2.1, C# (.NET Core 6.0, which is available only in **LA-Mexico City2**), and NET Core 3.1.

Creating a Project

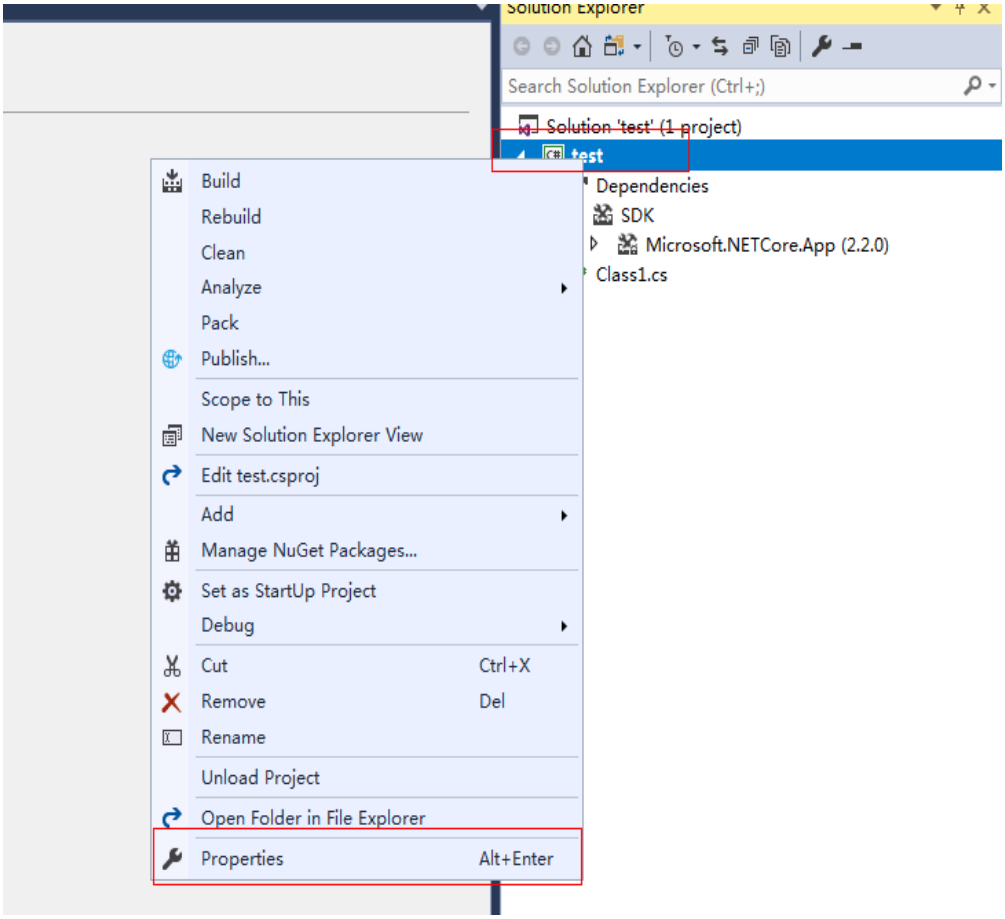
1. On the toolbar, choose **File** > **New** > **Project**, select **.NET Core**, select **Class Library (.NET Core)**, and change the project name to **test**, as shown in **Figure 7-5**.

   Figure 7-5 Creating a project



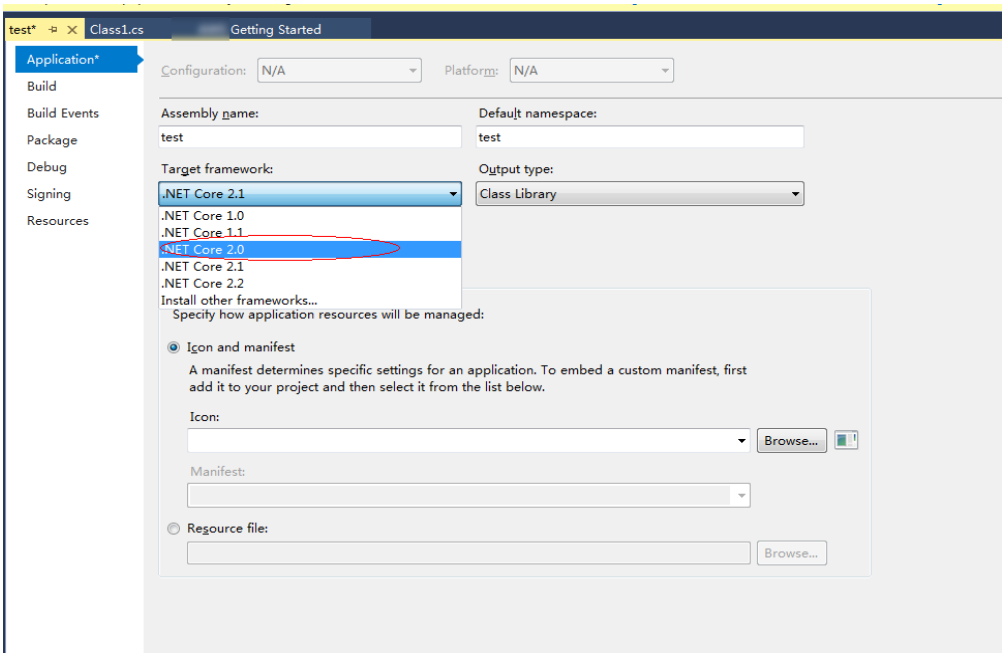
2. Select the **test** project in the navigation pane, right-click, and then choose **Properties**, as shown in **Figure 7-6**.

Figure 7-6 Properties



3. Choose **Application** and then set **Target framework** to **.NET Core 2.0**, as shown in **Figure 7-7**.

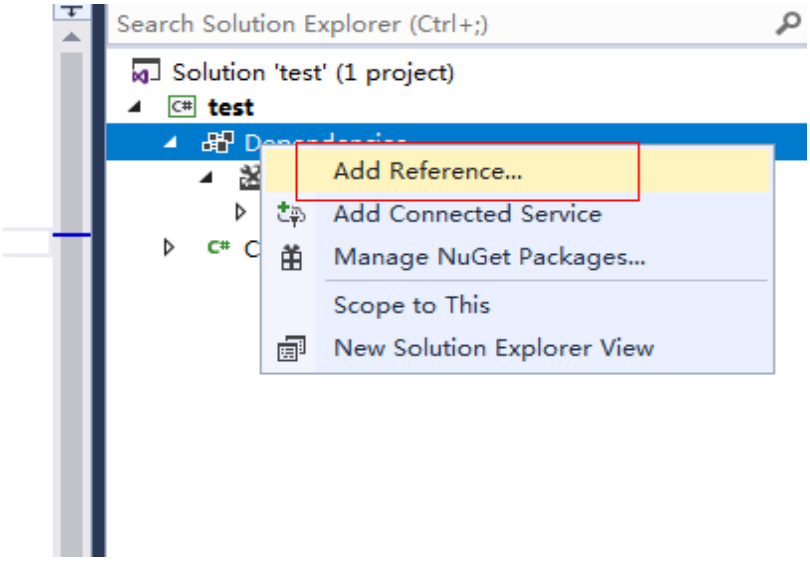
Figure 7-7 Selecting a target framework



Adding a Reference

1. Select the **test** project in **Search Solution Explorer**, right-click, and then choose **Add Reference** to reference the downloaded **.dll file**, as shown in **Figure 7-8**.

Figure 7-8 Adding a reference

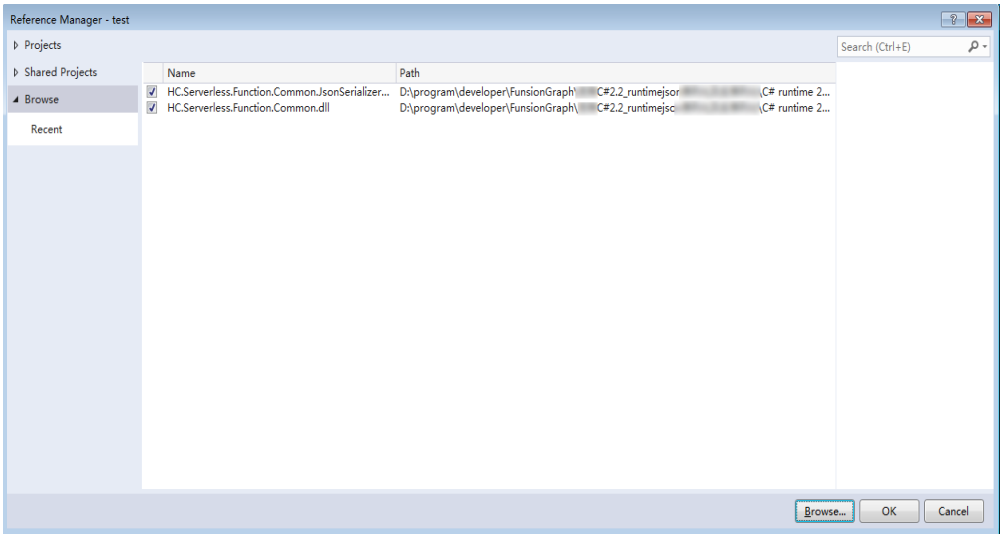


NOTE

Store **HC.Serverless.Function.Common.dll**, **HC.Serverless.Function.Common.JsonSerializer.dll**, and **Newtonsoft.Json.dll** in a lib file.

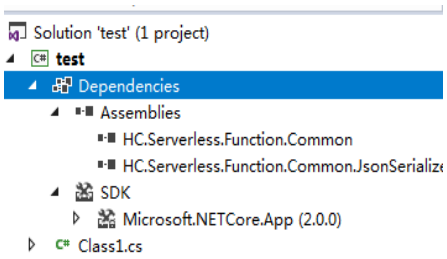
2. Choose **Browse**, click **Browse(B)**, reference the **HC.Serverless.Function.Common.dll** and **HC.Serverless.Function.Common.JsonSerializer.dll** files, and click **OK**, as shown in **Figure 7-9**.

Figure 7-9 Referencing files



3. View the references, as shown in **Figure 7-10**.

Figure 7-10 References



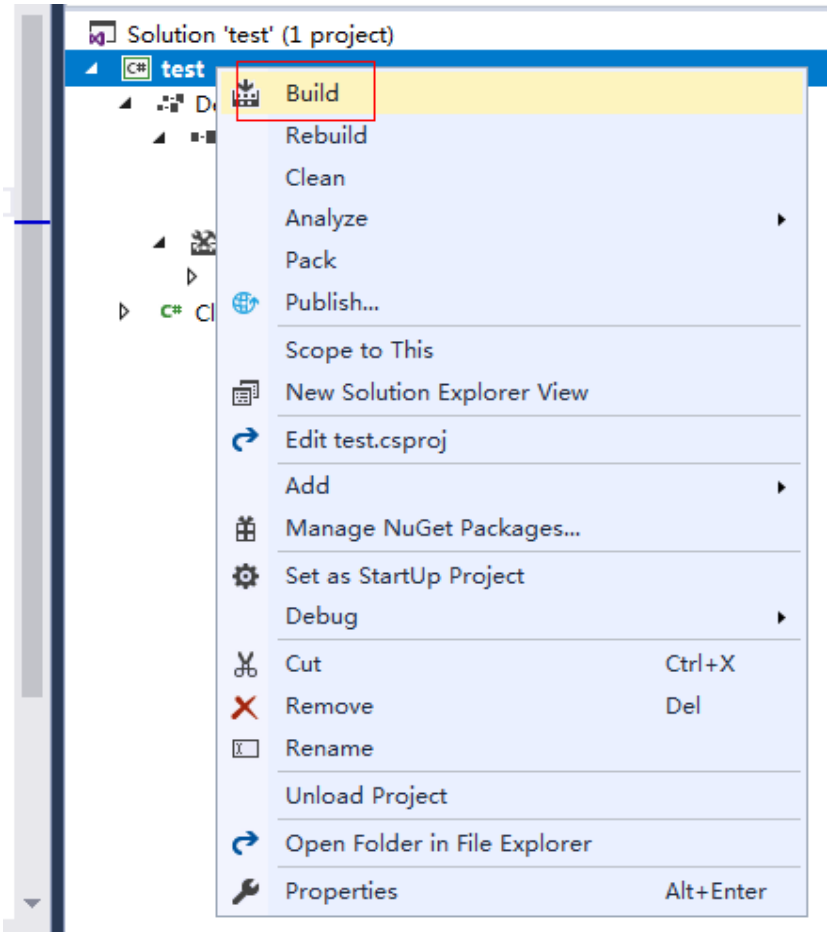
Packing Code

Sample code:

```
using HC.Serverless.Function.Common;
using System;
using System.IO;

namespace test
{
    public class Class1
    {
        public Stream ContextHandlerSerializer(Stream input, IFunctionContext context)
        {
            var logger = context.Logger;
            logger.Logf("CSharp runtime test(v1.0.2)");
            JsonSerializer test = new JsonSerializer();
            TestJson Testjson = test.Deserialize<TestJson>(input);
            if (Testjson != null)
            {
                logger.Logf("json Deserialize KetTest={0}", Testjson.KetTest);
            }

            return test.Serialize<TestJson>(Testjson);
        }

        public class TestJson
        {
            public string KetTest { get; set; }//Define the attribute of the serialization class as KetTest.

        }
    }
}
```

1. Right-click the **test** project and choose **Build**, as shown in **Figure 7-11**.

Figure 7-11 Build



2. Copy the path **C:\Users\xxx\source\repos\test\test\bin\Release\netcoreapp2.0** of the .dll files, as shown in **Figure 7-12**.

Figure 7-12 Path of .dll files

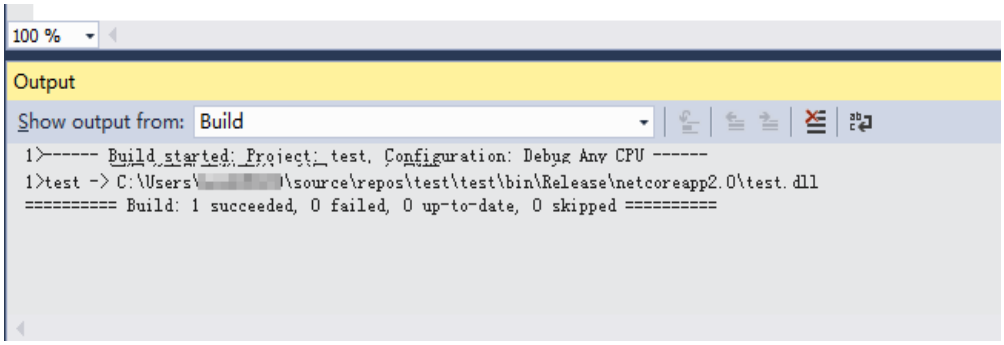
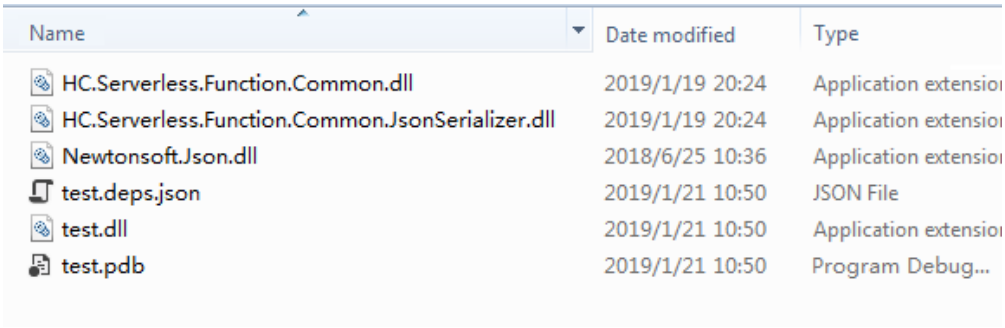


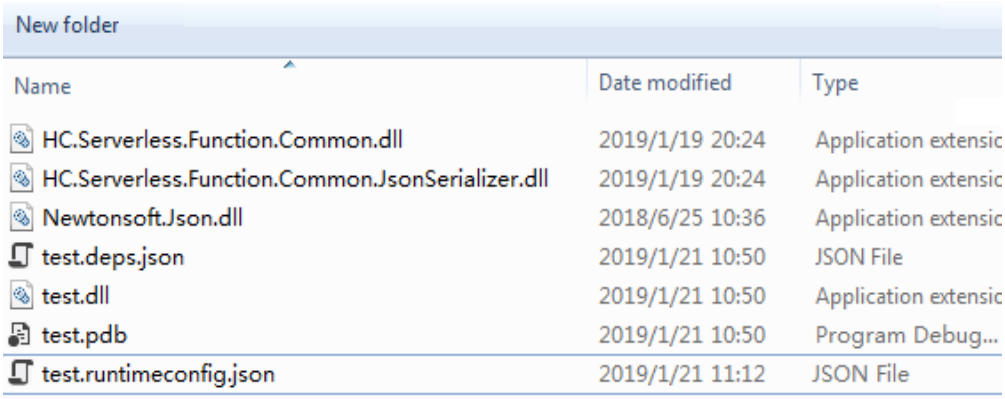
Figure 7-13 shows the files in the path.

Figure 7-13 Files



3. Create a **test.runtimeconfig.json** file in the path, as shown in **Figure 7-14**.

Figure 7-14 New file



Add the following content in the file:

```
{

"runtimeOptions": {

"framework": {

"name": "Microsoft.NETCore.App",

"version": "2.0.0"
      }
   }
}
```

NOTE

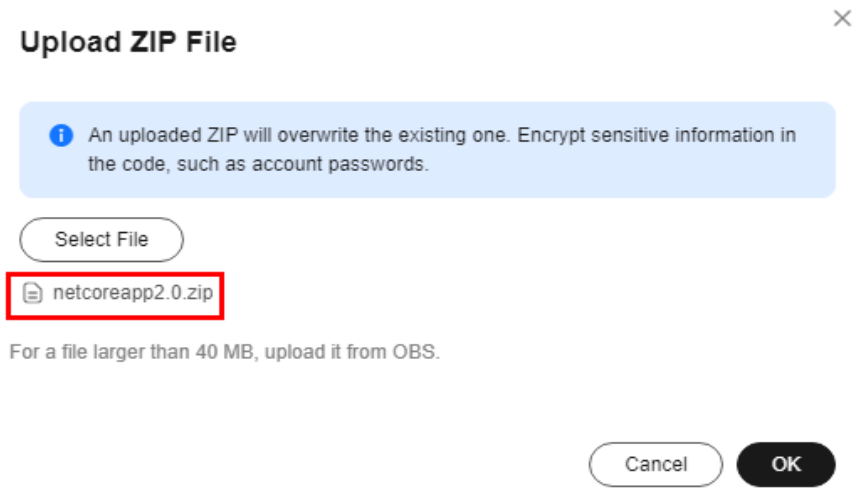
- The name of the ***.runtimeconfig.json** file is the name of an assembly.
- The **Version** parameter in the file indicates the version number of the target framework. If the framework is .NET Core 2.0, enter **2.0.0**. If the framework is .NET Core 2.1, enter **2.1.0**.

4. Compress the file into **netcoreapp2.0.zip**. (**The package name can be customized but must be ended with .zip.**)

Test Example

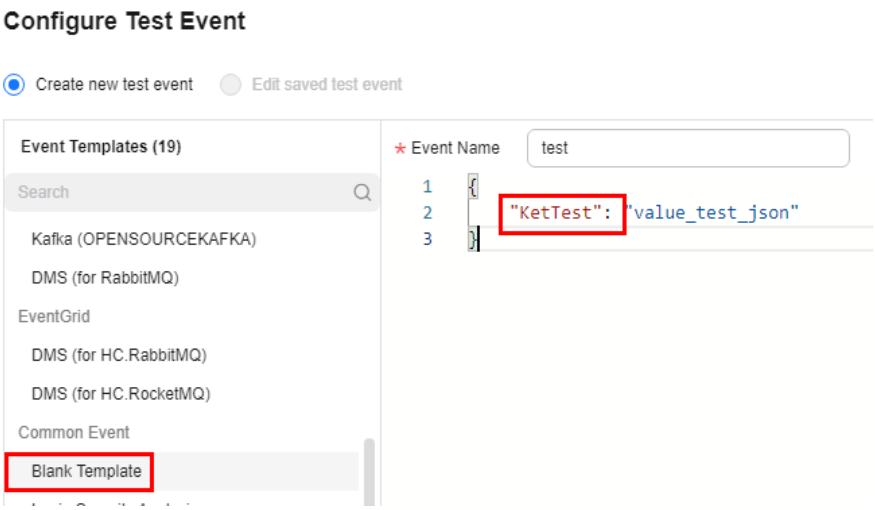
1. Create a C# (.NET 2.1) function on the FunctionGraph console and upload the code package, as shown in **Figure 7-15**.

   Figure 7-15 Uploading the code package

2. Configure a test event, as shown in **Figure 7-16**. The key must be set to **KetTest**, and the value can be customized. (**The test string must be in JSON format.**)

   Figure 7-16 Configuring a test event

   NOTE

   The attribute of the serialization class must be defined as **KetTest**.

3. Click **Test** and view the execution result.

8 PHP

8.1 Developing an Event Function

Function Syntax

Use the following syntax when creating a handler function in PHP 7.3:

function handler($event, $context)

- **$handler**: name of the function that FunctionGraph invokes to execute your code. The name must be consistent with that you define when creating a function.
- **$event**: event parameter defined for the function. The parameter is in JSON format.
- **$context**: runtime information provided for executing the function. For details, see **SDK APIs**.
- Function handler: **index.handler**.
- The function handler is in the format of *[File name].[Function name]*. For example, if you set the handler to **index.handler** in your function, FunctionGraph will load the handler function defined in the **index.php** file.

Constraints

Results returned by using the **getToken()**, **getAccessKey()**, and **getSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

PHP Initializer

FunctionGraph supports the following PHP runtime:

- Php 7.3 (runtime = Php7.3)

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.php** file.

