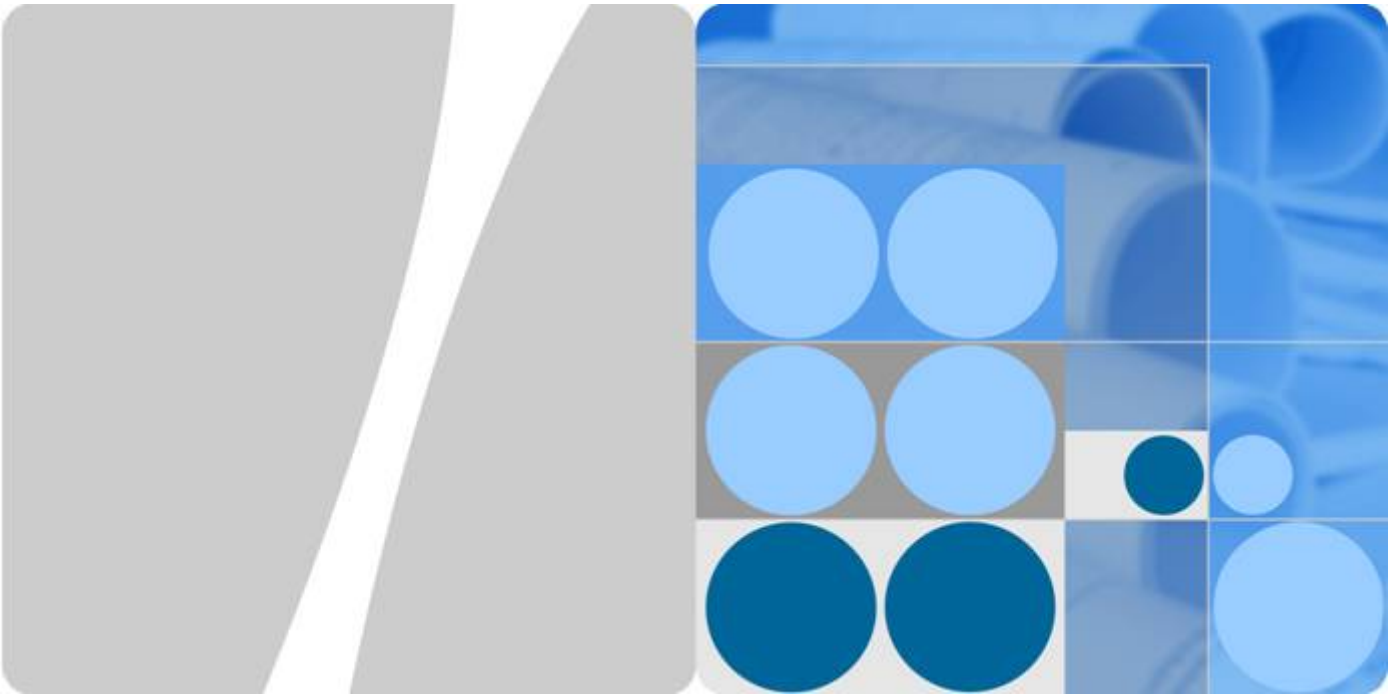


FunctionGraph

Developer Guide

Date 2023-09-30

Contents

1 Overview 1
1.1 Function Development 1
1.2 Supported Event Sources 3
1.3 Function Project Packaging Rules 12
1.4 Referencing DLLs in Functions 17
2 Initializer 19
3 Node.js 21
3.1 Developing an Event Function 21
3.2 Creating a Dependency 25
4 Python 26
4.1 Developing an Event Function 26
4.2 Creating a Dependency 29
5 Java 31
5.1 Developing an Event Function 31
5.1.1 Developing Functions in Java (Using Eclipse) 31
5.2 Creating a Dependency 35
6 Go 36
6.1 Developing an Event Function 36
7 C# 40
7.1 Developing an Event Function 40
7.1.1 C# Function Development 40
8 PHP 45
8.1 Developing an Event Function 45
9 Development Tools 49
9.1 Eclipse Plug-in 49
9.2 PyCharm Plug-in 52

1 Overview

1.1 Function Development

Supported Runtimes

The Node.js, Java, Python, Go, C#, PHP, 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, 16.17, 18.15	-
Python	2.7, 3.6, 3.9, 3.10	-
Java	8, 11	**Java SDK** **NOTE** The Java runtime has integrated with Object Storage Service (OBS) SDKs.
Go	1.8, 1.x	**Go1.8.3 SDK**
C#	.NET Core 2.0, .NET Core 2.1, .NET Core 3.1	**C# SDK**
PHP	7.3	-
Custom	-	-

Third-Party Components Integrated with the Node.js Runtime

Table 1-2 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 1-3 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

Library	Usage	Version
redis	Redis client	2.10.5
obsclient	OBS client	-
smnsdk	Simple Message Notification (SMN) access	1.0.1

Sample Project Packages

Table 1-4 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 1-4 Download links of the sample project packages

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

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.