To use PHP to build initialization logic, define a PHP function as the initializer. The following is a simple initializer:

```
<?php
Function my_initializer($context) {
   echo 'hello world' . PHP_EOL;
   }
?>
```

- Function Name

  The function name **my_initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.php** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

SDK APIs

The following table describes the context methods provided by FunctionGraph.

Table 8-1 Context methods

Method	Description
getRequestID()	Obtains a request ID.
getRemainingTimeIn-MilliSeconds ()	Obtains the remaining running time of a function.
getAccessKey()	Obtains the AK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.**
getSecretKey()	Obtains the SK (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function. **FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.**
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.

Method	Description
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getFunctionName()	Obtains the name of a function.
getRunningTimeInSeconds ()	Obtains the timeout of a function.
getVersion()	Obtains the version of a function.
getMemorySize()	Obtains the allocated memory.
getCPUNumber()	CPU usage of a function.
getPackage()	Obtains a function group, that is, an app.
getToken()	Obtains the token (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getLogger()	Obtains the **logger** method provided by the context and returns a log output class. Logs are output in the format of *Time-Request ID-Content* by using the **info** method. For example, use the **info** method to output logs: logg = context.getLogger()$ $logg->info("hello")
getAlias	Obtains function alias.

Developing a PHP Function

Constraints:

- FunctionGraph can return only the following types of values:
  - **Null**: The HTTP response body is empty.
  - **string**: The content in this string is the body of an HTTP response.
  - **Other**: FunctionGraph returns a value for JSON encoding, and uses the encoded object as the body of an HTTP response. The **Content-Type** header of the HTTP response is set to **text/plain**.
- For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

- In this example, the function project files are saved under the **~/Code/** directory. Select and package all files under the directory to ensure that the **index.php** file, the handler of your FunctionGraph function, is under the root directory when the **fss_examples_php7.3.zip** file is decompressed.

Perform the following steps to develop a PHP function:

Step 1 Create a function.

1. Write code for printing text **helloworld**.

   Open the text editor, compile a HelloWorld function, and save the code file as **helloworld.php**. The code is as follows:

   ```
   <?php
   function printhello() {
   echo 'Hello world!';
   }
   ```

2. Define a FunctionGraph function.

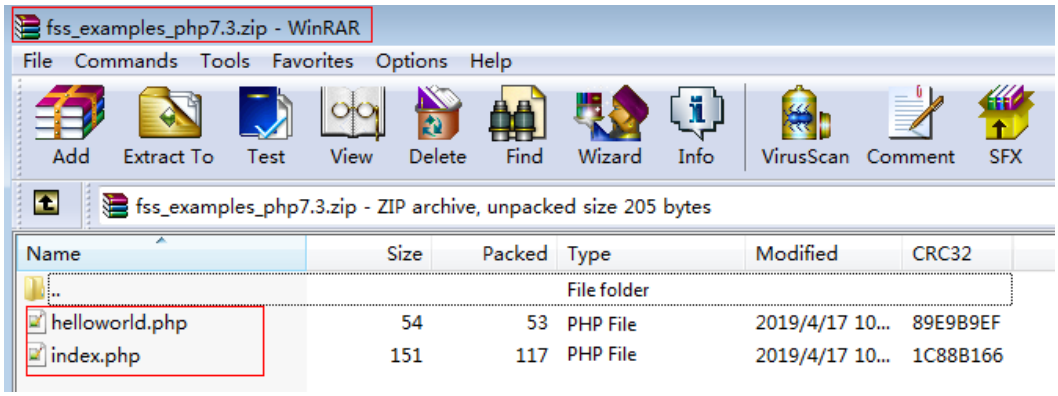
   Open the text editor, define a function, and save the function file as **index.php** under the same directory as the **helloworld.php** file. The function code is as follows:

   ```
   <?php
   include_once 'helloworld.php';

   function handler($event, $context) {
       $output = json_encode($event);
       printhello();
       return $output;
   }
   ```

Step 2 Package the project files.

After creating the function project, you get the following directory. Select all files under the directory and package them into the **fss_examples_php7.3.zip** file, as shown in **Figure 8-1**.

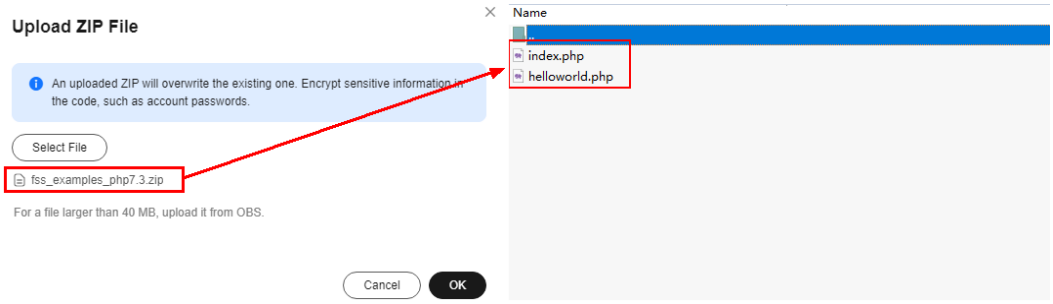
Figure 8-1 Packaging the project files



Step 3 Create a FunctionGraph function and upload the code package.

Log in to the FunctionGraph console, create a PHP function, and upload the **fss_examples_php7.3.zip** file, as shown in **Figure 8-2**.

Figure 8-2 Uploading the code package



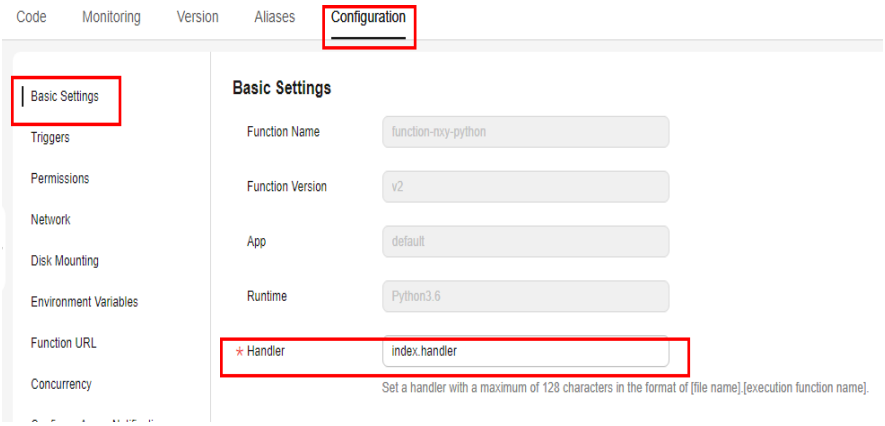
The **index** of the **handler** must be consistent with the name of the file created in **Step 1 (2)**, because the file name will help to locate the function file.

The **handler** is a function name, which must be the same as that in the **index.php** file created in **Step 1 (2)**.

After you upload the **fss_examples_php7.3.zip** file to OBS, when the function is triggered, FunctionGraph decompresses the file to locate the function file through **index** and locate the function defined in the **index.php** file through **handler**, and then executes the function.

In the navigation pane on the left of the FunctionGraph console, choose **Functions** > **Function List**. Click the name of the function to be set. On the function details page that is displayed, choose **Configuration** > **Basic Settings** and set the **Handler** parameter, as shown in **Figure 8-3**. The parameter value is in the format of **index.handler**. The values of **index** and **handler** can be customized.

Figure 8-3 Handler parameter

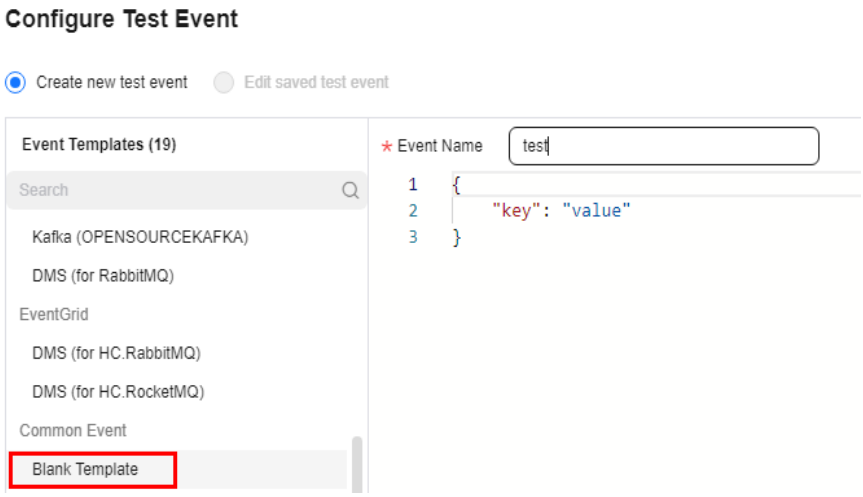


Step 4 Test the function.

1. Create a test event.

   On the function details page that is displayed, click **Configure Test Event**. Configure the test event information, as shown in **Figure 8-4**, and then click **Create**.

Figure 8-4 Configuring a test event



2. On the function details page, select the configured test event, and click **Test**.

Step 5 View the function execution result.

The function execution result consists of three parts: function output (returned by **return**), summary, and logs (output by using the **echo** method).

----End

Execution Result

The execution result consists of the function output, summary, and log output.

Table 8-2 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage**, **errorType**, and **stackTrace** is returned. The format is as follows:
		{
		"errorMessage": "",
		"errorType": "",
		"stackTrace": {}
		}
		errorMessage: Error message returned by the runtime.
		errorType: Error type.
		stackTrace: Stack error information returned by the runtime.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.

Parameter	Successful Execution	Failed Execution
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

8.2 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

Constraints

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

Creating a Dependency for a PHP Function

EulerOS 2.9.6 is recommended.

By default, Composer and PHP 7.3 have been installed in the environment. Install Protobuf 3.19 using Composer.

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

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

Run the following command:

```
Composer install
```

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

- Linux

  Run the following command to generate a ZIP package.

  ```
  zip –rq vendor.zip vendor
  ```

- Windows

  Compress **vendor** into a ZIP file.

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

NOTE

To use third-party dependencies downloaded using Composer in PHP project code, load the dependencies through **require "./vendor/autoload.php"**. By default, files decompressed from the uploaded ZIP package are placed in a directory at the same level as the project code.

9 Development Tools

9.1 FunctionGraph and IaC

Combination of FunctionGraph and IaC

Most FunctionGraph functions do not run independently, but work with storage, gateways, databases, and message queues to form serviceless applications. Infrastructure as code (IaC) automates the frequent deployments and updates of functions and triggers. This approach shortens the development cycle, simplifies configuration management, and maintains the consistency of resource deployment.

Suitable IaC Tools for FunctionGraph

Resource Formation Service (RFS) is a new final-state cloud resource orchestration engine that fully supports Terraform (HCL and Provider), the industry's de facto standard for infrastructure as code. It automatically builds cloud resources in batches based on open ecosystem templates that use the HashiCorp Configuration Language (HCL) syntax. With RFS, you can create, manage, and upgrade cloud resources (such as FunctionGraph functions, APIG gateways, and database instances) efficiently, securely, and consistently.

Getting Started with RFS for FunctionGraph

Step 1 Create the Python script file **index.py** with the following content:

```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
```

Step 2 Compress **index.py** into **index.zip**, **upload the ZIP file to an OBS bucket**, and obtain the file's **object link** in the bucket.

Step 3 Create the **variables.tf** file using the following content to define the parameters to be used in your RFS template.

```
variable "enterprise_project_id" {
  type        = string
  description = " Specifies enterprise_project_id"
  default     = "0"
}
variable "agency_name" {
  type        = string
  description = " Specifies the agency to which the function belongs."
  default     = ""
}
variable "region" {
  type        = string
  description = " Specifies the region."
  default     = "cn-north-7"
}
variable "code_url" {
  type        = string
  description = "code_url"
}
variable "apig_id" {
  type        = string
  description = "apig_id"
  default     = ""
}
```

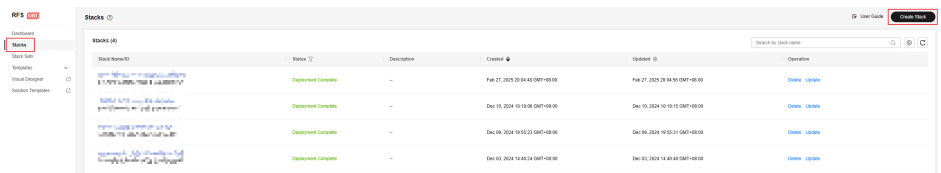
Step 4 Create the RFS template file **main.tf** using the following content to define functions and triggers.

```
variable "enterprise_project_id" {
  type        = string
  description = " Specifies enterprise_project_id"
  default     = "0"
}
variable "agency_name" {
  type        = string
  description = " Specifies the agency to which the function belongs."
  default     = ""
}
variable "region" {
  type        = string
  description = " Specifies the region."
  default     = "cn-north-7"
}
variable "code_url" {
  type        = string
  description = "code_url"
}
variable "apig_id" {
  type        = string
  description = "apig_id"
  default     = ""
}
```

Step 5 Compress **main.tf** and **variables.tf** into the **components.zip** file.

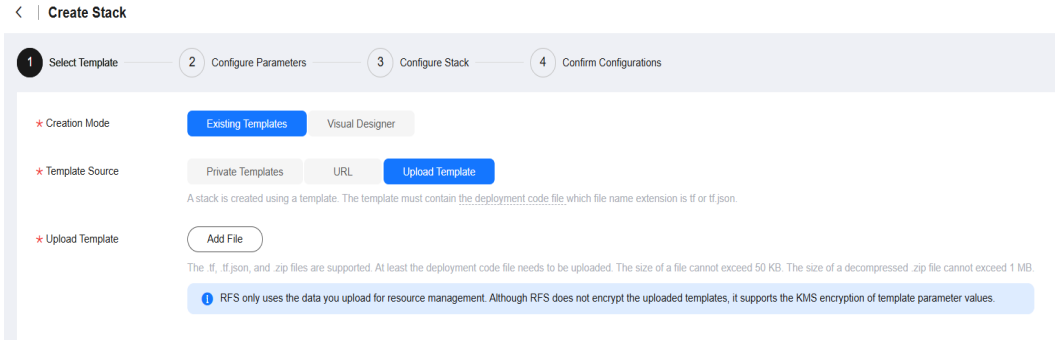
Step 6 Log in to the RFS console, and choose **Stacks** > **Create Stack** as shown in **Figure 9-1**. For details, see **Creating a Stack**.

Figure 9-1 RFS console



Step 7 Set parameters as shown in **Figure 9-2**, and upload **components.zip**. Then click **Next**.

Figure 9-2 Setting parameters



Step 8 On the **Configure Parameters** page, set the following parameters, and click **Next**.

- **enterprise_project_id**: enterprise project ID
- **agency_name**: function agency name
- **region**: region ID. Obtain it from **Regions and Endpoints**.
- **code_url**: OBS object link obtained in **step 2**
- **apig_id**: APIG gateway ID

Step 9 On the **Configure Stack** page, specify an IAM agency, retain the default values for other parameters, and click **Next** to confirm the deployment of this stack. The specified function and resource are automatically created on the FunctionGraph console.

Step 10 Log in to the **FunctionGraph console**, and click the function name in the function list to go to the details page. Choose **Configuration** > **Triggers**, locate the APIG trigger, copy its calling URL, and paste the URL into your browser or run the following **curl** command.

```
curl -s <Trigger_Invoke_URL>  # Replace <Trigger_Invoke_URL> with the calling URL of your APIG trigger.
```

The response is a JSON representation of the selected attributes in the original event object. It contains information about the request sent to the APIG endpoint. The following is an example response:

```
HTTP/1.1 200 OK
Content-Length: 658
Connection: keep-alive
Content-Type: application/json
Date: Wed, 12 Mar 2025 08:59:18 GMT
Server: api-gateway
Strict-Transport-Security: max-age=31536000; includeSubdomains;
X-Apig-Latency: 52
X-Apig-Ratelimit-Api: remain:97,limit:100,time:1 minute
X-Apig-Ratelimit-Api: remain:29973,limit:30000,time:1 second
X-Apig-Ratelimit-Api-Allenv: remain:199,limit:200,time:1 second
X-Apig-Ratelimit-Api-Allenv: remain:29973,limit:30000,time:1 second
X-Apig-Ratelimit-User: remain:3995,limit:4000,time:1 second
X-Apig-Upstream-Latency: 14
X-Cff-Billing-Duration: 1
X-Cff-Invoke-Summary:
{"funcDigest":"e6e9c99b8f5b9d6f9408d5210263330","duration":0.756,"billingDuration":1,"memorySize":128,
"memoryUsed":33.207,"podName":"pool22-300-128-fusion-844bdc7755-
bh55w","gpuMemorySize":0,"ephemeralStorage":512}
X-Cff-Request-Id: b43781ee-49f3-4762-8c24-236c718d3391
```



```
X-Content-Type-Options: nosniff
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Func-Err-Code: 0
X-Is-Func-Err: false
X-Request-Id: 90a48d7a4c699780579f4edc8983cdaf
X-Xss-Protection: 1; mode=block;

{"requestContext": {"requestId": "90a48d7a4c699780579f4edc8983cdaf", "apiId":
"01127600bb9f4d2ca8e532d1c378d8c8", "stage": ":DEBUG"}, "queryStringParameters": {}, "path": "/nxy-
sasa", "httpMethod": "GET", "isBase64Encoded": true, "headers": {"host":
"47f32d1efa1742f5a7ee5b720ca9c4a5.apig.cn-east-3.huaweicloudapis.com", "content-type": "application/
json", "x-forwarded-host": "47f32d1efa1742f5a7ee5b720ca9c4a5.apig.cn-east-3.huaweicloudapis.com",
"user-agent": "APIGatewayDebugClient/1.0", "x-forwarded-port": "443", "x-forwarded-proto": "https", "x-
request-id": "90a48d7a4c699780579f4edc8983cdaf", "x-apig-mode": "debug"}, "body": "",
"pathParameters": {}}
```

----End

9.2 Local Debugging with VSCode

Introduction

Huawei Cloud FunctionGraph is a Visual Studio Code (VSCode) plug-in of Huawei Cloud serverless products. With this plug-in, you can:

- Quickly create local functions.
- Run and debug local functions and deploy them to the cloud.
- Pull the function list from the cloud, call cloud functions, and upload ZIP packages to the cloud.

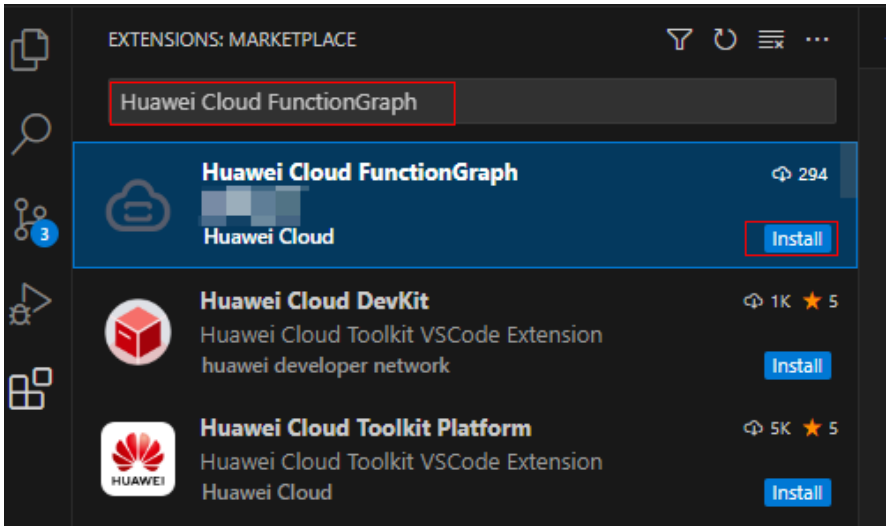
Prerequisites

You have downloaded the **VSCode tool (later than 1.63.0)** and installed it.

Installing the Plug-in

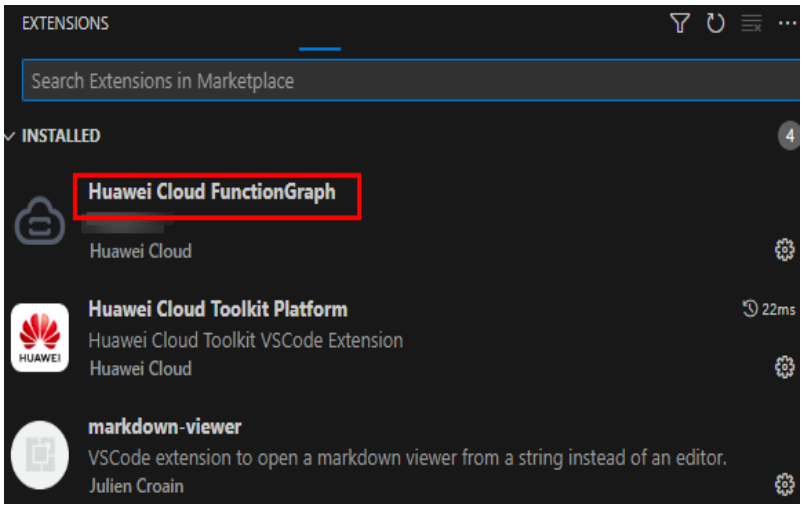
1. Open VSCode, search for **Huawei Cloud FunctionGraph** in the app store, and install it.

   Figure 9-3 Searching for and installing Huawei Cloud FunctionGraph



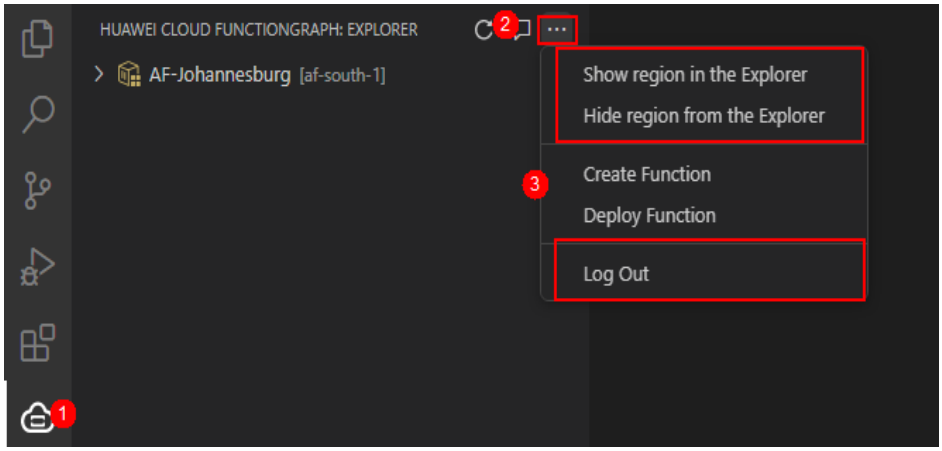
2. After the installation is successful, **Huawei Cloud FunctionGraph** is displayed in the plug-in list.

Figure 9-4 Installed plug-ins



Logging In to FunctionGraph Plug-in

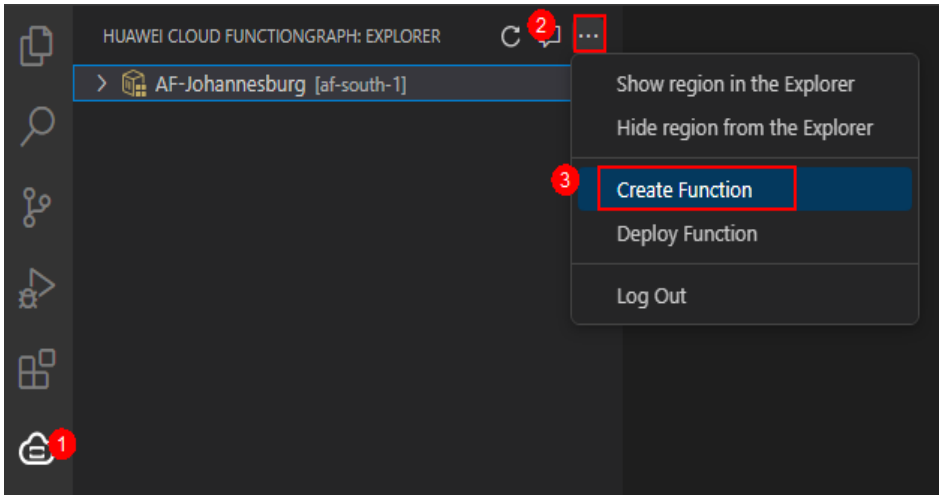
1. Click the **Huawei Cloud FunctionGraph** plug-in icon, click the login link, and select a login mode. If you select login with AK/SK, obtain an AK/SK. For details, see **Creating an Access Key**.
2. Select a region to view function information.
3. Show or hide desired regions, or log out of your account by referring to the following figure.
   - **Show region in the Explorer**: Show the regions where you need to perform operations.
   - **Hide region from the Explorer**: Hide the regions you do not need.
   - **Log Out**: Log out of your account.



Creating a Function

1. On the plug-in panel, select **Create Function** or press **Ctrl+Shift+p** to search for the **Create Function** command. Then, specify the runtime, template,

function name, and local folder as prompted. A function is created in the specified folder.



2. After the local function is created, the handler file is automatically opened.
3. You can customize the function's configuration by modifying the automatically generated configuration file. The parameters are as follows:

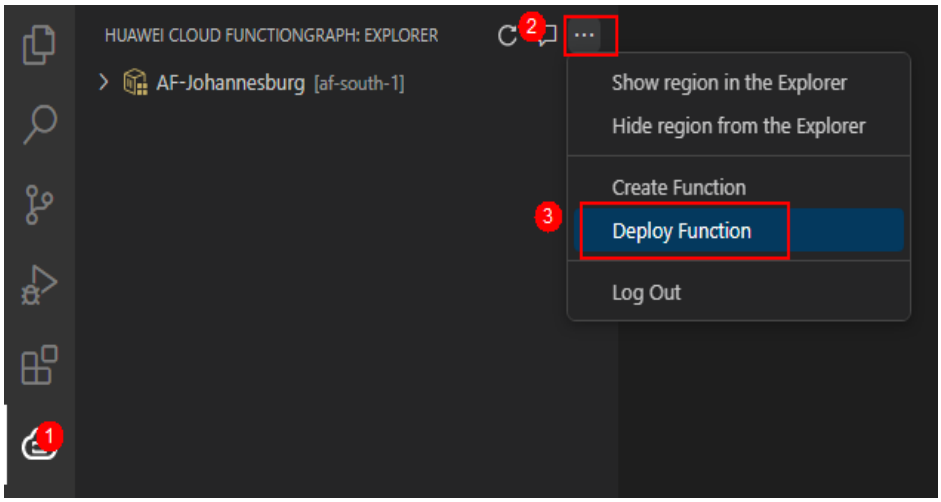
```
HcCrmTemplateVersion: v2
Resources:
  Type: HC::Serverless::Function
  Properties:
    Name: functionName //Function name
    Handler: handler // Handler of the function
    Runtime: runtime // Function runtime
    CodeType: inline // Default
    CodeFileName: index.zip // Default
    CodeUrl: ""
    Description: '' // Function runtime
    MemorySize: 128 // Memory for executing the function
    Timeout: 30 // Function timeout, in seconds.
    Version: latest // Default
    Environment:
      Variables: {} // Environment variables
    InitializerHandler: "" // Function initializer
    InitializerTimeout: 0 // Initialization timeout of the function
    EnterpriseProjectId: "0" // Enterprise project
    FuncType: v2
    URN: "" // Function URN, which is generated after the function is downloaded.
```

Deploying a Function

- **Prerequisites**

  Ensure that the function code path is correct. The code of Node.js, Python, and PHP functions is stored in the **src** directory, and the code of other functions is stored in the root directory.

On the plug-in panel, select **Deploy Function** or press **Ctrl+Shift+p** to search for the **Deploy Function** command, and select a function and region as prompted.



- If the deployment is successful, a success message is displayed in the lower right corner of the page. Switch to the target region to view the deployment result.
- If the deployment fails, view the error log in the **Output** area and rectify the fault.

Local Debugging

1. Node.js
   - **Prerequisites**

     Node.js has been installed in the local environment.
   - **Default mode**

     Click **Local Debug** of the handler method, configure the event content, and click **Invoke** for debugging.

     Figure 9-5 Clicking Local Debug

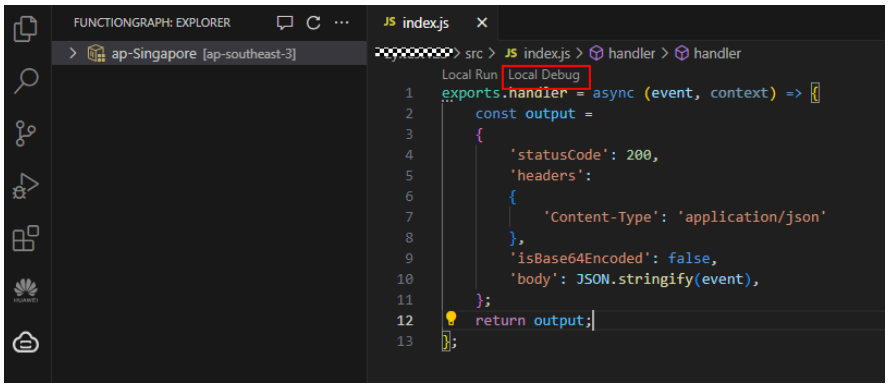
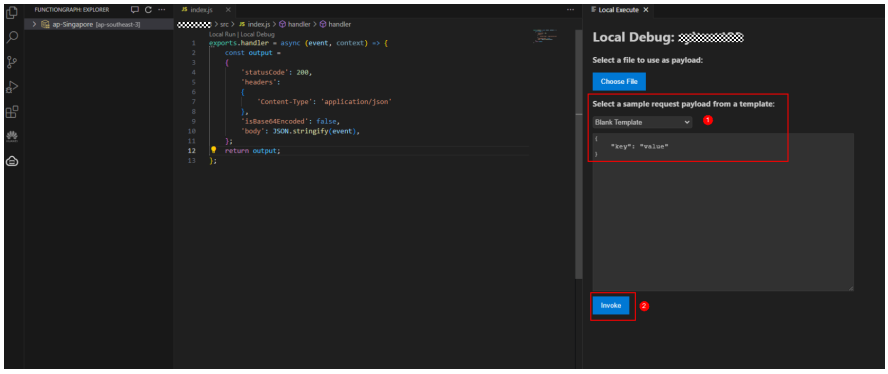


Figure 9-6 Configuring the event content



– **Debugging with VS Code**

Create the **main.js** file in the function folder and copy the following content to this file. Click the  icon on the left. Then, click **Add Config**, select Node.js, and press **F5** to start debugging.

```
const handler = require('./index'); //Path of the function handler file. Modify it as required.
const event = { 'hello': 'world' }; //Test event. Modify it as required.
const context = {}; //Context class.
console.log(handler.handler(event, context));
```

2. Python

– **Prerequisites**

Python has been installed in the local environment.

Create the **main.py** file in the function folder and copy the following content to this file. Click the  icon on the left. Then, click **Add Config**, select Python, and press **F5** to start debugging.

```
import sys
import index #Path of the function handler file. Modify it as required.

#The main method is used for debugging, and event is the selected debugging event.
if __name__ == '__main__':
....event = { 'hello': 'world' } #Test event. Modify it as required.
    context = ''
    content = index.handler(event, context)
....print('Returned value:')
    print(content)
```

3. Java

– **Prerequisites**

Java has been installed. VS Code supports Java testing.

In the **test** directory of the function folder, open the **TriggerTestsTest.java** file, and click the  icon on the left. Then, click **Add Config**, select Java, and press **F5** to start debugging.

Other Functions

- **Opening in Portal**

  Right-click the target function, and choose **Open in Portal**. The function details page is displayed.

- **Executing a Cloud Function**
  a. Right-click the target function and choose **Invoke Function...** from the shortcut menu.
  b. In the **Invoke Function** panel, select the event to be passed and click **Invoke**. The function log and result are displayed in the **Output** area.
- **Downloading a Cloud Function**
  – **Prerequisites**
    You have been granted the permission (**obs:object:GetObject**) for obtaining bucket objects.

  Right-click the function you want to download and choose **Download...** from the shortcut menu. The function code is downloaded from the cloud to the specified local path, and the handler file is automatically opened.
- **Updating a Cloud Function**
  Right-click the target function, choose **Upload Function...** from the shortcut menu, and select a ZIP package to upload.
- **Deleting a Cloud Function**
  a. Right-click the function to be deleted and choose **Delete...** from the shortcut menu.
  b. In the confirmation dialog box, click **Delete** to delete the function.
- **Copying URN**
  Right-click the target function and choose **Copy URN** from the shortcut menu.

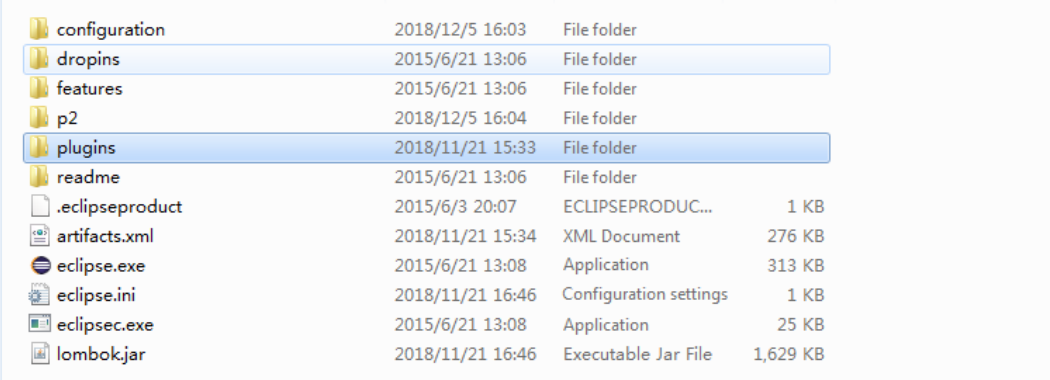
9.3 Eclipse Plug-in

Currently, FunctionGraph does not provide Java function templates and only allows you to upload Java function packages through OBS. With the Eclipse plug-in, you can quickly create Java function templates, package function project files, upload function packages to OBS, and deploy functions.

Step 1 Obtain the **Eclipse plug-in** (software package verification file: **Eclipse plug-in.sha256**).

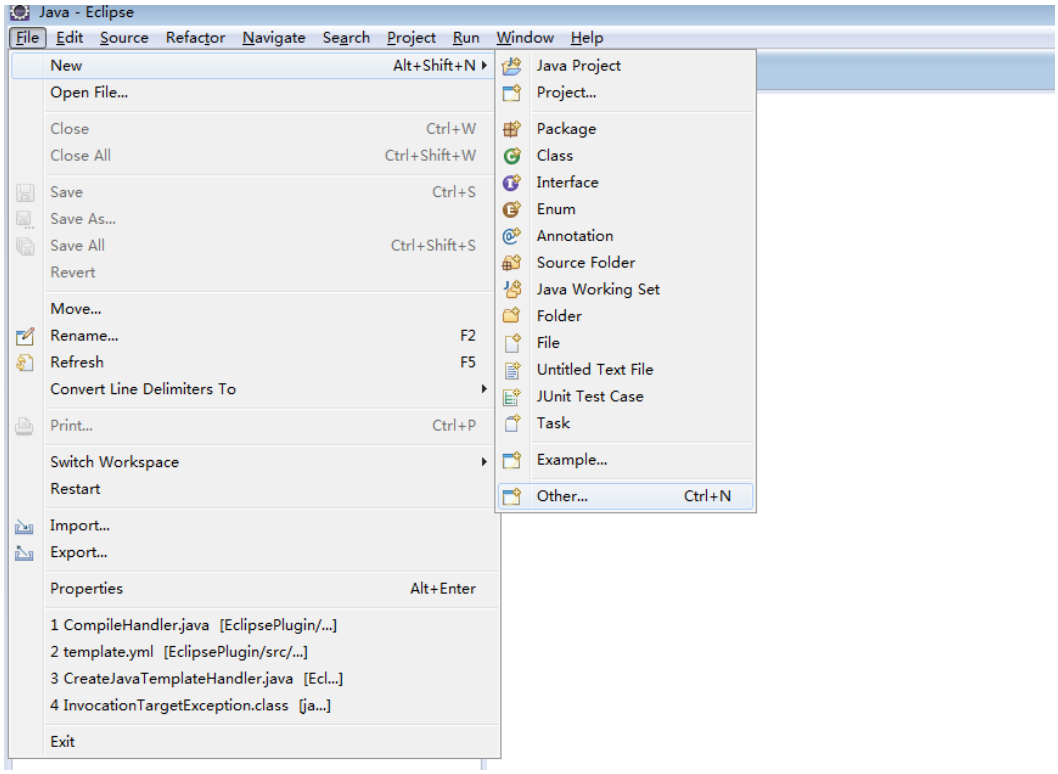
Step 2 Put the Eclipse plug-in package (.jar or .zip) in the **plugins** folder under the Eclipse installation directory. Then restart Eclipse. **Figure 9-7** shows the Eclipse installation directory.

Figure 9-7 Installing the Eclipse plug-in



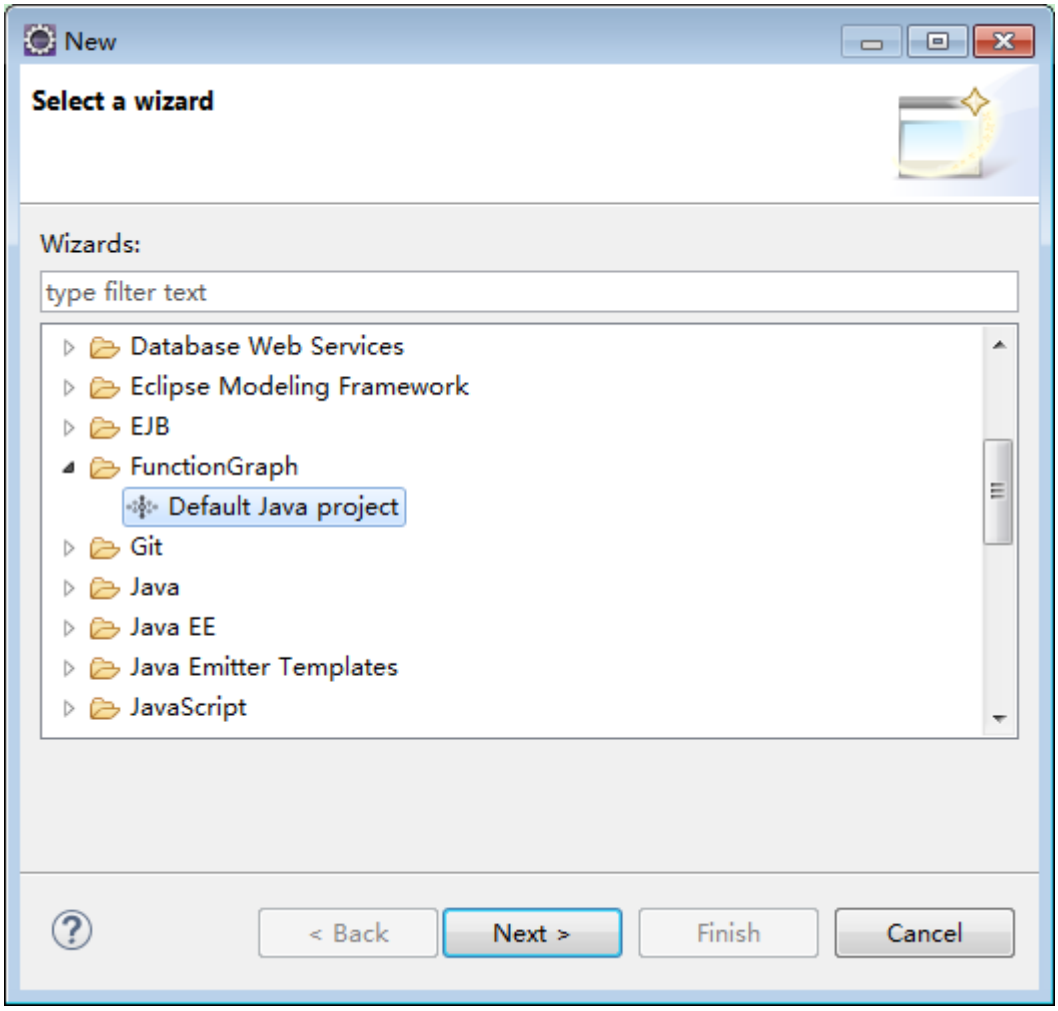
Step 3 Open Eclipse and choose **File** > **New** > **Other**, as shown in **Figure 9-8**.

Figure 9-8 Creating a template



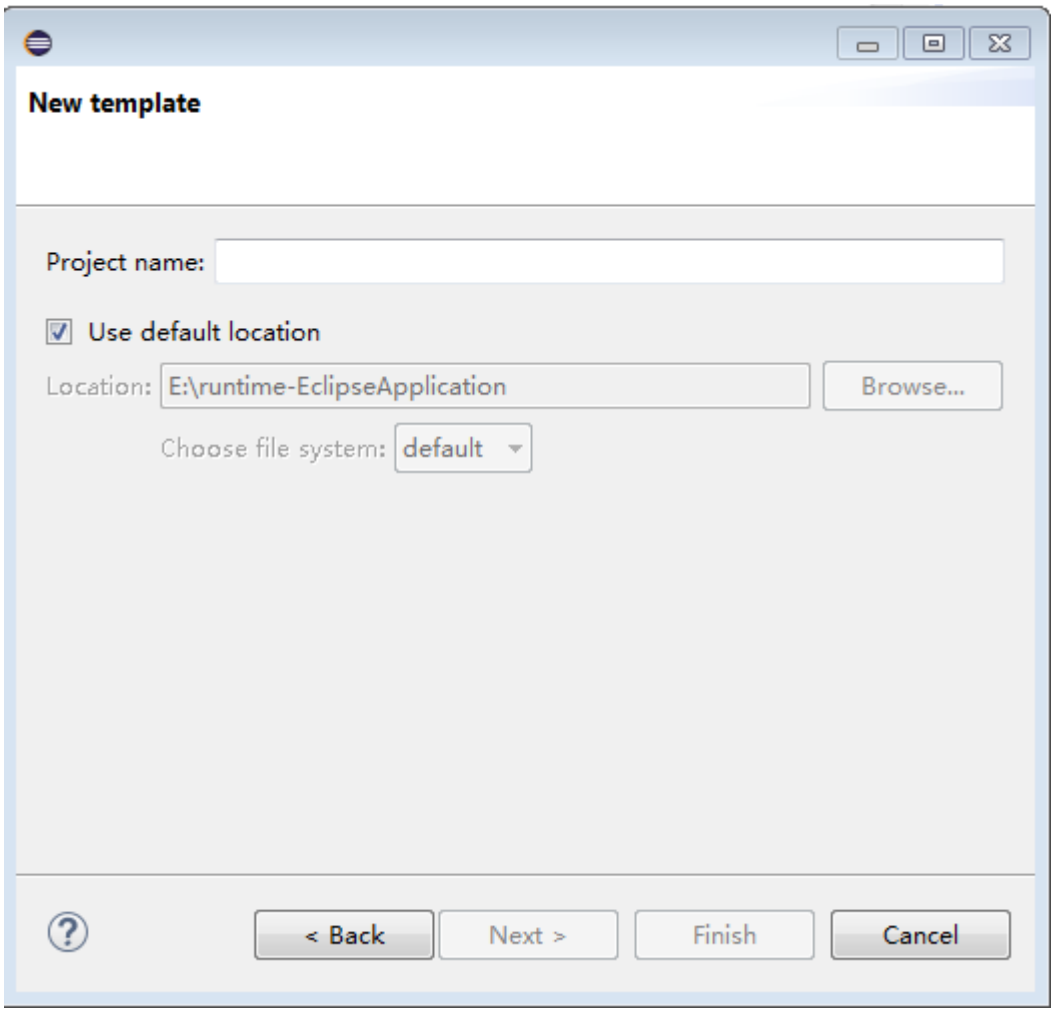
Step 4 Choose **FunctionGraph** > **Default Java project**, as shown in **Figure 9-9**.