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**.

OBS

Object Storage Service (OBS) is a stable, secure, efficient, and easy-to-use cloud storage service. You can create a function to process OBS bucket events, for example, creating and deleting objects. When an image is uploaded to a specified bucket, OBS invokes the function to read the image and create a thumbnail. For details, see **Using an OBS Trigger**.

Table 1-5 Event types supported by OBS

Event	Description
ObjectCreated	All kinds of object creation operations, including PUT, POST, and COPY of objects, as well as the merging of parts.
Put	Use the PUT method to upload objects.
Post	Use the POST method to upload objects.
Copy	Use the COPY method to replicate objects.
CompleteMultipartUpload	Merge parts of multi-part tasks.
ObjectRemoved	Delete objects.
Delete	Delete objects by versions.
DeleteMarkerCreated	Delete objects without specifying versions.

NOTE

Multiple event types can be used on the same object. For example, if you have selected **Put**, **Copy**, and **Delete** in an event notification rule, a notification message will be sent to you when the specified object is uploaded to, copied to, or deleted from the bucket. **ObjectCreated** contains **Put**, **Post**, **Copy**, and **CompleteMultipartUpload**. If you select **ObjectCreated**, the others are automatically selected and cannot be selected again. Similarly, if you select **ObjectRemoved**, **Delete** and **DeleteMarkerCreated** are automatically selected and cannot be selected again.

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**.

LTS

Log Tank Service (LTS) collects and stores logs, allowing you to query them in real time. If you create a function with an LTS trigger, subscribed logs collected by LTS will be passed as a parameter (**LTS example event**) to invoke the function. Then, the function processes or analyzes the logs, or loads the logs to other systems. For details, see **Using an LTS Trigger**.

CTS

Cloud Trace Service (CTS) collects operation records of subscribed cloud resources. If you create a function with a CTS trigger, collected operation records of specified cloud services will be passed as a parameter (**CTS example event**) to invoke the function. Then, the function analyzes and processes key information in the operation records, automatically recovers system or network modules, or reports alarms to service personnel by SMS or email. **Using a CTS 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**.

Example Events

- OBS example event

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

               "name": "functionstorage-template",
               "ownerIdentity": {
                  "PrincipalId": "0ed1b73473f24134a478962e631651eb"
               }
            }
         },
         "Region": "{region}",
         "eventName": "ObjectCreated:Post",
         "userIdentity": {
```

```
            "principalId": "9bf43789b1ff4b679040f35cc4f0dc05"
        }
      }
   ]
}
```

Table 1-6 Parameter description

Parameter	Type	Example Value	Description
eventVersion	String	2.0	Event version
eventTime	String	2018-01-09T07:50:50.028Z	Time when an event occurs. The ISO-8601 time format is used.
sourceIPAddress	String	103.218.216.125	Source IP address
s3	Map	See the example.	OBS event content
object	Map	See the example.	object parameter description
bucket	Map	See the example.	bucket parameter description
ownerIdentity	Map	See the example.	ID of the user who creates the bucket
Region	String	ap-southeast-4	Region where the bucket is located
eventName	String	ObjectCreated:Post	Event name
userIdentity	Map	See the example.	ID of the account that initiates the request

- 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
}
```

NOTE

- 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 1-7 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
queryStringPara-meters	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

Parameter	Type	Example Value	Description
headers	Map	See the example.	Complete headers
path	String	/apig-event-template	Complete path
isBase64Encoded	Boolean	True	Default value: **true**

- 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 1-8 Parameter description