Figure 9-9 Selecting the default Java template



Step 5 Enter a project name, specify a project directory (or use the default directory), and click **Finish**, as shown in **Figure 9-10**.

Figure 9-10 Setting the project name and directory



----End

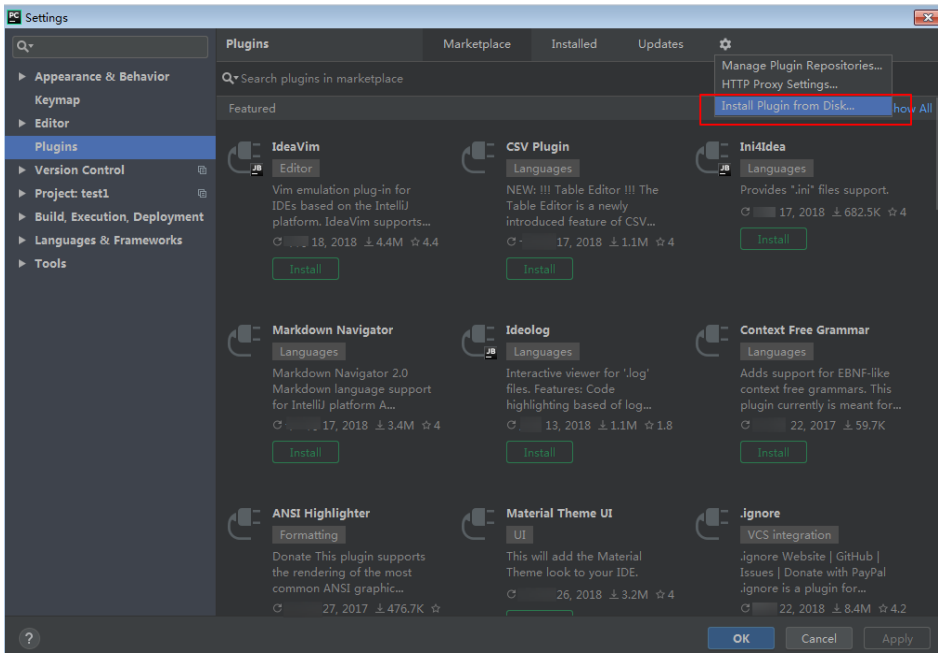
9.4 PyCharm Plug-in

With PyCharm, you can quickly generate Python templates, package project files, and deploy Python functions.

Step 1 Obtain the **plug-in** (**Plug-in.sha256**) .

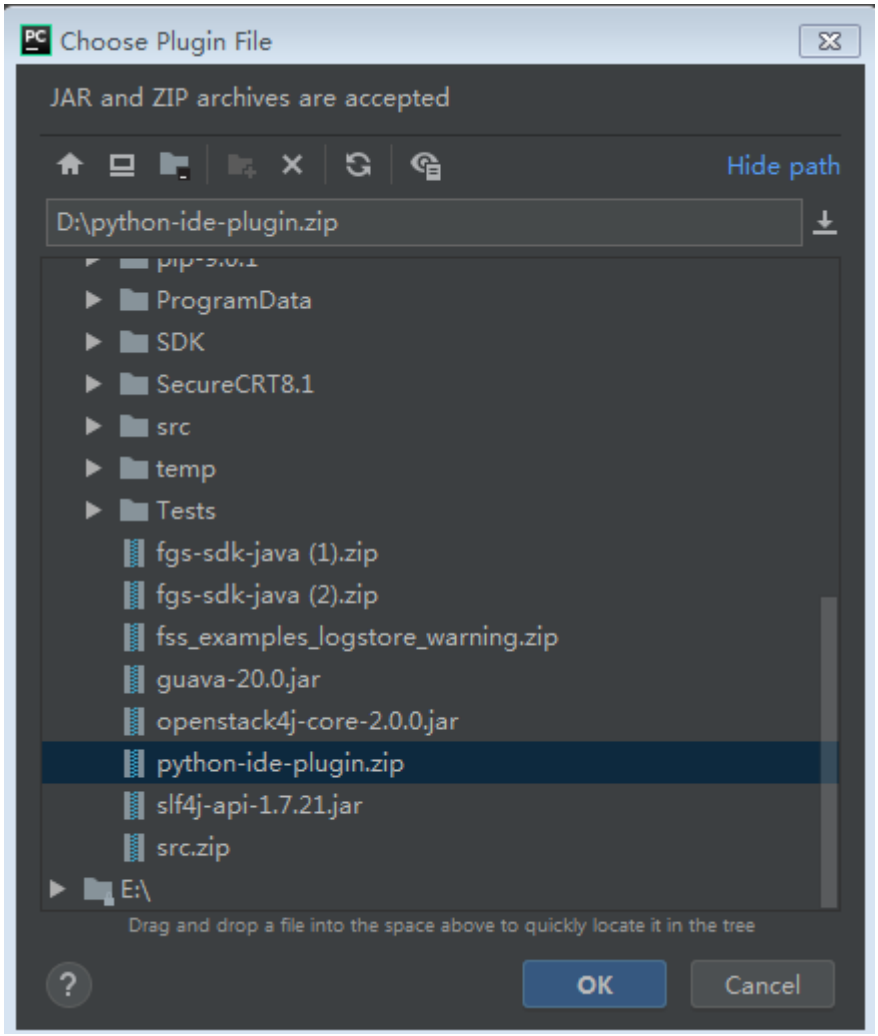
Step 2 Run JetBrains PyCharm. Choose **File** > **Settings**, choose **Plugins** in the left pane, and then click **Install Plugin from Disk** in the upper right corner, as shown in **Figure 9-11**.

Figure 9-11 Installing the plug-in



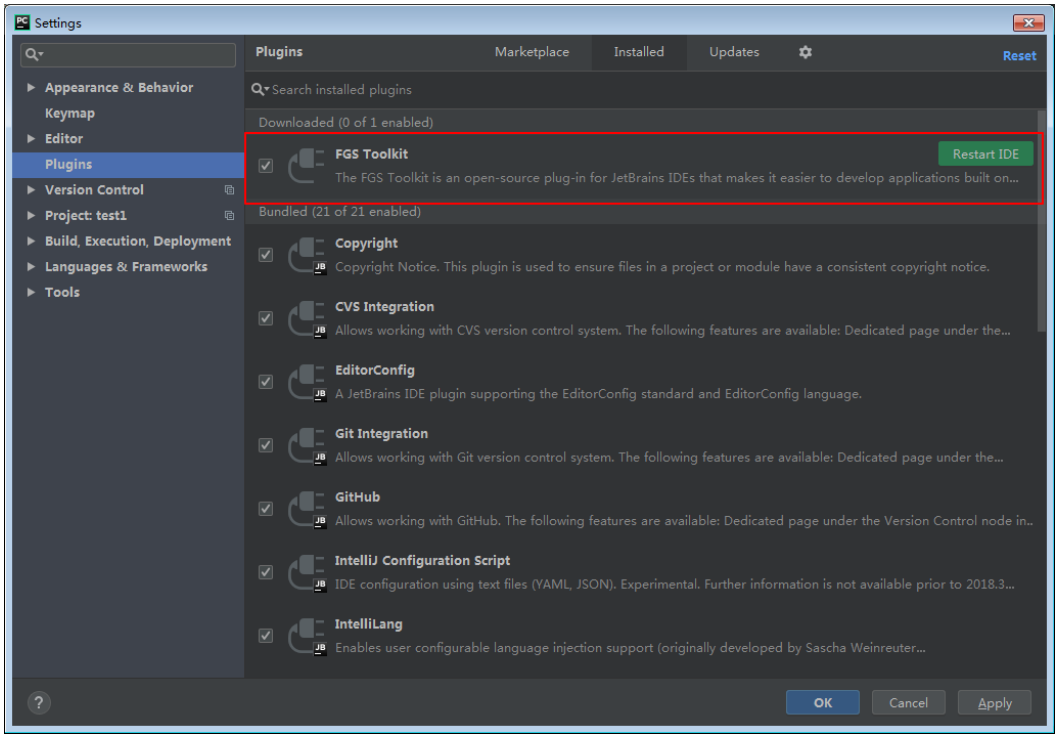
Step 3 Select the plug-in package you want to install, and click **OK**, as shown in **Figure 9-12**.

Figure 9-12 Selecting a plug-in package



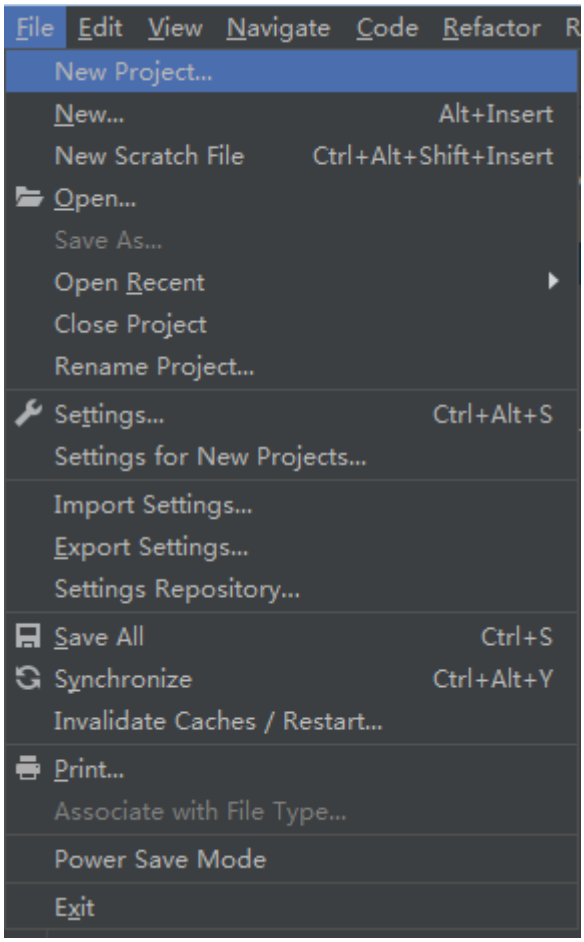
Step 4 In the plug-in list, select the desired plug-in and click **Restart IDE**, as shown in **Figure 9-13**.

Figure 9-13 Restarting the IDE



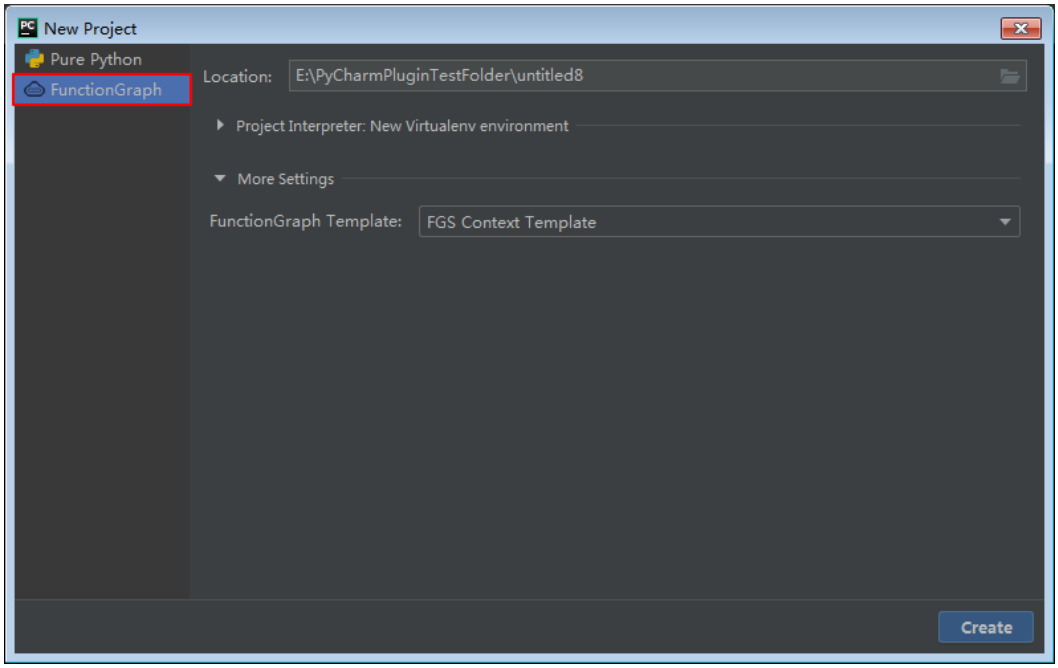
Step 5 Choose **File** > **New Project**, as shown in **Figure 9-14**.

Figure 9-14 Creating a project



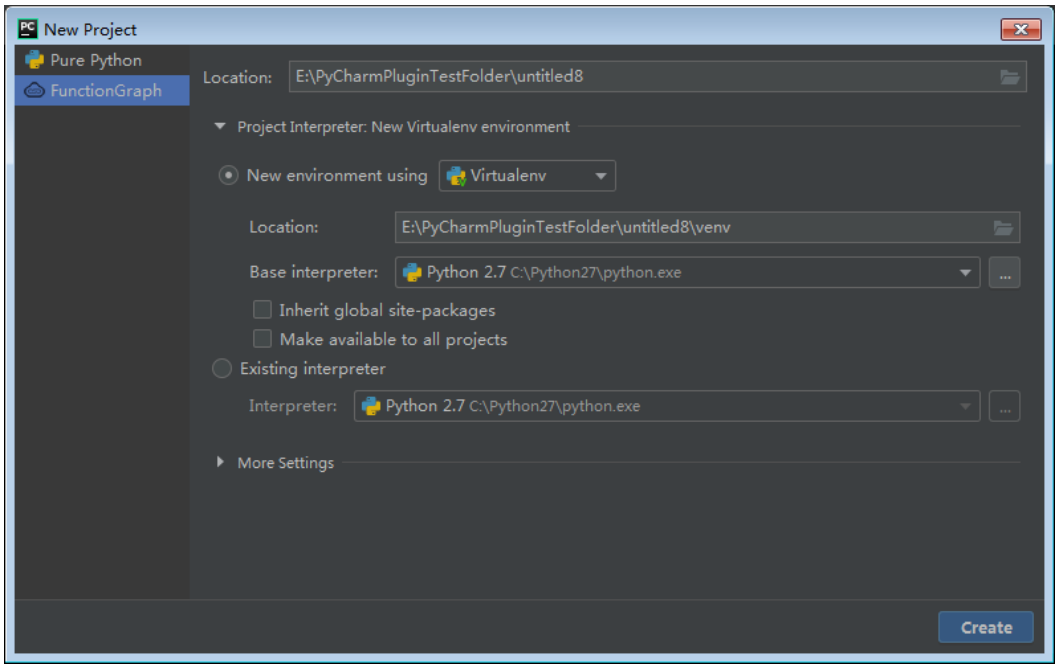
Step 6 On the displayed **New Project** page, choose **FunctionGraph**, as shown in **Figure 9-15**.

Figure 9-15 FunctionGraph



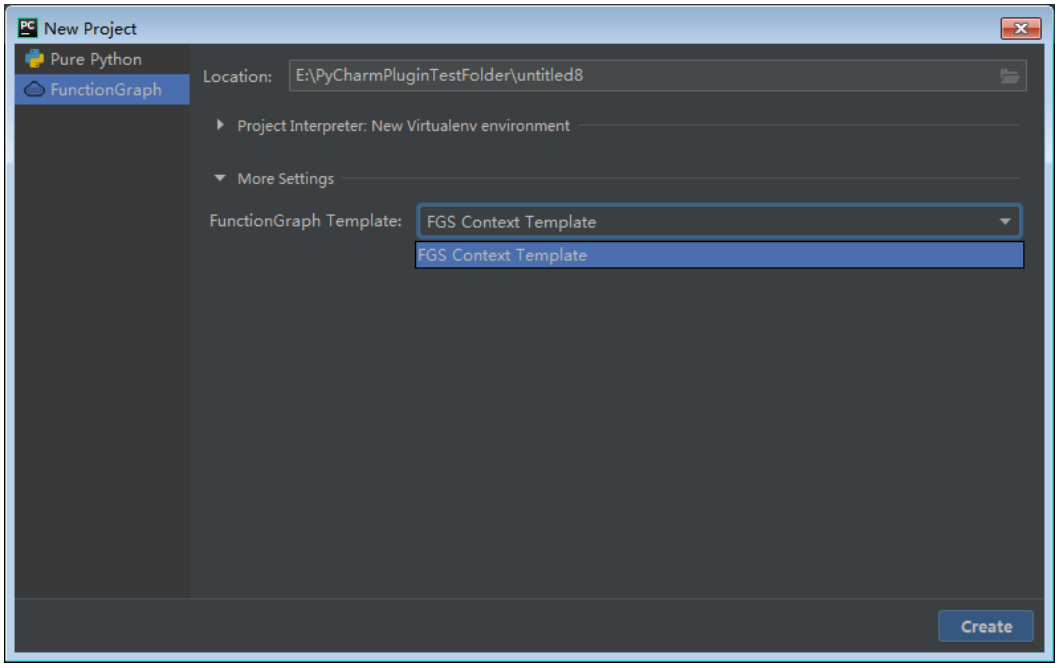
Step 7 Select the path in which the project will be stored in **Location**, and select a Python version in **Base interpreter**, as shown in **Figure 9-16**.

Figure 9-16 Selecting a version



Step 8 Select a template you want to create in the **More Settings** area, as shown in **Figure 9-17**.

Figure 9-17 Selecting a template



NOTE

Currently, only the Python 2.7 context template is supported.

Step 9 Click **Create**.

----End

9.5 Serverless Devs

9.5.1 Introduction

Component Description

Huawei Cloud FunctionGraph component is a tool used to manage the lifecycle of Huawei Cloud function applications. It is developed based on **Serverless Devs**. By configuring the resource configuration file **s.yaml**, you can easily and quickly deploy applications on **Huawei Cloud FunctionGraph**.

Prerequisites

Node.js has been installed on the local host.

Getting Started

Step 1 Run the **npm install -g @serverless-devs/s** command to install the Serverless Devs developer tool.

After the installation is complete, you need to configure the key. For details, see **Key Configuration**.

Step 2 Initialize a Hello World project by running **s init start-fg-http-nodejs14**.

Step 3 After the initialization is complete, enter the project and run **s deploy** to deploy a function.

----End

Using Commands

The capabilities of the FunctionGraph component are listed in **Table 9-1**.

Table 9-1 Component capabilities

Build & Deploy	Publish & Configure	Other Functions
deploy	version	Project Migration fun2s
remove	alias	-

When using the FunctionGraph component, you also need to compile resource description files. For details about YAML specifications of the FunctionGraph component, see **Huawei Cloud FunctionGraph YAML Specifications**.

9.5.2 Key Configuration

Obtaining Key Information

1. Log in to Huawei Cloud and choose **My Account** > **My Credentials** in the upper right corner. The **My Credentials** page is displayed.
2. In the navigation pane on the left, choose **Access Keys**. Click **Create Access Key** to generate a new access key and download and save it.

Configuring the AK/SK

Using Wizard

Run **config add** to add a key:

```
$ s config add

? Please select a provider: (Use arrow keys)
  Alibaba Cloud (alibaba)
  AWS (aws)
  Azure (azure)
  Baidu Cloud (baidu)
  Google Cloud (google)
> Huawei Cloud (huawei)
  Tencent Cloud (tencent)
  Custom (others)
```

After selecting a provider, you will be prompted to enter the corresponding information:

```
s config add

? Please select a provider: Huawei Cloud (huawei)
? AccessKeyID **********
```



```
? SecretAccessKey **********
? Please create alias for key pair. If not, please enter to skip default
```

Using a Command

Add a key using a command:

```
$ s config add --AccessKeyID ****** --SecretAccessKey ******
```

Or:

```
$ s config add -kl AccessKeyID,SecretAccessKey -il ${AccessKeyID},${SecretAccessKey}
```

9.5.3 Using Commands

9.5.3.1 deploy

9.5.3.2 version

9.5.3.3 Project Migration fun2s

fun2s is used to convert the configuration of a function into the **s.yaml** format so that it can be identified by Serverless Devs.

- **Command Parsing**
  - **Parameter Parsing**
  - **Examples**

Command Parsing

You can run **fun2s -h** or **fun2s --help** to view the documentation.

Parameter Parsing

Table 9-2 Parameter description

Parameter Name	Abbreviation	Required in CLI	Description
region	-	Yes	Region
function-name	-	Yes	Function name
target	-	No	Path of the generated Serverless Devs configuration file (**s.yaml** by default)

This command also supports some global parameters, such as **-a/--access** and **--debug**. For details, see **Global Parameters of Serverless Devs**.

Examples

In the Funcraft project, run **fun2s** to convert a function into the YAML format. For example:

```
s cli fgs fun2s --region cn-north-4 --function-name fgs-deploy-test --target ./s.yml

Tips for next step
======================
* Deploy Function: s deploy -t ./s.yml
```

In this way, you can convert the original function configuration into **s.yaml** so that it complies with Serverless Devs specifications.