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

- LTS example event

```
{
    "lts": {
        "data":
"lCB7CiAgICAibG9ncyI6W3sKICAgICAgICAgIm1lc3NhZ2UiOiIyMDE4LTA4LTA4LzA4OjA4OjA4IFtXUk5dIFt0ZXN0LmdvOjA4XVRoaXMgaXMgYSB0ZXN0IG1lc3NhZ2UuIiwKICAgICAgICAgInRpbWUiOjE1MzAwMDk2NTMwNTksCiAgICAgICAgICJob3N0X25hbWUiOiJlY3MtdGVzdCIsCiAgICAgICAgICJpcCI6IjE5Mi4xNjguMS4xIiwKICAgICAgICAgInBhdGgiOiJ2YXIvbG9nL3Rlc3QubG9nIiwKICAgICAgICAgImxvZ191aWQiOiI2NjNkNjkzMC03OTJkLTExZTgtOGIwOC0yODZlZDQ4OGNlNzAiLAogICAgICAgICAibGluZV9ubyI6MQogICAgICAgICAgImxldmVsIjogIjwzPiIKICAgICAgICAgImxvZ19ncm91cF9pZCI6ImI4NTFjZjVjYTA1MTI5IiwKICAgICAgICAgIlN0cmVhbV9pZCI6ImI4NTdhOWQyODQtNDQ0OC0xMWU4LThmYTQtMjg2ZWQ0ODhjZTcwIiwKICAgICAgICAgIlN0cmVhbV9uYW1lIjoiMWE5Njc1YTctNzg0ZC0xMWU4LTlmNzAtMjg2ZWQ0ODhjZTcwIgogfQ=="
    }
}
```

Table 1-9 Event parameter description

Parameter	Type	Example Value	Description
data	Sting	See the example.	Base64-encoded data

- CTS example event

```
{
   "cts": {
      "time": "2018/06/26 08:54:07 GMT+08:00",
      "user": {
         "name": "userName",
         "id": "5b726c4fbfd84821ba866bafaaf56aax",
         "domain": {
            "name": "domainName",
            "id": "b2b3853af40448fcb9e40dxj89505ba"
         }
      },
      "request": {},
      "response": {},
      "code": 204,
      "service_type": "vpc",
      "resource_type": "VPC",
      "resource_name": "workflow-2be1",
      "resource_id": "urn:fgs:{region}:2d1d891d93054bbaa69b9e866c0971ac:graph:workflow-2be1",
      "trace_name": "deleteGraph",
      "trace_type": "ConsoleAction",
      "record_time": "2018/06/26 08:54:07 GMT+08:00",
      "trace_id": "69be64a7-0233-11e8-82e4-e5d37911193e",
      "trace_status": "normal"
   }
}
```

Table 1-10 Parameter description

Parameter	Type	Example Value	Description
User	Map	See the example.	Information about the user who initiates the request
Request	Map	See the example.	Event request
Response	Map	See the example.	Event response
Code	Int	204	Response code, for example, **200** and **400**
service_type	String	vpc	Abbreviation of the sender, for example, **vpc** and **ecs**
resource_type	String	VPC	Resource type of the sender, for example, **vm** and **vpn**

Parameter	Type	Example Value	Description
resource_name	String	workflow-2be1	Resource name, for example, the name of an ECS
trace_name	String	deleteGraph	Event name (trace name), for example, **startServer** and **shutDown**
trace_type	String	ConsoleAction	Event type (trace type), for example, **ApiCall**
record_time	string	2018/06/26 08:54:07 GMT +08:00	Time when the CTS service receives the trace
trace_id	String	69be64a7-0233-11e8-82e4-e5d37911193e	Event ID (trace ID)
trace_status	String	normal	Event status (trace status)

- 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

Parameter	Type	Example Value	Description
region	String		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

- 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 1-12 Parameter description

Parameter	Type	Example Value	Description
event_version	String	v1.0	Event version
Region	String		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 1-13** describes the rules for packaging a function project.

Table 1-13 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 1.8	Not supported.	Compress project files into a ZIP file, and ensure that the name of the dynamic library file is consistent with the handler plugin name. For example, if the name of the dynamic library file is **testplugin.so**, set the handler plugin name to **testplugin.Handler**. **Handler** indicates the function handler.	Compress project files into a ZIP file and upload it to an OBS bucket.
Go 1.x	Not supported.	Zip the compiled file and ensure that the name of the binary file is consistent with that of the handler. For example, if the name of the binary file is **Handler**, set the name of the handler to **Handler**.	Compress project files into a ZIP file and upload it to an OBS bucket.
C#	Not supported.	Compress project files into a ZIP file. The ZIP file must contain the following files: *Project_name*.**deps.json**, *Project_name*.**dll**, *Project_name*.**runtimeconfig.json**, *Project_name*.**pdb**, and **HC.Serverless.Function.Common.dll**.	Compress project files into a ZIP file and upload it to an OBS bucket.
Custom	Not supported.	Compress project files into a ZIP file. The ZIP file must contain a bootstrap file.	Compress project files into a ZIP file and upload it to an OBS bucket.