After conversion (**s.yaml**):

```
edition: 1.0.0
name: transform_fun
access: default
vars:
  region: cn-north-4
  functionName: fgs-deploy-test
services:
  component-test: #  Service name
    component: fgs # Component name
    props:
      region: ${vars.region}
      function:
        functionName: ${vars.functionName}
        handler: index.handler
        memorySize: 256
        timeout: 300
        runtime: Node.js14.18
        codeType: zip
        code:
          codeUri: ./code
```

9.5.3.4 remove

9.5.3.5 alias

9.5.3.6 YAML File

Complete YAML Configuration

The YAML fields of Huawei Cloud FunctionGraph component are as follows:

```
edition: 1.0.0 #  YAML specifications version for the command line, which complies with the Semantic Versioning specifications.
name: fg-test #  Project name
access: "default" #  Key alias

vars: # Global variables
    region: "cn-east-3"
    functionName: "start-fg-event-nodejs14"

services:
    component-test: #  Service name
        component: fgs # Component name
        props:
            region: ${vars.region}
            function:
                functionName: ${vars.functionName} # Function name
                handler: index.handler # Handler
                memorySize: 256 # Memory required for the function
                timeout: 30 # Timeout for executing the function
```



```
    runtime: Node.js14.18 # Runtime
    agencyName: fgs-vpc-test # Agency name
    environmentVariables: # Environment variables
      test: test
      hello: world
    vpcId: xxx-xxx # Unique ID of a Virtual Private Cloud (VPC)
    subnetId: xxx-xxx # Subnet ID
    concurrency: 10 # Maximum number of instances for the function
    concurrentNum: 10 # Maximum number of concurrent requests per instance
    codeType: zip # Function code type
    dependVersionList: # Dependency ID
      - xxx-xxx
    code: # Local code address
      codeUri: ./code
  trigger:
    triggerTypeCode: TIMER # Trigger type
    status: DISABLED # Trigger status
    eventData: # Trigger configuration
      name: APIG_test  # API name
      groupName: APIGroup_xxx # Group name
      auth: IAM # Security authentication
      protocol: HTTPS # Request protocol
      timeout: 5000 # Backend timeout duration
```

Table 9-3 Parameter description

Parameter	Required	Type	Description
region	True	Enum	Region
function	True	**Struct**	Function
triggers	False	**Struct**	Triggers

Description of function Fields

The **function** fields in the YAML file are described in **Table 9-4**.

Table 9-4 Description of function fields

Parameter Name	Required	Type	Description
functionName	True	String	Function name.
handler	True	String	Handler of the function in the format of "xx.xx". It must contain a period (.).

Parameter Name	Required	Type	Description
runtime	True	String	Runtime of the function. Currently, the the following runtimes are supported: • Node.js 14.18, Node.js 12.13, Node.js 10.16, Node.js 8.10, Node.js 6, Node.js 4.4 • Python 3.9, Python 3.6, Python 2.7 • Java 11, Java 8 • Go 1.x, Go 1.8 • PHP 7.3 • HTTP • Custom
package	False	String	Package (or group) to which the function belongs. The default value is **default**.
memorySize	True	Number	Memory size (MB) allocated to the function. The options include 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2560, 3072, 3584, and 4096.
timeout	True	Number	Maximum duration the function can be executed. Value range: 3s–900s.
CodeType	True	String	Function code type. Options: • **inline**: inline code • **zip**: ZIP file • **obs**: function code stored in an OBS bucket • **jar**: JAR file, mainly for Java functions
codeUrl	False	String	If **CodeType** is set to **obs**, enter the OBS URL of the function code package. If **CodeType** is not set to **obs**, leave this parameter blank.
environmentVariables	False	**Struct**	Environment variable. A maximum of 20 environment variables can be defined. The total length cannot exceed 4 KB.
agencyName	False	String	Name of an agency created in IAM. This is required when the function needs to access other services.
vpcId	False	String	Unique ID of a VPC. **agencyName** is required if you set this parameter. Log in to the **VPC Console** to view the VPC ID.
subnetId	False	String	Subnet ID. **agencyName** is required if you set this parameter. To obtain the subnet ID, log in to the **VPC Subnet page**.

Parameter Name	Required	Type	Description
dependVersionList	False	List<String>	Dependency package IDs.
code	False	Struct	Local code address, which is required if **CodeType** is set to **zip**.
concurrency	False	Number	Maximum number of instances in which a function can run. Range: –1 to 1,000. **–1**: The function can run in any number of instances. **0**: The function is disabled.
concurrentNum	False	Number	Maximum number of concurrent requests per instance. Range: –1 to 1,000.
description	False	String	Brief description of **function**.

- Func Code parameters

  Table 9-5 Func Code parameters

Parameter Name	Required	Type	Description
codeUri	True	String	Local code address

- Environment variables

  Object format. For example:

  ```
  DB_connection: jdbc:mysql://ip:port/dbname
  ```

  Do not write sensitive information to **s.yaml** in plaintext.

Example

```
function:
  functionName: event-function
  description: this is a test
  runtime: Node.js14.18
  handler: index.handler
  memorySize: 128
  timeout: 60
  code:
    codeUri: ./code
  environmentVariables:
    test: 123
    hello: world
```

Description of triggers Fields

The **triggers** fields in the YAML file are described in **Table 9-6**.

Table 9-6 trigger parameter description

Parameter Name	Required	Type	Description
triggerTypeCode	True	String	Trigger type
status	False	Enum	Trigger status. Options: **ACTIVE** (default) and **DISABLED**
eventData	True	Struct	Trigger configurations, including **APIG** and **timer** triggers

- APIG trigger

  Table 9-7 APIG trigger parameters

Parameter Name	Required	Type	Description
name	False	String	API name. The function name is used by default.
groupName	False	String	Group. The first one is selected by default.
auth	False	Enum	Authentication mode. Default: **IAM**. API authentication mode. Options: • **App**: AppKey and AppSecret high security authentication. This authentication mode is recommended. For details, see **App Authentication**. • **IAM**: IAM authentication. Only IAM users are allowed to access the system. The security level is medium. For details, see **IAM Authentication**. • **None**: No authentication. This mode grants access permissions to all users.
protocol	False	Enum	Request protocol. Default: **HTTPS**. Options: • HTTP • HTTPS
timeout	False	Number	Backend timeout in milliseconds. Range: 1–60,000. Default: **5000**.

  Example:

```
trigger:
  triggerTypeCode: APIG
```



```
  status: ACTIVE
  eventData:
    name: APIG_test
    groupName: APIGroup_xxx
    auth: IAM
    protocol: HTTPS
    timeout: 5000
```

- Timer trigger

Table 9-8 Timer trigger parameter description

Parameter Name	Required	Type	Description
name	False	String	Timer name.
scheduleType	True	Enum	Trigger rule. The value can be **Rate** or **Cron**.
schedule	True	String	Timer rule content.
userEvent	False	String	Additional information, which will be put into the **user_event** field of the timer event source.

Example:

```
trigger:
  triggerTypeCode: TIMER
  status: ACTIVE
  eventData:
    name: Timer-xxx
    scheduleType: Rate
    schedule: 3m
    userEvent: xxxx

trigger:
  triggerTypeCode: TIMER
  status: ACTIVE
  eventData:
    name: Timer-xxx
    scheduleType: Cron
    schedule: 0 15 2 * * ?
    userEvent: xxxx
```

9.5.4 Huawei Cloud FunctionGraph YAML Specifications

Field Parsing

Table 9-9 Parameter description

Parameter Name	Required	Type	Description
region	True	Enum	Enum
function	True	Struct	Function

Parameter Name	Required	Type	Description
trigger	False	Struct	Triggers

Complete YAML Configuration

The YAML fields of Huawei Cloud FunctionGraph component are as follows:

```
edition: 1.0.0 #  YAML specifications version for the command line, which complies with the Semantic Versioning specifications.
name: fg-test # Project name
access: "default" # Key alias

vars: # Global variable
  region: "cn-east-3"
  functionName: "start-fg-event-nodejs14"

services:
  component-test: # Service name
  component: fgs # Component name
    props:
      region: ${vars.region}
      function:
        functionName: ${vars.functionName} # Function name
        handler: index.handler # Handler of the function
        memorySize: 256 # Memory required for the function
        timeout: 30 # Timeout for executing the function
        runtime: Node.js14.18 # Runtime
        agencyName: fgs-vpc-test # Agency name
        environmentVariables: # Environment variables
          test: test
          hello: world
        vpcId: xxx-xxx # Unique ID of a Virtual Private Cloud (VPC)
        subnetId: xxx-xxx #  Subnet ID
        concurrency: 10 # Maximum number of instances of a single function
        concurrentNum: 10 # Maximum number of concurrent tasks of a single instance
        codeType: zip # Code type of the function
        dependVersionList: # Dependency ID
          - xxx-xxx
        code: # Local code address
          codeUri: ./code
      trigger:
        triggerTypeCode: TIMER # Trigger type
        status: DISABLED # Trigger status
        eventData: # Trigger configuration
          name: APIG_test  # API name
          groupName: APIGroup_xxx # Group name
          auth: IAM # Security authentication
          protocol: HTTPS # Request protocol
          timeout: 5000 # Backend timeout duration
```


9.5.5 Global Parameters of Serverless Devs

Table 9-10 Global parameters of Serverless Devs

Parameter Name	Abbreviation	Default Value	Description	Remarks
template	t	**s.yaml** or **s.yml**	Specify a resource description file.	-
access	a	**access** information specified in the YAML file or **default**	Specify the key information for this deployment.	You can use the key information configured by running the **config** command and the key information configured to environment variables.
skip-actions	-	-	Skip the **actions** module set in the YAML file.	-
debug	-	-	Enable the Debug Mode	After the Debug Mode is enabled, you can view more information about the tool execution process.
output	o	**default**	Specify the data output format.	The **default**, JSON, YAML, and RAW formats are supported.
version	v	-	View version information	-
help	h	-	View help information	-

9.6 Serverless Framework

9.6.1 User Guide

Welcome to Serverless Huawei Cloud FunctionGraph User Guide.

NOTE

Before using the CLI, you need to provide **Huawei Cloud Credentials**.

9.6.1.1 Introduction

The Serverless Framework helps you develop and deploy serverless applications using FunctionGraph. It is a CLI that offers structure, automation, and best practices out-of-the-box, allowing you to focus on building sophisticated, event-driven, serverless architectures, comprised of **Functions** and **Events**.

The Serverless Framework is different from other application frameworks because:

- It manages your code as well as your infrastructure.
- It supports multiple languages (Node.js, Python, Java, and more).

9.6.1.2 Quick Start

This guide is designed to help you get started as quickly as possible.

Initial Setup

There are a few prerequisites you need to install and configure:

- Install Node.js 14.x or later version on your local machine. For details, see **Installing Node.js and NPM**.
- Install the Serverless Framework open-source CLI 3.28.1 or later version. For details, see **Installing the Open-source CLI of the Serverless Framework.**.

If you already have these prerequisites set up, you can skip ahead to deploy an example service.

Installing Node.js and NPM

Step 1 Install Node.js and NPM. For details about the download address, see the **download description**.

Step 2 At the end, you can run **node -v** from your command line and you will get a result like this:

```
$ node -v
vx.x.x
```

You can also run **npm -v** from your command line and you will get a result like this:

```
$ npm -v
x.x.x
```

----End

Installing the Open-source CLI of the Serverless Framework.

Step 1 Run this command in your terminal:

```
npm install -g serverless
```

Step 2 After the installation is complete, you can run **serverless -v** from your command line and you will get a result like this:

```
$ serverless -v
x.x.x
```

----End

Creating and Deploying a Serverless Service

The setup is completed, now you can create and deploy a serverless service.

Step 1 Create a new service.

1. Create a new service with the **huawei-nodejs** template.
   ```
   serverless create --template-url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs --path my-service
   ```
2. Install the dependencies.
   ```
   cd my-service
   npm install
   ```

Step 2 Set up the credentials. For details, see **credential settings**.

Step 3 Update the **serverless.yml**.

Update the **region** and **credentials** in **serverless.yml**.

Step 4 Deploy.

Use this command to deploy a service for the first time or to deploy all changes in the service after modifying the functions, events, or resources in the **serverless.yml** file. For details about this command, see **Deploy**.

```
serverless deploy
```

----End

9.6.1.3 Installation

Serverless is a Node.js CLI tool, so the first thing you need is to install Node.js on your machine.

Go to the **official Node.js website**, download and follow the **download description** to install Node.js on your local machine.

You can verify whether the Node.js is installed successfully by running **node --version** in your terminal. If installed, you can see the corresponding Node.js version number printed out.

Installing the Serverless Framework

Step 1 Next, install the Serverless Framework via **npm** which was already installed when you installed Node.js.

Step 2 Open a terminal and type **npm install -g serverless** to install Serverless.

```
npm install -g serverless
```


Step 3 Once the installation is done, you can verify whether Serverless is installed successfully by running the following command in your terminal:

```
serverless
```

To see which Serverless version of, run:

```
serverless --version
```

----End

Installing the FunctionGraph Plug-ins

To install the latest package from npm, run:

```
npm i --save serverless-huawei-functions
```

Setting up FunctionGraph

To run Serverless commands that issue requests to Huawei Cloud, you will need to set up your Huawei Cloud credentials For details, see **credential settings**

9.6.1.4 Credentials

The Serverless Framework needs access to your Huawei Cloud credentials so that it can create and manage resources on your behalf.

Creating a Huawei Cloud Account

Open **Huawei Cloud official website**, and then choose **Sign Up**. For details, see **Registering a HUAWEI ID and Enabling HUAWEI CLOUD Services**.

Getting the Credentials

You need to create credentials so that Serverless can use them to create resources in your project.

Step 1 Go to the **Access Keys** page to get the access keys of your Huawei Cloud account.

Step 2 Create a file named **credentials** containing the credentials that you have collected.

```
access_key_id=<collected in step 1>
secret_access_key=<collected in step 1>
```

Step 3 Save the credentials file somewhere secure. We recommend creating a folder in your root folder and putting it there, like **~/.fg/credentials**. Remember the path you saved it to.

----End

Updating the provider Configuration in serverless.yml

Open your **serverless.yml** file and update the **provider** section with the path to your credentials file (the path should be absolute). The result should be like this:

```
provider:
  name: huawei
  runtime: Node.js14.18
  credentials: ~/.fg/credentials
```


9.6.1.5 Service

A service is like a project. It is where you define your FunctionGraph functions and the events that trigger them, all in a file called **serverless.yml**.

To build your first Serverless Framework project, create a service.

Organizing

For the beginners, you can use a single service to define all of the functions and events.

```
myService/
  serverless.yml  # Contains all functions and infrastructure resources
```

As your application grows, you can split it into multiple services. Most users organize their services by workflows or data models, and group the functions related to those workflows and data models together.

```
users/
  serverless.yml # Contains 4 functions that do Users CRUD operations and the Users database
posts/
  serverless.yml # Contains 4 functions that do Posts CRUD operations and the Posts database
comments/
  serverless.yml # Contains 4 functions that do Comments CRUD operations and the Comments database
```

This makes sense since related functions usually use common infrastructure resources, and users want to keep those functions and resources together as a single unit of deployment for better organization and separation of concerns.

Creating

To create a service, use the **create** command. You can also input a path to create a directory and auto-name your service:

```
# Create service with Node.js template in the folder ./my-service
serverless create --template-url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs --path my-service
```

huawei-nodejs is an available runtime of FunctionGraph.

Check out the **Create** for all the details and options.

Contents

You will see the following files in your working directory:

- serverless.yml
- src/index.js

serverless.yml

Each service configuration is managed in the **serverless.yml** file. The main responsibilities of this file are:

- Declare a Serverless service.
- Define one or more functions in the service:
  - Define the provider the service will be deployed to (and the runtime if provided).
  - Define any custom plug-ins to be used.

- Define events that trigger functions to execute (such as HTTP requests).
- Allow events listed in the **events** section to automatically create the resources required for the event upon deployment.
- Allow flexible configuration using Serverless variables.

You can see the name of the service, the provider configuration, and the first function inside the **functions** definition. Any further service configuration will be done in this file.

```
# serverless.yml
service: my-fc-service

provider:
  name: huawei
  runtime: Node.js14.18
  credentials: ~/.fg/credentials # path must be absolute

plugins:
  - serverless-huawei-functions

functions:
  hello_world:
    handler: index.handler
```

index.js

The **index.js** file contains your exported functions.

Deploying

When you deploy a service, all of the functions and events in your **serverless.yml** are translated into calls to the Huawei Cloud API to dynamically define those resources.

To deploy a service, use the **deploy** command:

```
serverless deploy
```

Check out the **deployment guide** to learn more about deployments and how they work. Or, check out the **deploy command docs** for all the details and options.

Removing

To easily remove your service on Huawei Cloud, you can use the **remove** command.

Run **serverless remove** to trigger the removal process.

You will be notified of the process in the console when the removal starts. A success message is printed once the whole service is removed.

Only the service on your provider's infrastructure is removed during the removal process. The service directory will still remain on your local machine so you can still modify and (re)deploy it to another stage, region, or provider later on.

Version Pinning

The Serverless Framework is usually installed globally via **npm install -g serverless**. Therefore, the Serverless CLI is available for all your services.

Installing tools globally has the downside that the version cannot be pinned inside the **package.json**. This can lead to issues if you upgrade Serverless, but your

colleagues or CI system do not. You can use a feature in your **serverless.yml** without worrying that your CI system will deploy with an old version of Serverless.

- **Pinned version**

  To configure version pinning, define a **frameworkVersion** attribute in your **serverless.yml**. Whenever you run a Serverless command from the CLI, it checks whether your current Serverless version is in the **frameworkVersion** range. The CLI uses **Semantic Versioning** so you can pin it to an exact version or provide a range. In general, we recommend pinning to an exact version to ensure everybody in your team has the exact same setup and no unexpected problems occur.

  Examples

  Explicit version

  ```
  # serverless.yml

  frameworkVersion: '2.1.0'
  ```

  Version range

  ```
  # serverless.yml

  frameworkVersion: ^2.1.0 # >=2.1.0 && <3.0.0
  ```

Installing Serverless in an Existing Service

If you already have a Serverless service, and prefer to lock down the framework version using **package.json**, then you can install Serverless as follows:

```
# from within a service
npm install serverless --save-dev
```

- **Invoke serverless locally**

  To execute the locally installed Serverless, you have to reference the binary out of the **node_ modules** directory. An example is as follows:

  ```
  node ./node_modules/serverless/bin/serverless deploy
  ```

9.6.1.6 Functions

If you are using Huawei Cloud FunctionGraph as a provider, all functions inside the service are Huawei Cloud FunctionGraph functions.

Configuration

All of the Huawei Cloud FunctionGraph in your Serverless service can be found in **serverless.yml** under the **functions** attribute.

```
# serverless.yml
service: fg-service

provider:
  name: huawei

plugins:
  - serverless-huawei-functions

functions:
  first:
    handler: index.handler
```


Handler

The **handler** attribute should be the function name you have exported in your handler file.

For example, when you export a function with the name **handler** in **index.js**, your **handler** should be **handler: index.handler**.

```
// index.js
exports.handler = (event, context, callback) => {};
```

Memory Size and Timeout

The **memorySize** and **timeout** for the functions can be specified at the provider or function level. The provider definition enables all functions to share this configuration, whereas the function definition indicates that this configuration is only valid for the function.

The default **memorySize** is 256 MB and the default **timeout** is 30s if not specified.

```
# serverless.yml

provider:
  memorySize: 512
  timeout: 90

functions:
  first:
    handler: first
  second:
    handler: second
    memorySize: 256
    timeout: 60
```

Handler Signature

The signature of an event handler is:

```
function (event, context) { }
```

- **event**

  If the function is triggered by an APIG event specified, the **event** passed to the handler will be:

  ```
  // JSON.parse(event)
  {
    events: {
      "body": "",
      "requestContext": {
          "apiId": "xxx",
          "requestId": "xxx",
          "stage": "RELEASE"
      },
      "queryStringParameters": {
          "responseType": "html"
      },
      "httpMethod": "GET",
      "pathParameters": {},
      "headers": {
          "accept-language": "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2",
          "accept-encoding": "gzip, deflate, br",
          "x-forwarded-port": "443",
          "x-forwarded-for": "xxx",
          "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
          "upgrade-insecure-requests": "1",
          "host": "xxx",
  ```

```
      "x-forwarded-proto": "https",
      "pragma": "no-cache",
      "cache-control": "no-cache",
      "x-real-ip": "xxx",
      "user-agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0"
    },
    "path": "/apig-event-template",
    "isBase64Encoded": true
  }
}
```

- **context**

  The **context** parameter contains the runtime information about the function. For example, request ID, temporary AK, and function metadata. See **Developing an Event Function**.

9.6.1.7 Events

Simply put, events are the things that trigger your functions to run.

If you are using Huawei Cloud as your provider, events in the service are limited to the Huawei Cloud APIG and OBS. For details, see **Event list**.

Upon deployment, the Framework will set up the corresponding event configuration your function should listen to.

Configuration

Events belong to each function and can be found in the **events** attribute in **serverless.yml**.

```
# serverless.yml
functions:
  first: # Function name
    handler: index.http # Reference to file index.js & exported function 'http'
    events:
      - apigw:
          env_id: DEFAULT_ENVIRONMENT_RELEASE_ID
          env_name: RELEASE
          req_method: GET
          path: /test
          name: API_test
```

NOTE

Currently, only one event definition per function is supported.

Types

The Serverless Framework supports OBS and APIG events of the FunctionGraph. For details, see **Event list**.

Deployment

To deploy or update your functions and events, run:

```
serverless deploy
```

9.6.1.8 Deploy

The Serverless Framework is designed to provision functions, events and resources of FunctionGraph safely and quickly.

Deploying All

This is the main method for deploying with the Serverless Framework:

```
serverless deploy
```

Use this method when you have updated your function, event, or resource configuration in **serverless.yml** and you want to deploy that change (or multiple changes at the same time) to Huawei Cloud.

Working Principles

The Serverless Framework translates all syntax in **serverless.yml** to a configuration template.

1. The provider plugin parses **serverless.yml** configuration and translates it to Huawei Cloud resources.
2. The code of your functions is then packaged into a directory, zipped, and uploaded to the deployment bucket.
3. Resources are deployed.

   NOTE

   Use this in your CI/CD system, as it is the safest method for deployment.

Check out the **deploy command docs** for all the details and options.

9.6.1.9 Package

Packaging CLI Command

Using the Serverless CLI tool, you can package your project without deploying it to Huawei Cloud. This is best used with CI/CD workflows to ensure consistent deployable artifacts.

Running the following command will build and save all of the deployment artifacts in the **.serverless** directory of the service:

```
serverless package
```

Packaging Configuration

If you want to have more control over function artifacts and package methods.

You can use the **patterns** configuration.

- **Patterns**

  Patterns allow you to define globs that will be excluded/included from the resulting artifact. If you want to exclude files, you can use a global pattern prefixed with **!**, such as **!exclude-me/****. Serverless Framework will run the global patterns so that you can always re-include previously excluded files and directories.

  Examples

  Exclude all **node_modules** but then re-include a specific module (**node-fetch** in this case) using **exclude**

  ```
  package:
    patterns:
  ```



```
  - '!node_modules/**'
  - 'node_modules/node-fetch/**'
```

Exclude all files but **handler.js**

```
package:
  patterns:
    - '!src/**'
    - src/function/handler.js
```

NOTE

If you want to exclude directories, use the correct global syntax. The following is an example:

```
package:
  patterns:
    - '!tmp/**'
    - '!.git/**'
```

- Artifact

  For complete control over the packaging process, you can specify your own artifact ZIP file.

  Serverless will not zip your service if this is configured and therefore **patterns** will be ignored. Either you use **artifact** or **patterns**.

  The artifact option is especially useful if your development environment allows you to generate a deployable artifact like Maven does for Java.

  Example

```
service: my-service
package:
  patterns:
    - '!tmp/**'
    - '!.git/**'
    - some-file
  artifact: path/to/my-artifact.zip
```

- **Package functions separately**

  If you want even more control over your functions for deployment, you can configure them to be packaged separately. This allows for more control to optimize your deployment. To enable individual packaging, set **individually** to **true** in the packaging settings of the service or the function.

  Then for every function you can use the same **patterns** or **artifact** configuration options. The **patterns** options will be merged with the service options to create a **patterns** configuration for each function during packaging.

```
service: my-service
package:
  individually: true
  patterns:
    - '!excluded-by-default.json'
functions:
  hello:
    handler: handler.hello
    package:
      # We're including this file so it will be in the final package of this function only
      patterns:
        - excluded-by-default.json
  world:
    handler: handler.hello
    package:
      patterns:
        - '!some-file.js'
```

  You can also select which functions to be packaged separately, and have the rest use the service package by setting the **individually** to **true**.


```
service: my-service
functions:
  hello:
    handler: handler.hello
  world:
    handler: handler.hello
    package:
      individually: true
```

- **Development Dependencies**

  Serverless will auto-detect and exclude development dependencies based on the runtime your service is using to ensure that only the production relevant packages and modules are included in your ZIP file. This drastically reduces the overall size of the deployment package which will be uploaded to the cloud provider.

  You can opt out of automatic exclusion of development dependency by setting the **excludeDevDependencies** to **false**:

```
package:
  excludeDevDependencies: false
```

9.6.1.10 Variables

The Serverless Framework provides a powerful variable system which allows you to add dynamic data into your **serverless.yml**. With Serverless variables, you will be able to do the following:

- Reference & load environment variables.
- Reference & load variables from CLI options.
- Recursively reference properties of any type from the same **serverless.yml** file.
- Recursively reference properties of any type from other **YAML** or **JSON** files.
- Recursively nest variable references for ultimate flexibility.
- Combine multiple variable references to overwrite each other.

Constraints

You can only use variables in **values** attribute instead of key attribute in **serverless.yml**. So you cannot use variables to generate dynamic logical IDs in the custom resources.

Referencing Environment Variables

To reference environment variables, use the *${env:someProperty}* syntax in your **serverless.yml**.

```
service: new-service

provider:
  name: huawei
  runtime: Node.js14.18
  credentials: ~/.fg/credentials # path must be absolute
  environment:
    variables:
      ENV_FIRST: ${env:TENCENTCLOUD_APPID}

plugins:
  - serverless-huawei-functions

functions:
```



```
hello:
  handler: index.hello
```

9.6.2 CLI Reference

Welcome to the Serverless CLI reference for FunctionGraph!

NOTE

Before continuing, **Huawei Cloud credentials** are required for using the CLI.

9.6.2.1 Create

Creates a new service in the current working directory based on the specified template.

- Create a service in the current working directory:
  ```
  serverless create --template-url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs
  ```
- Create a service in a new folder using a custom template:
  ```
  serverless create --template-url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs --path my-service
  ```

Options

- **--template-url** or **-u**: A URL pointing to a remotely hosted template. **Required if --template and --template-path are not present.**
- **--template-path**: The local path of your template. **Required if --template and --template-url are not present.**
- **--path** or **-p**: The path where the service is created.
- **--name** or **-n**: The name of the service in **serverless.yml**.

Examples

- **Create a new service**
  ```
  serverless create --template-url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs --name my-special-service
  ```
  This example generates a Node.js runtime in the current working directory for the service with Huawei as the provider.
- **Create a named service in a (new) directory**
  ```
  serverless create --template-url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs --path my-new-service
  ```
  This example will generate a Node.js runtime in the **my-new-service** directory for the service with Huawei as the provider. This directory will be automatically created if it does not exist. Otherwise Serverless will use the already present directory.

  Additionally, Serverless will rename the service according to the path you provide. In this example, the service will be renamed to **my-new-service**.

9.6.2.2 Install

Install a service from a GitHub URL in the current working directory.

```
serverless install --url https://github.com/some/service
```


Options

- **--url** or **-u**: The services GitHub URL, which is required.
- **--name** or **-n**: Name for the service.

Examples

- **Install a service from a GitHub URL**
  ```
  serverless install --url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs
  ```
  This example will download the **.zip** file of the **huawei-nodejs** service from GitHub, create a new directory named **huawei-nodejs** in the current working directory, and unzip the file in this directory.
- **Install a service from a GitHub URL with a new service name**
  ```
  serverless install --url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs--name my-huawei-service
  ```
  The execution process is as follows:
  a. Download the **.zip** file of the **huawei-nodejs** service from GitHub.
  b. Create a new directory with the name **my-huawei-service** in the current working directory.
  c. Unzip the files in this directory.
  d. Rename the service to **my-huawei-service** if **serverless.yml** exists in the service root.
- **Install a service from a directory in a GitHub URL**
  ```
  serverless install --url https://github.com/zy-linn/examples/tree/v3/legacy/huawei-nodejs
  ```
  In this example, the **huawei-nodejs** service will be downloaded from GitHub.

9.6.2.3 Package

The **sls package** command packages your entire infrastructure into the **.serverless** directory by default and makes it ready for deployment.

Example

```
serverless package
```

In this example, your service will be packaged. The package will be generated in the default **.serverless** directory.

9.6.2.4 Deploy

The **serverless deploy** command deploys your entire service via the Huawei Cloud API. Run this command when you have edited the **serverless.yml** file.
```
serverless deploy
```

Artifacts

After running the **serverless deploy** command, all created deployment artifacts will be placed in the **.serverless** folder of the service.

9.6.2.5 Info

The **serverless info** command displays information about the deployed service. Run this command in your services directory.


```
serverless info
```

9.6.2.6 Invoke

Invoke a deployed function. You can send event data, read logs, and view other important information about function calls.

```
serverless invoke --function functionName
```

Options

- **--function** or **-f**: The name of the function that you want to invoke. **Required**
- **--data** or **-d**: Data you want to pass into the function.
- **--path** or **-p**: The path to a JSON file which contains the input data to be transferred to the invoked function. This path is relative to the root directory of the service.

Examples

- **Simple function invocation**

  ```
  serverless invoke --function functionName
  ```

  In this example the deployed function will be invoked and the result of the invocation will be displayed in the terminal.

- **Function invocation with data**

  ```
  serverless invoke --function functionName --data '{"name": "Bob"}'
  ```

  In this example, the function will be invoked with the provided data and the result will be displayed in the terminal.

- **Function invocation with passed data**

  ```
  serverless invoke --function functionName --path lib/event.json
  ```

  In this example, the **JSON** data in the **lib/event.json** file will be passed (relative to the root of the service) while invoking the specified or deployed function.

  Example of **event.json**

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

9.6.2.7 Logs

View the logs of a specific function.

```
serverless logs --function functionName
```

Options

- **--function** or **-f**: The function for obtaining the logs. **Required**.
- **--count** or **-c**: The number of logs to display.

Example

Retrieve logs

```
serverless logs --function functionName
```

This will display logs for the specified function.

9.6.2.8 Remove

The **serverless remove** command will remove the deployed service defined in your current working directory from the provider.

```
serverless remove
```

NOTE

Only the deployed service and all its resources will be removed. The code on the local computer will be retained.

Example

Remove service

```
serverless remove
```

This command will remove the deployed service in your current working directory.

9.6.3 Event list

9.6.3.1 APIG Events

FunctionGraph can create function-based API endpoints through APIG.

To create HTTP endpoints as event sources for FunctionGraph, use the **HTTP** event syntax.

HTTP Endpoint

In this setting, the **first** function should be run when someone accesses the API endpoint via a **GET** request. You can get the URL for the endpoint by running the **serverless info** command after deploying your service.

Here is an example:

```
# serverless.yml

functions:
  hello:
    handler: index.hello
    events:
      - apigw:
          env_id: DEFAULT_ENVIRONMENT_RELEASE_ID
          env_name: RELEASE
          req_method: GET
          path: /test
          name: API_test
```

NOTE

For details about the handler signature used for such events, see the **Handler**.

9.6.3.2 OBS Events

Huawei Cloud functions can be triggered by different event sources, which can be defined and configured through **events**.

OBS Events

In this example, an OBS event is set. Each time an object is uploaded to **my-service-resource**, the event triggers the **first** function.

```
# serverless.yml

functions:
  first:
    handler: index.first
    events:
      - obs:
          bucket: bucket
          events:
            - s3:ObjectCreated:Put
            - s3:ObjectCreated:Post
// index.js

exports.first = async (event, context) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify({
      message: 'Hello!',
    }),
  };

  return response;
};
```


10 Automated Deployment

10.1 Preparing an Environment

This section uses a Linux host as an example to describe how to set up a function CI/CD environment using KooCLI and **DevCloud**.

ECS

This server functions as a DevCloud task deployment host to deploy and update FunctionGraph functions.

- Specifications: 1 vCPU | 1 GiB
- Image: CentOS 8.2 64 bits
- Other: Configure an EIP because you will need to install the Python library and DevCloud and configure the ECS as a deployment host.
- DevCloud uses this server as a deployment host through port 22 over SSH. For high security, add the following IP addresses to the security group of the server. Otherwise, authorization will fail.

  42.202.130.147

  49.4.3.11

  122.112.212.206

  139.159.226.153

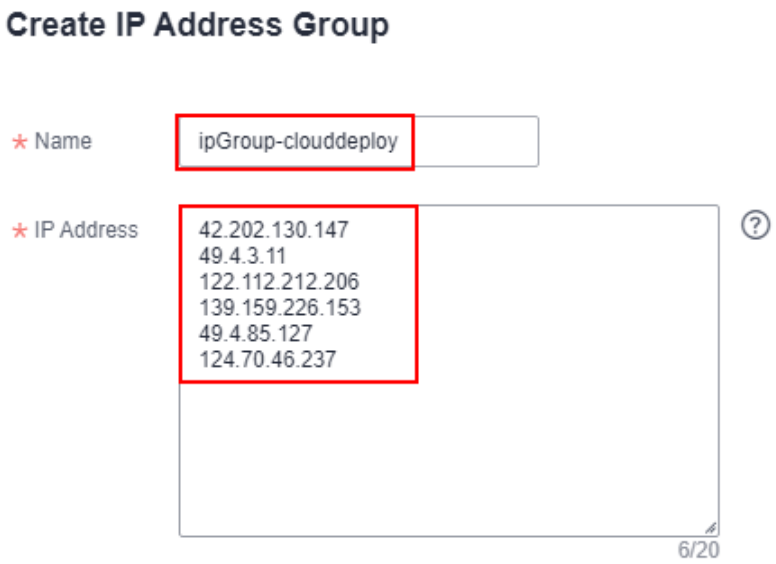
  49.4.85.127

  124.70.46.237

Configuring a Security Group for the Deployment Host

1. Log in to the ECS console. In the navigation pane, choose **Security Group** to go to the network console.
2. In the navigation pane of the network console, choose **Access Control** > **IP Address Groups**. Click **Create IP Address Group**, set parameters shown in **Figure 10-1**, and click **OK**.

Figure 10-1 Creating an IP address group

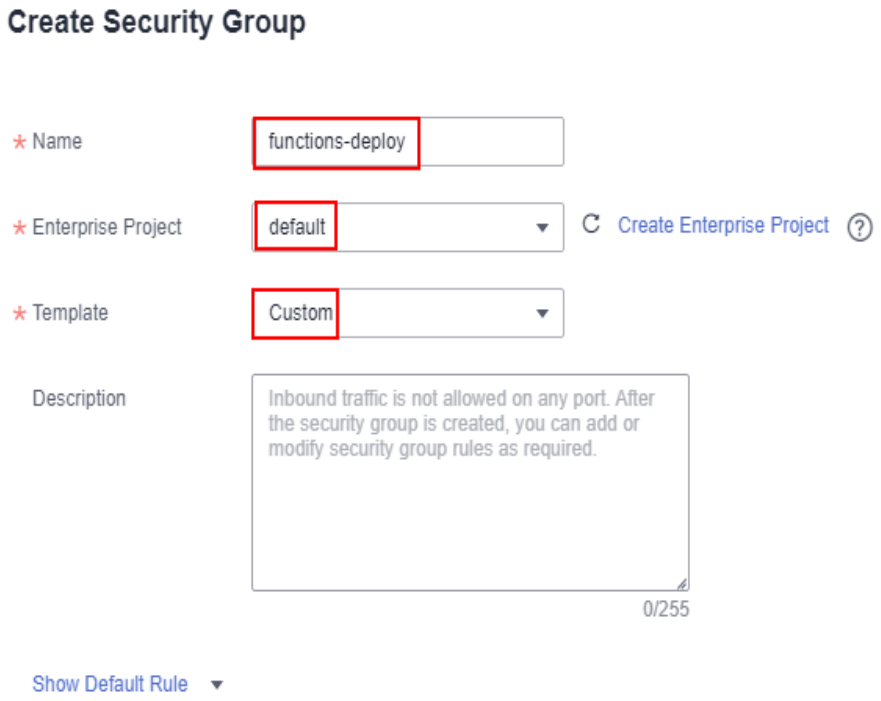


- **Name**: Enter **ipGroup-clouddeploy**.
- **IP Address**:

```
42.202.130.147
49.4.3.11
122.112.212.206
139.159.226.153
49.4.85.127
124.70.46.237
```

3. In the navigation pane, choose **Access Control** > **Security Groups**. Click **Create Security Group**, set parameters shown in **Figure 10-2**, and click **OK**.

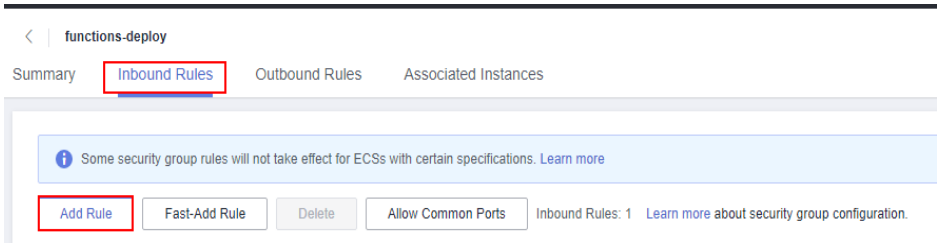
Figure 10-2 Configuring a security group



- **Name**: Enter **functions-deploy**.
- **Enterprise Project**: Enter **default**.
- **Template**: Select **Custom**.

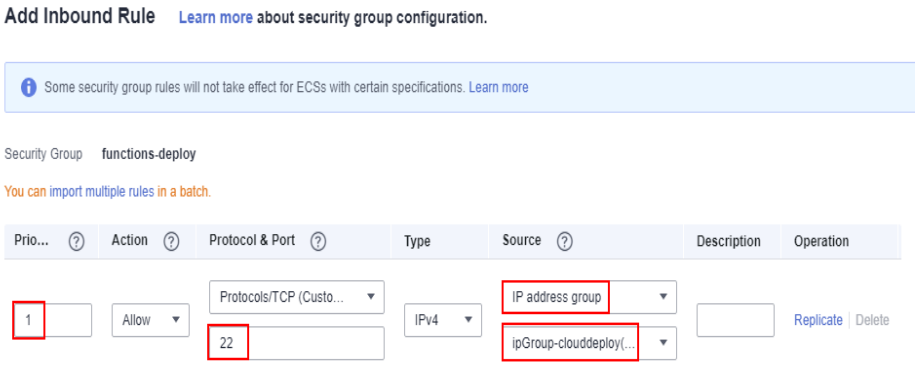
4. Go to the details page of **functions-deploy**, click the **Inbound Rules** tab, and click **Add Rule** to add an inbound rule.

Figure 10-3 Adding a rule



Set **Priority** to **1**, **Protocol & Port** to **22**, and **Source** to the IP address group **ipGroup-clouddeploy**. Then click **OK**.

Figure 10-4 Adding an inbound rule



5. Return to the ECS console, click the ECS name, and change the security group to **functions-deploy**.

Installing Python Libraries

Run the following commands to install the pyyaml and pycryptodome libraries: The two libraries will respectively parse the **cam.yaml** configuration file of your function and encrypts and decrypts the function's environment variables.

```
pip3 install pyyaml
pip3 install pycryptodome
```

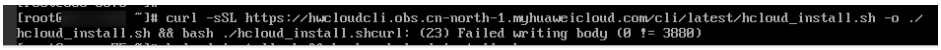
Installing KooCLI

1. Install KooCLI.

   Remotely log in to the purchased ECS, and run the following command to install KooCLI:

   ```
   curl -sSL https://ap-southeast-3-hwcloudcli.obs.ap-southeast-3.myhuaweicloud.com/cli/latest/hcloud_install.sh -o ./hcloud_install.sh && bash ./hcloud_install.sh
   ```

   Figure 10-5 Installing KooCLI

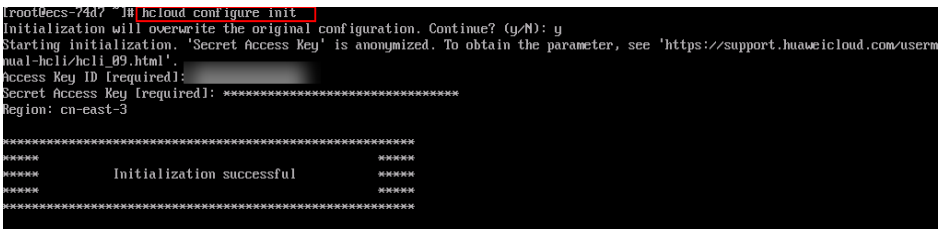
2. Initialize KooCLI.

   Run the following command to initialize KooCLI:

   ```
   hcloud configure init
   ```

   Enter an access key ID (AK), secret access key (SK), and region name. For details about how to obtain them, see **3** and **4**.

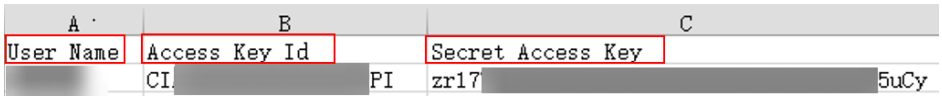
   Figure 10-6 Initializing KooCLI

3. Obtain an access key (AK/SK).

- If you have access to the console, log in, and create an access key on the **My Credentials** page. For details, see **Creating an Access Key**. An AK/SK file is downloaded. Generally, it is named **credentials.csv**. As shown in the following figure, the file contains a username, AK, and SK.

  Figure 10-7 Content of the credentials.csv file

- If you do not have access to the console, request the administrator to create an access key for you on the IAM console in case your access key is lost or needs to be reset. For details, see **Managing Access Keys for an IAM User**.

4. Obtain a region name.

   For details, see **Regions and Endpoints**.

   Figure 10-8 Obtaining region information

Region Name	Region
AF-Johannesburg	af-south-1
AP-Bangkok	ap-southeast-2
AP-Singapore	ap-southeast-3

10.2 Hosting Function Code with DevCloud

10.2.1 Step 1: Create a Project

1. Log in to the **DevCloud** console.
2. Choose **ProjectMan** in the navigation pane, and click **Access Service**.
3. Click **New Project** and choose **New** > **Scrum**.
4. Enter project name **function** and retain the default settings for other parameters.
5. Click **OK**.

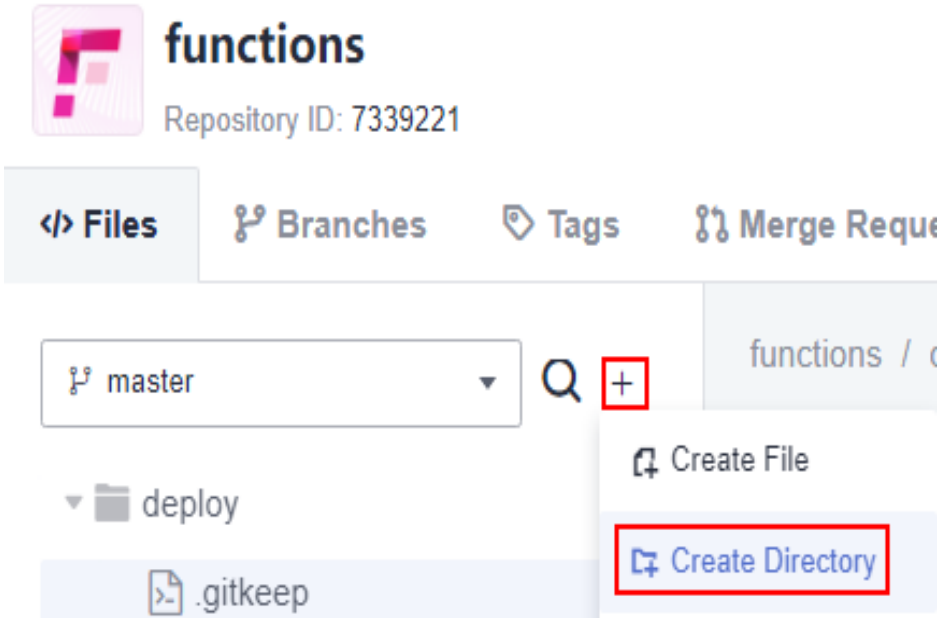
10.2.2 Step 2: Host Function Code

1. On the DevCloud console, choose **Code** > **CodeHub**, and click **Create Directly**.

   Create a code repository named **functions**, and retain the default settings for other parameters.

2. Go to the **functions** repository created in **1**. Create the **deploy** directory for storing the function deployment script **deploy.py**.