Example ZIP Project Packages

- Example directory of a Nods.js project package

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

- Example directory of a PHP project package

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

- Example directory of a Python project package

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

- Example directory of a Java project package

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

- Example directory of a Go project package

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

- Example directory of a C# project package

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

- 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.

- 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, and Node.js 14.18

  Node.js 8.10, Node.js 10.16, Node.js 12.13, and Node.js 14.18 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.

Table 3-1 Context methods

Method	Description
getRequestID()	Obtains a request ID.
getRemainingTimeInMilliSeconds ()	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. **NOTE** 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. **NOTE** 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.
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.

Method	Description
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.

WARNING

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

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 Creating a Dependency

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

NOTE

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

Creating a Dependency for a 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, and Python 3.9.

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)

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.

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. **NOTE** 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. **NOTE** 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.

WARNING

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

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 Creating a Dependency

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

NOTE

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

Creating a Dependency for a Python Function

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

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

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

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

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

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

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

The required dependency is generated.

NOTE

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

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

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]*.

Java Initializer

FunctionGraph supports the following Java runtime:

- Java 8 (runtime = Java8)

  Initializer syntax:

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

For example, if the initializer is named **com.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com** 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.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com** 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-sha256) provides context and logging APIs.

- 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 5-1 describes the context APIs provided by FunctionGraph.

  Table 5-1 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. **NOTE** FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.

Method	Description
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. **NOTE** 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.
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.

WARNING

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

- Logging API

  Table 5-2 describes the logging API provided in the Java SDK.

  Table 5-2 Logging API

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

Execution Result

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

Table 5-3 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 **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.
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.

5.2 Creating a Dependency

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

NOTE

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

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

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**.

SDK APIs

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

- 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 6-1 describes the context APIs provided by FunctionGraph.

  Table 6-1 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. **NOTE** 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. **NOTE** 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.

WARNING

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

- Table 6-2 describes the logging API provided in the Go SDK.

Table 6-2 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**.

Execution Result

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

Table 6-3 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 C#

7.1 Developing an Event Function

7.1.1 C# Function Development

Function Syntax

NOTE

You are advised to use .NET Core 3.1.

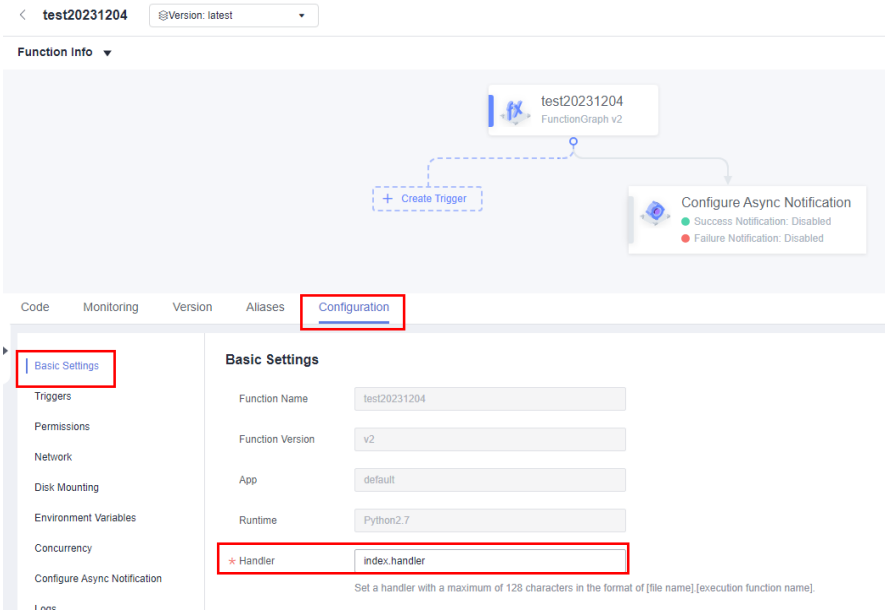
FunctionGraph supports C# (.NET Core 2.0), C# (.NET Core 2.1), and C# (.NET Core 3.1).