Figure 10-9 Creating a directory

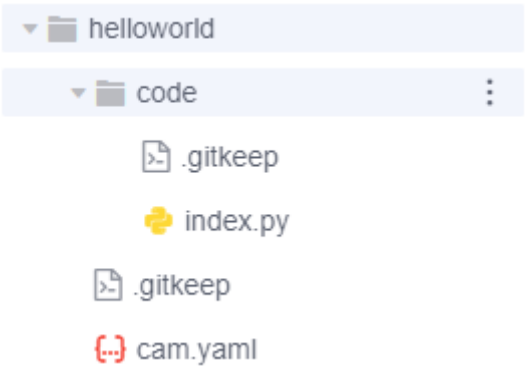


NOTE

The **deploy.py** script will read the function configuration file **cam.yaml** and construct an **hcloud** command to update the function code and configuration. For details about the configurations in **cam.yaml**, see **Analyzing cam.yaml**. Logs generated when executing this script will be recorded in the **/home/function/deploy/function.log** file.

3. Create another directory named **helloworld**, with a complete structure.

Figure 10-10 Complete function directory structure



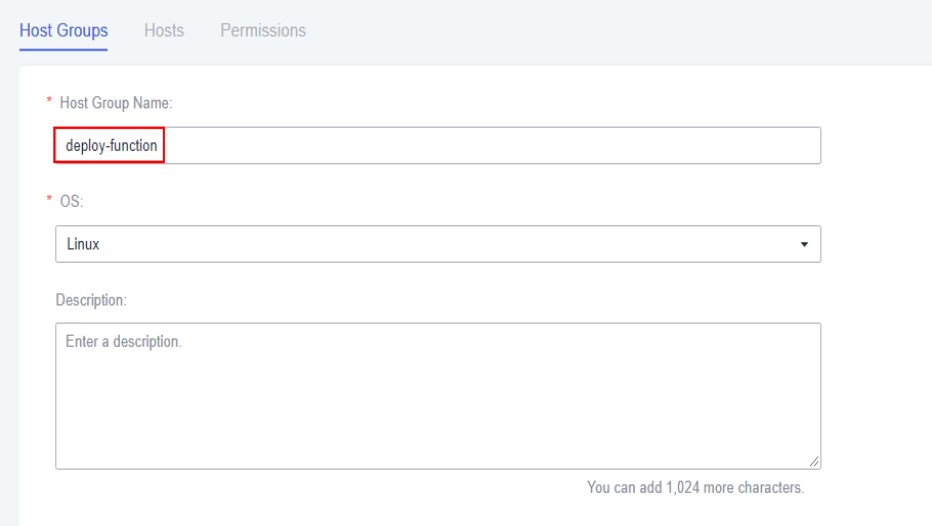
- **helloworld**: the **helloworld** function
- **cam.yaml**: the configuration file
- **code**: function code directory, which stores the **index.py** file

10.2.3 Step 3: Configure a Deployment Host

1. On the DevCloud console, choose **Settings** > **General Settings**, choose **Host Groups** in the navigation pane, and click **Add Host Group**.

2. Enter host group name **deploy-function** and click **Save**.

   Figure 10-11 Entering a host group name

3. On the **Hosts** tab page, click **Import ECS**.

   Import the ECS configured in **Preparing an Environment**, enter the ECS username, password, and SSH port 22, read and agree to the agreement, and add the ECS.

4. Wait till the connectivity test is successful.

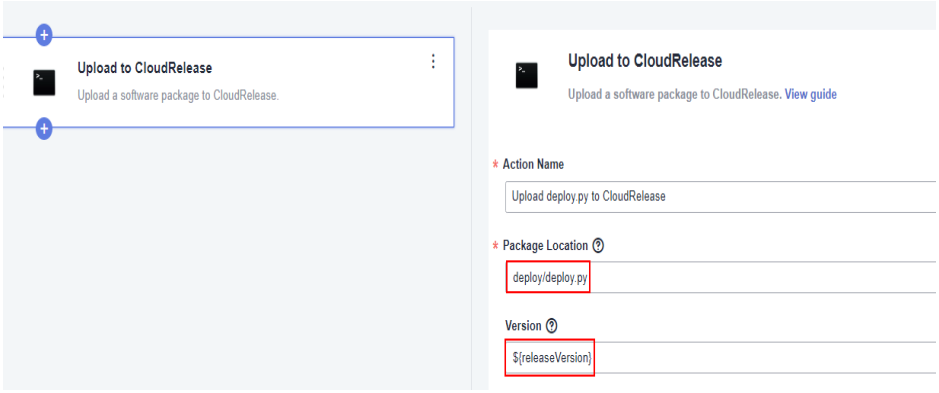
10.2.4 Step 4: Set Up a Pipeline for Updating the Function Deployment Script

The pipeline allows you to release the function deployment script **deploy.py** to the deployment host for function updates.

Creating a Build Task

1. Choose **Build & CloudArtifact** > **CloudBuild**, and click **Create Task**.
2. Select **functions** for **Source Code Repository** and **Blank Template** for the template, and click **OK**.
3. On the **Build Actions** tab page, add only **Upload to CloudRelease**. The action is displayed in the left pane.
4. Click the **Upload to CloudRelease** action, and set parameters.

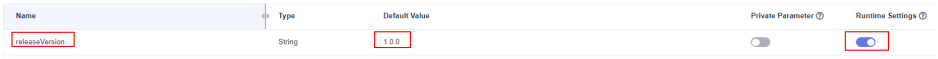
Figure 10-12 Setting parameters



- **Action Name**: Enter **Upload deploy.py to CloudRelease**.
- **Package Location**: Enter **deploy/deploy.py**.
- **Version**: Enter **${releaseVersion}**.

5. On the **Parameters** tab page, add **releaseVersion** and enable **Runtime Settings**.

Figure 10-13 Configuring the version parameter

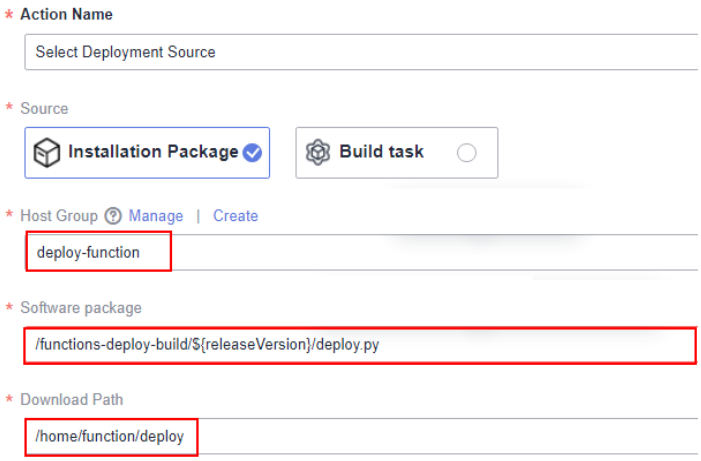


6. On the **Basic Information** tab page, change the task name to **functions-deploy-build** and click **Save**.

Creating a Deployment Task

1. Choose **Build & CloudArtifact** > **CloudDeploy**, and click **Create Task**.
2. Select **Blank Template** and click **Next**.
3. Add only the **Select Deployment Source** action.
4. Configure the **Select Deployment Source** action.

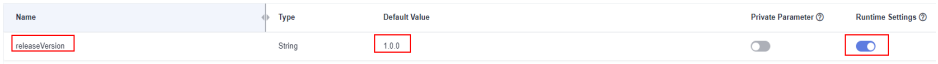
Figure 10-14 Configuring the Select Deployment Source action



- **Host Group**: Select **deploy-function**.
- **Software package**: Enter **/functions-deploy-build/${releaseVersion}/deploy.py**.
- **Download Path**: Enter **/home/function/deploy**.

5. On the **Parameters** tab page, add **releaseVersion** and enable **Runtime Settings**.

Figure 10-15 Setting parameters

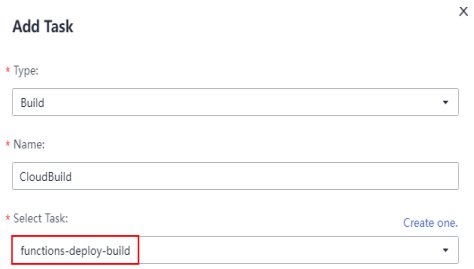


6. On the **Basic Information** tab page, change the task name to **update-function-deploy** and click **Save**.

Configuring a Pipeline

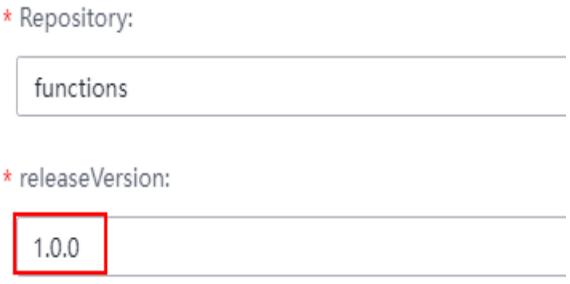
1. Choose **Build & CloudArtifact** > **CloudPipeline**, and click **Create Pipeline**.
2. Select **functions** for **Repository** and **Blank Template** for the template.
3. Configure **Build and Check**.
   a. Add a build task, and select the **function-deploy-build** task.

Figure 10-16 Adding a task



   b. Set **releaseVersion** as a pipeline parameter.

Figure 10-17 Setting releaseVersion



   c. Click **Save**.
4. Configure a deployment task.
   a. Add a stage named **Deploy** after **Build_and_Check**, set **Task Execution** to **Serial**, and click **Save**.

Figure 10-18 Adding a stage

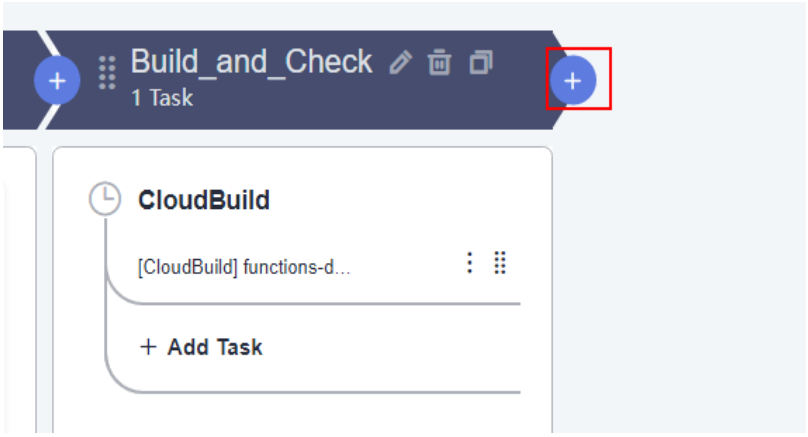
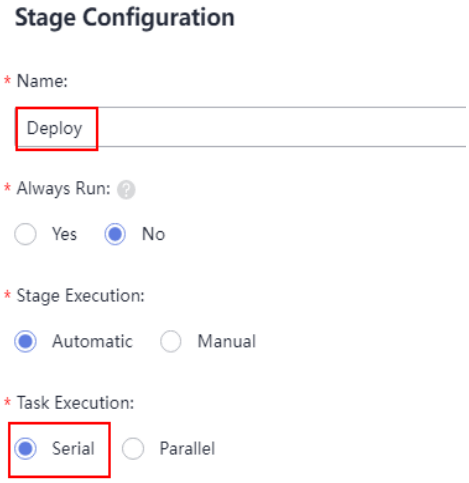


Figure 10-19 Configuring the stage



b. Click **Add Task** to add a deployment task named **DeployScript**, and select the **update-function-deploy** task.

Figure 10-20 Adding a task

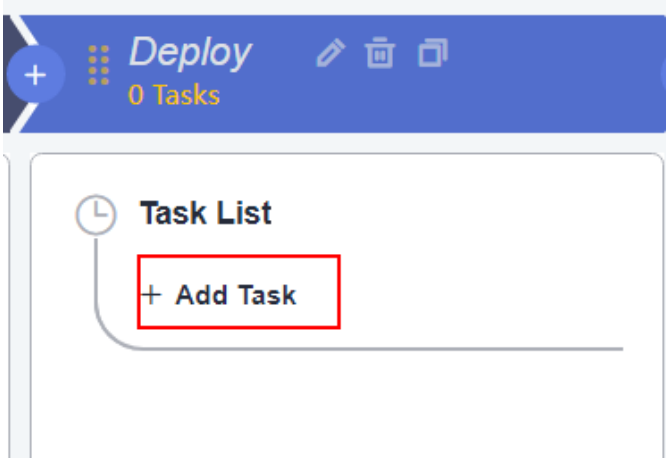
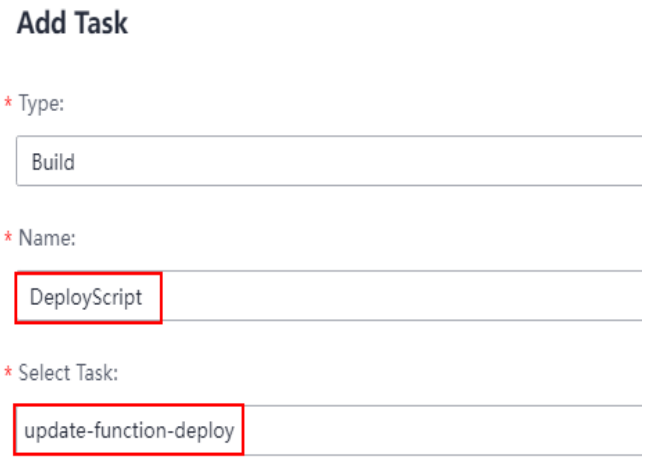
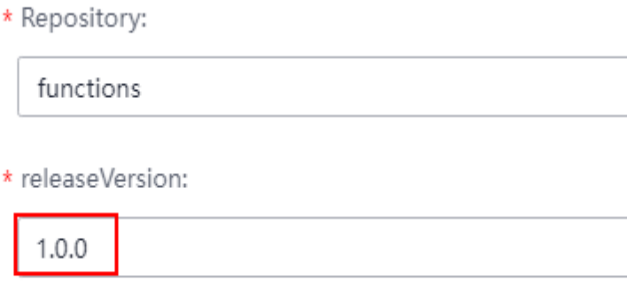


Figure 10-21 Configuring the task



Set **releaseVersion** as a pipeline parameter.

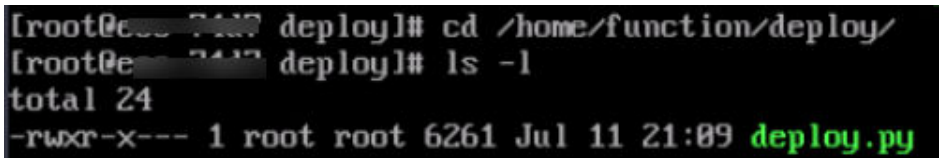
Figure 10-22 Setting releaseVersion



c. Click **Save**.

5. On the **Basic Information** tab page, change the pipeline task name to **pipeline-update-function-deploy** and click **Save**.
6. Execute the pipeline.
   a. Set the runtime parameter **releaseVersion** to **1.0.0** and click **Execute**.
   b. Wait till the **deploy.py** script is released.

Figure 10-23 Execution successful



10.2.5 Step 5: Set Up a Function Update Pipeline

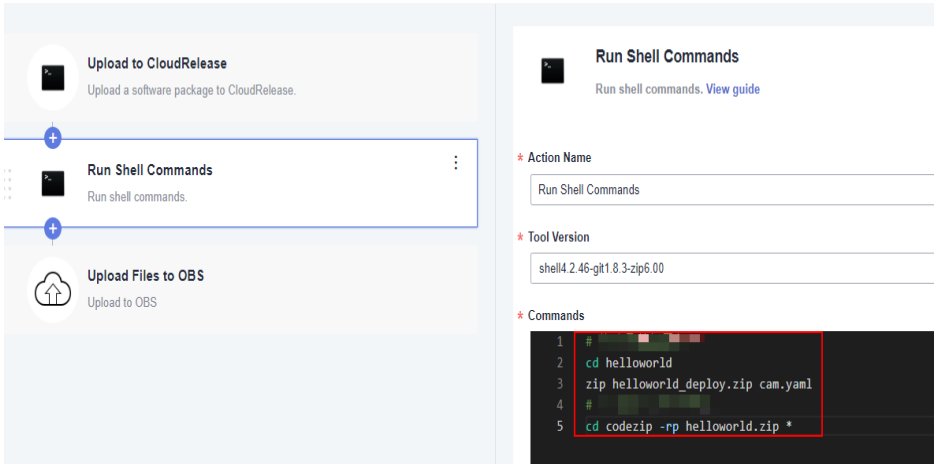
The pipeline allows you to release and update the **helloworld** function code in the **functions** repository to FunctionGraph.

Creating a Build Task

1. Choose **Build & CloudArtifact** > **CloudBuild**, and click **Create Task**.
2. Select **functions** for **Source Code Repository** and **Blank Template** for the template.
3. Add these three actions: **Run Shell Commands**, **Upload Files to OBS**, and **Upload Deployment Package to CloudRelease**.
   a. Configure the **Run Shell Commands** action.

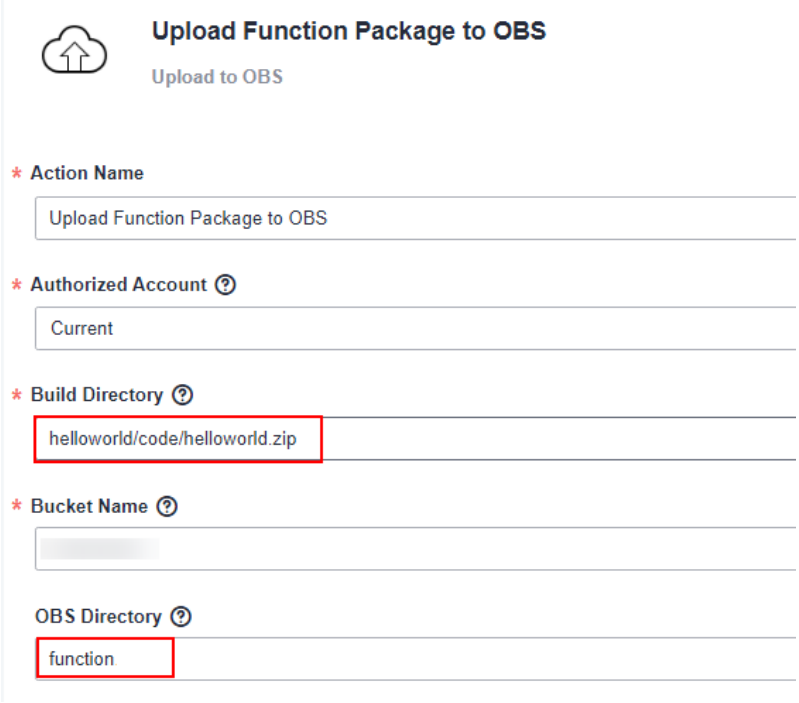
```
# Build a function deployment package.
cd helloworld
zip helloworld_deploy.zip cam.yaml
# Build a function code package.
cd code
zip -rp helloworld.zip *
```

Figure 10-24 Run Shell Commands



   b. Configure the **Upload Files to OBS** action.

Figure 10-25 Upload Files to OBS



- **Action Name**: Enter **Upload Function Package to OBS**.
- **Build Directory**: Enter **helloworld/code/helloworld.zip**.
- **Bucket Name**: Specify a private bucket to store the function code ZIP package.
- **OBS Directory**: Enter **function**.

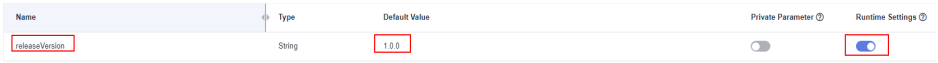
c. Configure the **Upload Deployment Package to CloudRelease** action.

Figure 10-26 Upload Deployment Package to CloudRelease

- **Action Name**: Enter **Upload Deployment Package to CloudRelease**.
- **Package Location**: Enter **helloworld/helloworld_deploy.zip**.
- **Version**: Enter **${releaseVersion}**.

4. On the **Parameters** tab page, add **releaseVersion** and enable **Runtime Settings**.

Figure 10-27 Setting parameters



5. On the **Basic Information** tab page, change the task name to **pipeline-update-function-deploy** and click **Save**.

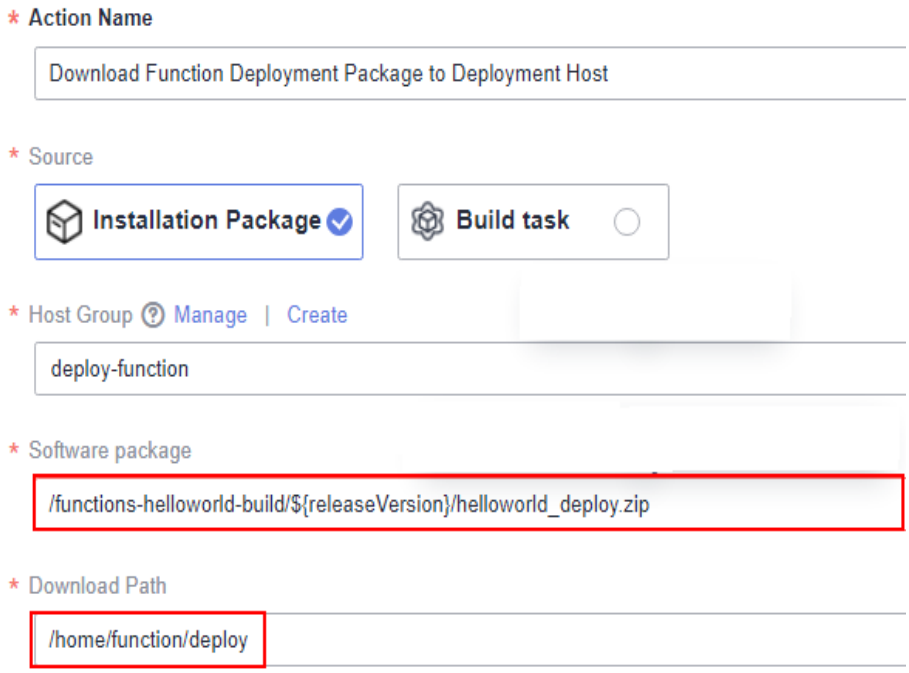
Creating a Deployment Task

1. Choose **Build & CloudArtifact** > **CloudDeploy**, and click **Create Task**.
2. Select **Blank Template** and click **Next**.
3. Add the actions **Select Deployment Source** and **Run Shell Commands**.

Figure 10-28 Adding deployment actions

a. Configure the **Select Deployment Source** action.

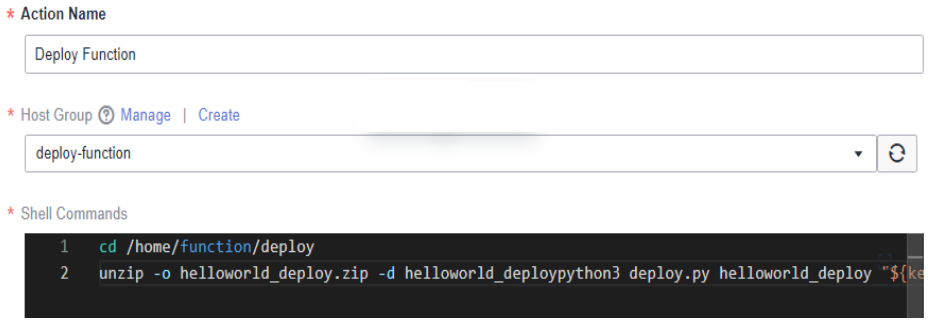
Figure 10-29 Setting action name to "Download Function Deployment Package to Deployment Host"



- **Action Name**: Enter **Download Function Deployment Package to Deployment Host**.
- **Host Group**: Select **deploy-function**.
- **Software package**: Select **/functions-helloworld-build/${releaseVersion}/helloworld_deploy.zip**.
- **Download Path**: Enter **/home/function/deploy**.

b. Configure the **Run Shell Commands** action.

Figure 10-30 Setting action name to "Deploy Function"



- **Action Name**: Enter **Deploy Function**.
- **Host Group**: Select **deploy-function**.

- **Shell Commands**:

```
cd /home/function/deploy
unzip -o helloworld_deploy.zip -d helloworld_deploy
python3 deploy.py helloworld_deploy "${key}"
```

4. Add two parameters.
   - **releaseVersion**: Use the default value **1.0.0** and enable **Runtime Settings**.
   - **key**: Enter a key and enable **Private Parameter**.

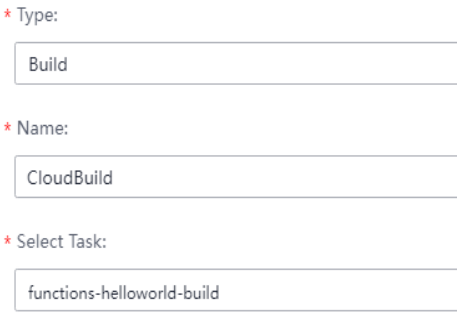
   Figure 10-31 Setting parameters

5. On the **Basic Information** tab page, change the task name to **update-function-deploy** and click **Save**.

Configuring a Pipeline

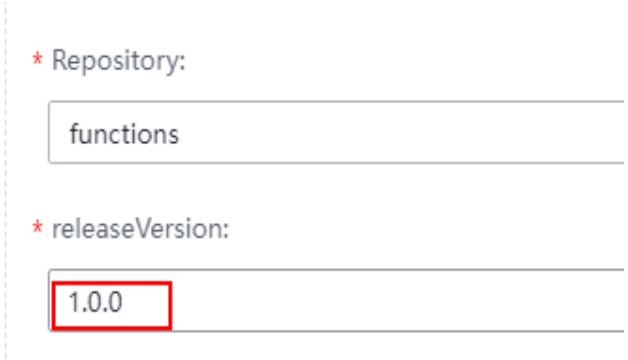
1. Choose **Build & CloudArtifact** > **CloudPipeline**, and click **Create Pipeline**.
2. Select **functions** for **Source Code Repository** and **Blank Template** for the template.
3. Configure **Build and Check**.
   a. Add a build task, and select the **functions-helloworld-build** task.

   Figure 10-32 Adding a task



   b. Set **releaseVersion** as a pipeline parameter.

   Figure 10-33 Setting releaseVersion

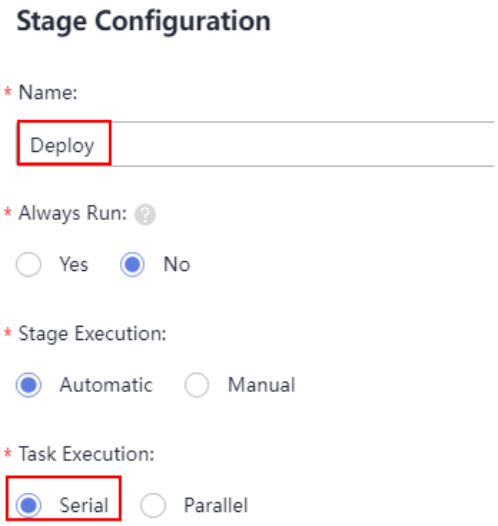


c. Click **Save**.

4. Configure a deployment task.

a. Add a stage named **Deploy** after **Build_and_Check**, set **Task Execution** to **Serial**, and click **Save**.

Figure 10-34 Configuring the stage



b. Click **Add Task** to add a deployment task named **DeployhelloworldScript**, and select the **update-function-deploy** task.

Figure 10-35 Clicking Add Task

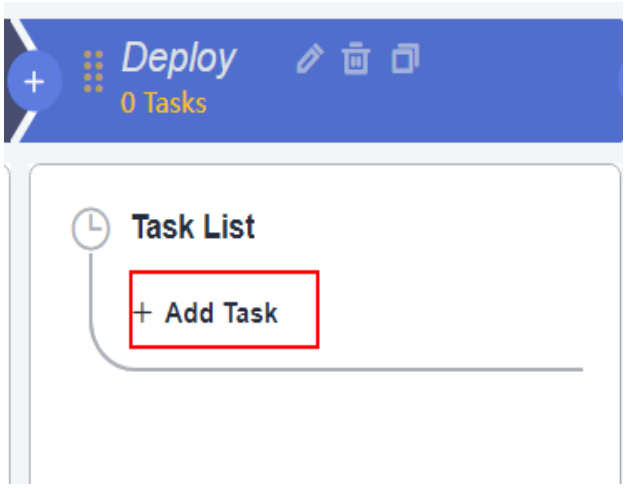
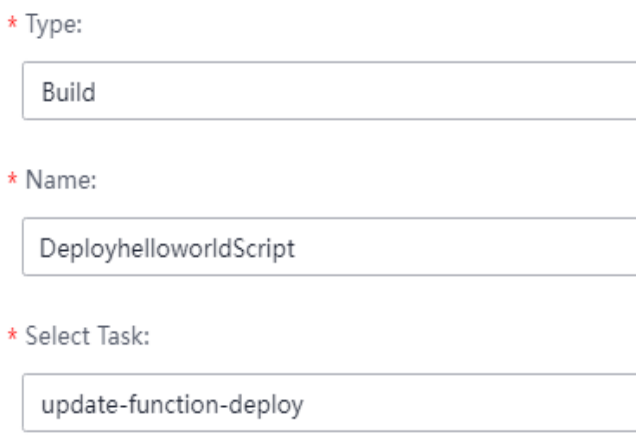
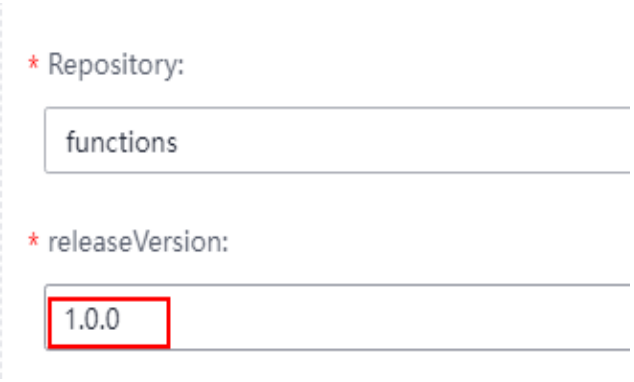


Figure 10-36 Adding a task



c. Set **releaseVersion** as a pipeline parameter.

Figure 10-37 Setting releaseVersion



d. Click **Save**.

5. On the **Basic Information** tab page, change the pipeline task name to **pipeline-update-function-helloworld** and click **Save**.
6. Execute the pipeline.

   Set the runtime parameter **releaseVersion** to **1.0.0** and click **Execute**.

10.3 Sample Code of deploy.py

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

import os
import sys
import json
import logging
import subprocess
from yaml import load
from base64 import b64decode
from Crypto.Cipher import AES

# need: pip install pyyaml
try:
    from yaml import CLoader as Loader, CDumper as Dumper
except ImportError:
    from yaml import Loader, Dumper
```



```
logging.basicConfig(level=logging.INFO,
                    filename='function.log',
                    filemode='a',
                    format='%(asctime)s - %(pathname)s[line:%(lineno)d] - %(levelname)s: %(message)s')


def decrypt(json_input, key):
    # We assume that the key was securely shared beforehand
    try:
        b64 = json.loads(json_input)
        json_k = ['nonce', 'header', 'ciphertext', 'tag']
        jv = {k: b64decode(b64[k]) for k in json_k}
        cipher = AES.new(key.encode(), AES.MODE_GCM, nonce=jv['nonce'])
        cipher.update(jv['header'])
        plaintext = cipher.decrypt_and_verify(jv['ciphertext'], jv['tag'])
        return plaintext.decode()
    except (ValueError, KeyError) as e:
        raise e


def generate_update_function_config_cmd(new_config, old_config, key):
    # Function handler
    handler = new_config['handler']
    # Runtime (required and not modifiable)
    runtime = new_config['runtime']
    # Memory
    memory_size = new_config['memorySize']
    # Timeout
    timeout = new_config['timeout']
    # Project ID
    project_id = new_config['projectID']
    # Command for updating the function configuration
    update_cmd = f'hcloud FunctionGraph UpdateFunctionConfig'
                 f' --cli-region="{region}"'
                 f' --function_urn="{function_urn}"'
                 f' --project_id="{project_id}"'
                 f' --handler="{handler}"'
                 f' --timeout={timeout}'
                 f' --memory_size={memory_size}'
                 f' --runtime="{runtime}"'
                 f' --func_name="{function_name}"'

    # Environment variables
    # Environment variables are directly overwritten. Manually configured variables that are not updated to
the cam.yaml file will be lost.

    user_data = new_config.get('userData', None)
    if user_data is not None:
        user_date_json_str = json.dumps(user_data)
        user_date_json_str = json.dumps(user_date_json_str)
        update_cmd = update_cmd + f' --user_data={user_date_json_str}'

    encrypted_user_data = new_config.get('encryptedUserData', None)
    if encrypted_user_data is not None:
        encrypted_user_data = decrypt(encrypted_user_data, key)
        encrypted_user_date_json_str = json.dumps(encrypted_user_data)
        update_cmd = update_cmd +
                     f' --encrypted_user_data={encrypted_user_date_json_str}'

    # Keep this part if a VPC is used.
    vpc_config = old_config.get('func_vpc', None)
    if vpc_config is not None:
        update_cmd = update_cmd +
                     f' --func_vpc.vpc_name={vpc_config["vpc_name"]}'
                     f' --func_vpc.vpc_id={vpc_config["vpc_id"]}'
                     f' --func_vpc.subnet_id={vpc_config["subnet_id"]}'
                     f' --func_vpc.cidr={vpc_config["cidr"]}'
                     f' --func_vpc.subnet_name={vpc_config["subnet_name"]}'
```



```
                f' --func_vpc.gateway={vpc_config["gateway"]}'

    # Keep "xrole": "function-admin" and "app_xrole": "function-admin" if an agency is specified.
    xrole_config = old_config.get('xrole', None)
    if xrole_config is not None:
        update_cmd = update_cmd + f' --xrole="{xrole_config}"'

    app_xrole_config = old_config.get('app_xrole', None)
    if app_xrole_config is not None:
        update_cmd = update_cmd + f' --app_xrole="{app_xrole_config}"'

    # Configure the initializer and initialization timeout.
    initializer_handler = new_config.get('initializerHandler', None)
    initializer_timeout = new_config.get('initializerTimeout', None)
    if initializer_handler is not None and initializer_timeout is not None:
        update_cmd = update_cmd +
                f' --initializer_handler="{initializer_handler}" '
                f'--initializer_timeout={initializer_timeout}'

    # Concurrency settings
    strategy_config = new_config.get('strategyConfig', None)
    if strategy_config is not None:
        concurrency = strategy_config.get('concurrency', None)
        # Maximum number of concurrent requests per instance
        concurrent_num = strategy_config.get('concurrentNum', None)
        update_cmd = update_cmd +
                f' --strategy_config.concurrency="{concurrency}" '
                f'--strategy_config.concurrent_num={concurrent_num}'

    # Keep this part if a file system is mounted to the function.
    mount_config = old_config.get('mount_config', None)
    if mount_config is not None:
        mount_user = mount_config["mount_user"]
        update_cmd = update_cmd +
                f' --mount_config.mount_user.user_id={mount_user["user_id"]}'
                f' --mount_config.mount_user.user_group_id={mount_user["user_group_id"]}'
        func_mounts = mount_config["func_mounts"]
        i = 1
        for func_mount in func_mounts:
            update_cmd = update_cmd +
                    f' --mount_config.func_mounts.{i}.mount_resource="{func_mount["mount_resource"]}"'
                    f' --mount_config.func_mounts.
{i}.mount_share_path="{func_mount["mount_share_path"]}"'
                    f' --mount_config.func_mounts.{i}.mount_type="{func_mount["mount_type"]}"'
                    f' --mount_config.func_mounts.{i}.local_mount_path="{func_mount["local_mount_path"]}"'
            i = i + 1

    return update_cmd


def exec_cmd(cmd):
    proc = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE,
                            stderr=subprocess.STDOUT)
    outs, _ = proc.communicate()
    return outs.decode('UTF-8')


def check_result(stage, exec_result):
    if "USE_ERROR" in exec_result:
        error_info = f"failed to {stage}: {exec_result}"
        logging.error(error_info)
        raise Exception(error_info)

    if "FSS.0409" in exec_result:
        error_info = f"failed to {stage}: {exec_result}"
        logging.error(error_info)
        # Return an error if the function code has no changes to update.
        return
```

```
    try:
        result_object = json.loads(exec_result)
    except Exception:
        error_info = f"failed to {stage}: {exec_result}"
        logging.error(error_info)
        raise Exception(error_info)

    if "error_code" in result_object:
        error_message = result_object["error_msg"]
        error_info = f"failed to {stage}: {error_message}"
        logging.error(error_info)
        raise Exception(error_info)


def generate_update_function_code_cmd():
    cmd =
        f'hcloud FunctionGraph UpdateFunctionCode --cli-region="{region}"'
        f' --function_urn="{function_urn}" --project_id="{project_id}"'
        f' --code_url="{code_url}" --func_code.link="" --func_code.file="" --code_type="obs" '

    depend_list = old_function_code.get("depend_list", None)
    if depend_list is not None and len(depend_list) > 0:
        i = 1
        for depend_id in depend_list:
            cmd = cmd + f'--depend_list.{i}="{depend_id}"'

    return cmd


if __name__ == '__main__':
    deploy_function_path = sys.argv[1]
    key = sys.argv[2]
    f = open(os.path.join(deploy_function_path, "cam.yaml"))
    data = load(f, Loader=Loader)
    function_config = data['components'][0]
    function_name = function_config['name']
    function_properties = function_config['properties']
    region = function_properties['region']
    code_url = function_properties['codeUri']
    project_id = function_properties['projectID']
    # Obtain the function URN.
    function_urn = "urn:fss:" + region + ":" + project_id +
                   ":function:default:" + function_name + ":latest"
    logging.info(f"start to deploy functionURN:{function_urn}")

    # Query the function configuration.
    query_function_config_cmd =
        f'hcloud FunctionGraph ShowFunctionConfig --cli-region="{region}"'
        f' --function_urn="{function_urn}" --project_id="{project_id}"'
    result = exec_cmd(query_function_config_cmd)
    # Check whether a VPC and an agency have been configured for the function. If yes, they must be
included during function updates.
    old_function_config = json.loads(result)
    check_result("query function config", result)

    # Query the function code. Keep this part if a dependency is bound to the function.
    query_function_code_cmd =
        f'hcloud FunctionGraph ShowFunctionCode --cli-region="{region}"'
        f' --function_urn="{function_urn}" --project_id="{project_id}"'
    result = exec_cmd(query_function_code_cmd)
    old_function_code = json.loads(result)
    logging.info("query function %s code result: %s", function_urn, result)
    check_result("query function code", result)

    # Update the function code.
    query_function_code_cmd = generate_update_function_code_cmd()
    result = exec_cmd(query_function_code_cmd)
    logging.info("update function %s code result: %s", function_urn, result)
    check_result("update function code", result)
```



```
  # Update the function configuration.
  update_function_config_cmd = generate_update_function_config_cmd(
      function_properties, old_function_config, key)
  result = exec_cmd(update_function_config_cmd)
  logging.info("update function %s config result: %s", function_urn, result)
  check_result("update function config", result)

  logging.info(f"succeed to deploy function {function_urn}")
```

10.4 Analyzing cam.yaml

Example

```
metadata:
  description: This is an example application for FunctionGraph.
  author: Serverless team
  homePageUrl: https://www.huaweicloud.com/product/functiongraph.html
  version: 1.0.0
components:
  - name: helloworld
    type: Huawei::FunctionGraph::Function
    properties:
      region: cn-east-4
      codeUri: https://test-wkx.obs.cn-north-4.myhuaweicloud.com/helloworld.zip
      projectID: 0531e14952000f742f3ec0088c4b25cf
      handler: index.handler
      runtime: Python3.9
      memorySize: 256
      timeout: 60
      userData:
        key1: value1
        key2: value2
      encryptedUserData: '{"nonce": "ZEUOREFaiahRbMz+K9xQwA==", "header": "aGVhZGVy", "ciphertext":
"SCxXsffvpU1BF2Ci8a2RedNQ", "tag": "a+EYRVPOsQ+YpQkMuFg1wA=="}'
      initializerTimeout: 30
      initializerHandler: index.init_handler
      strategyConfig:
        concurrency: 80
        concurrentNum: 20
```

Parameter Description

Function configuration is included in **properties** of the **cam.yaml** file. The following table details the function configuration.

Parameter	Mandatory	Can Be Updated	Description
region	Yes	No	Region where the function is located.
codeUri	Yes	No	Location of the function code. It is an OBS address where the function code package is stored.
projectID	Yes	No	Project ID.
handler	Yes	Yes	Function handler.

Parameter	Mandatory	Can Be Updated	Description
runtime	Yes	No	Execution environment. Options: Python 2.7 Python 3.6 PHP 7.3 Java 8 Node.js 6.10 Node.js 8.10 C# (.NET Core 2.0) C# (.NET Core 2.1) C# (.NET Core 3.1) Custom
memorySize	Yes	Yes	Memory size. Unit: MB. Enumerated values: 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2560, 3072, 3584, 4096
timeout	Yes	Yes	Timeout. Value range: 3s to 900s.
userData	No	Yes	Name/value information defined for the function.
encryptedUserData	No	Yes	Name/value information to be encrypted.
initializerTimeout	No	Yes	Initialization timeout. Value range: 1s to 300s.
initializerHandler	No	Yes	Function initializer. It must be in the format of "xx.xx". For example, if the function file name is **myfunction.js** and the initializer function is **initializer**, the initializer is **myfunction.initializer**.
concurrentNum	No	Yes	Maximum number of concurrent requests per instance.
concurrency	No	Yes	Maximum number of instances per function. Value **0** indicates that a function is disabled, and value **–1** indicates that there is no instance limit. For example, the value **100** means that a function can have a maximum 100 instances, including common and reserved instances.

NOTE

1. Currently, the **cam.yaml** file does not support the update of VPC, agency, file system, and dynamic memory settings. If a function uses a VPC, agency, file system, or dynamic memory settings, configure them on the function details page. These settings will be kept when executing the function update pipeline.
2. To avoid displaying **encryptedUserData** in plaintext in the **cam.yaml** file, the CI/CD process encrypts its value using AES with Galois/Counter Mode (GCM). The ciphertext is the value of **encryptedUserData** in the **cam.yaml** file. This value is transferred in ciphertext in both the **functions** repository and the function update pipeline. It is decrypted and updated when the function is deployed. Therefore, the key for AES encryption must be provided when the function update pipeline is executed. Example:

   Value of **encryptedUserData** in plaintext:

   ```
   '{"password":"123"}'
   ```

   After AES-GCM encryption:

   ```
   {"nonce": "ZEUOREFaiahRbMz+K9xQwA==", "header": "aGVhZGVy", "ciphertext": "SCxXsffvpU1BF2Ci8a2RedNQ", "tag": "a+EYRVPOsQ+YpQkMuFg1wA=="} ciphertext is the encrypted value.
   ```

Keep the key for AES encryption properly.

Python AES-GCM example: **https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html?highlight=GCM#gcm-mode**

The AES-GCM encryption script is as follows:

```
import json
from base64 import b64encode
from Crypto.Cipher import AES
import sys

if __name__ == '__main__':
    key = sys.argv[1].encode()
    data = sys.argv[2].encode()
    header = b"header"
    cipher = AES.new(key, AES.MODE_GCM)
    cipher.update(header)
    ciphertext, tag = cipher.encrypt_and_digest(data)
    json_k = ['nonce', 'header', 'ciphertext', 'tag']
    json_v = [b64encode(x).decode('utf-8') for x in
              [cipher.nonce, header, ciphertext, tag]]
    result = json.dumps(dict(zip(json_k, json_v)))
    print(result)
```

To execute the script, run the following command on an ECS:

python3 aes_gcm_encrypt_tool.py *"16-byte key"* **'{"password":"123"}'**