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.

  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 7-1**. The parameter value is in the format of **index.handler**. The values of **index** and **handler** can be customized.

Figure 7-1 Handler parameter



- 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
}
```

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. **NOTE** FunctionGraph has stopped maintaining the String AccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.
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. **NOTE** 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 SecuritySecretToken	Obtains the SecuritySecretToken (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 RemainingTimeInMilliSeconds	Remaining running time of a function.

Attribute	Description
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);

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.

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 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.

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. **NOTE** 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. **NOTE** 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.

Method	Description
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.

NOTE

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

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.
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.

9 Development Tools

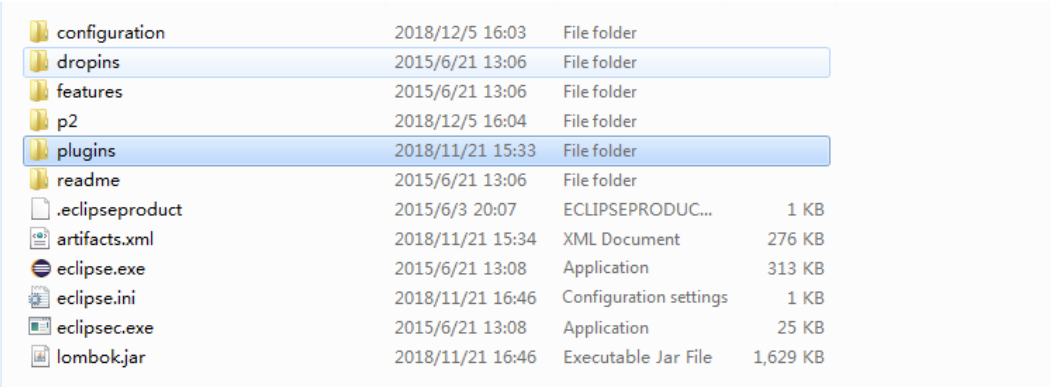
9.1 Eclipse Plug-in

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**).

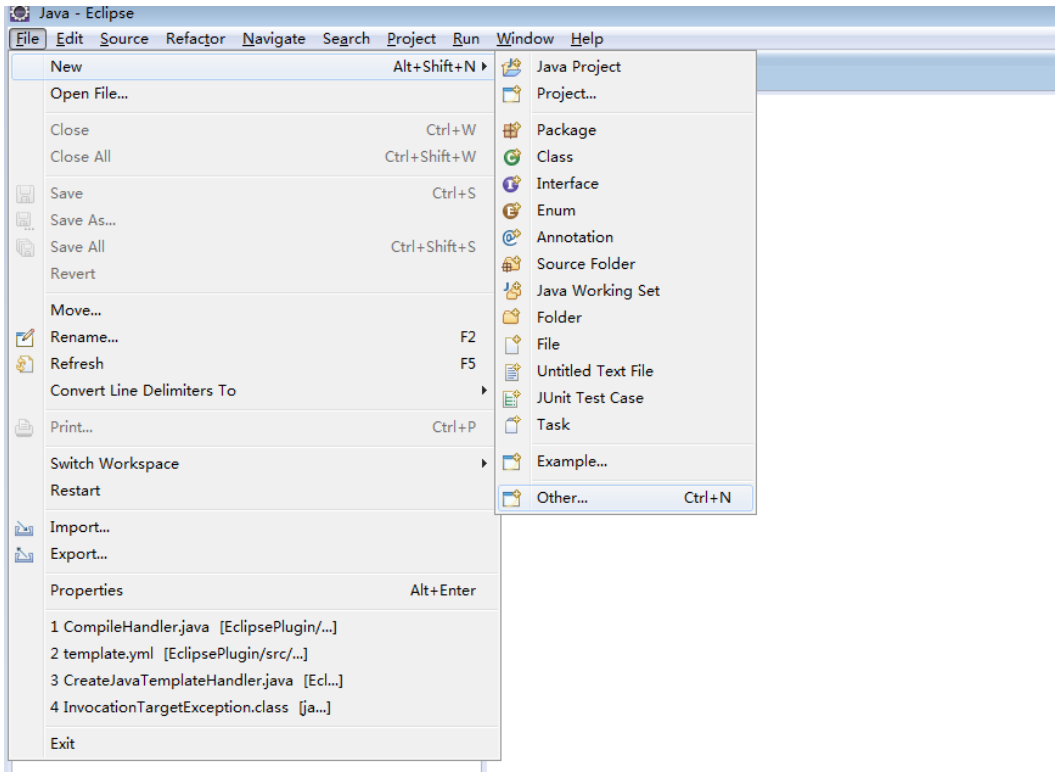
Step 2 Put the Eclipse plug-in package (.jar or .zip) in the **plugins** folder under the Eclipse installation directory. Then restart Eclipse. **Figure 9-1** shows the Eclipse installation directory.

Figure 9-1 Installing the Eclipse plug-in



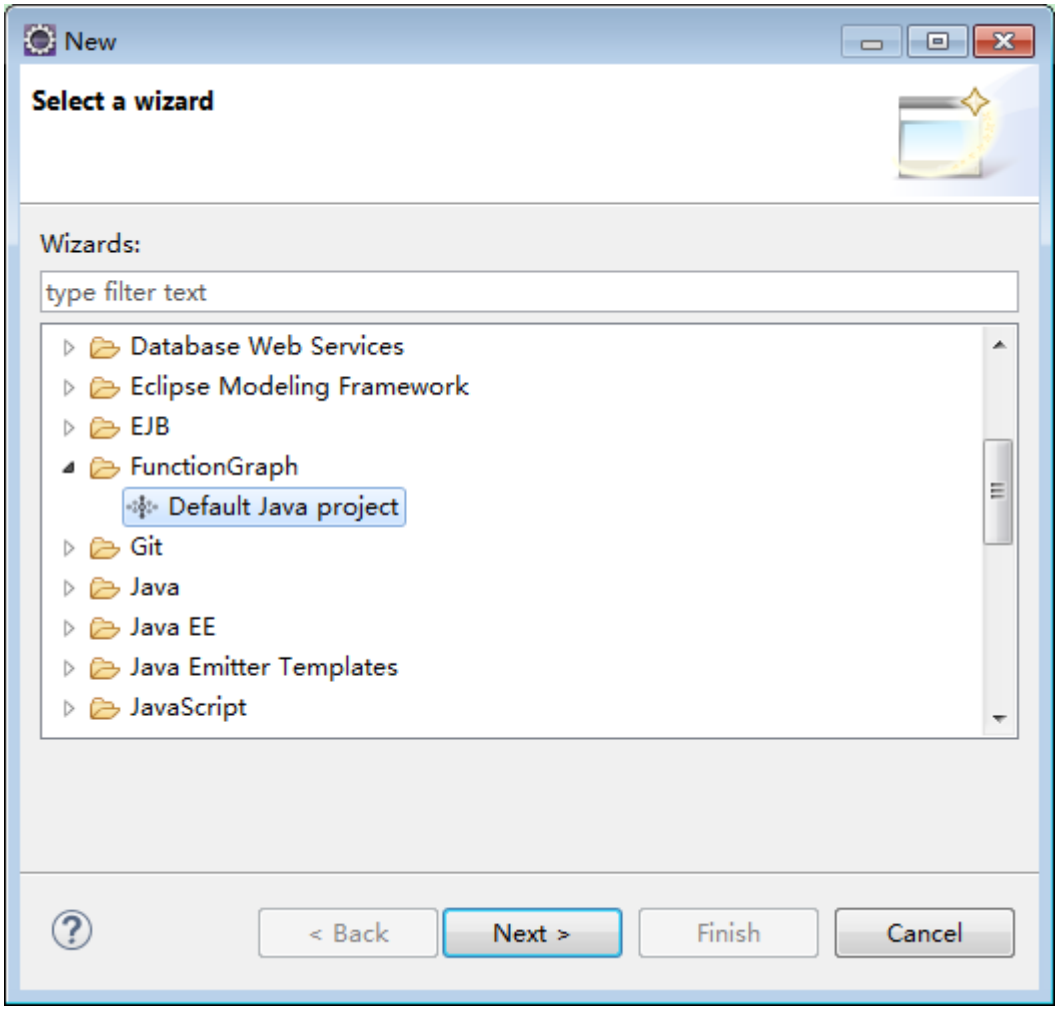
Step 3 Open Eclipse and choose **File** > **New** > **Other**, as shown in **Figure 9-2**.

Figure 9-2 Creating a template



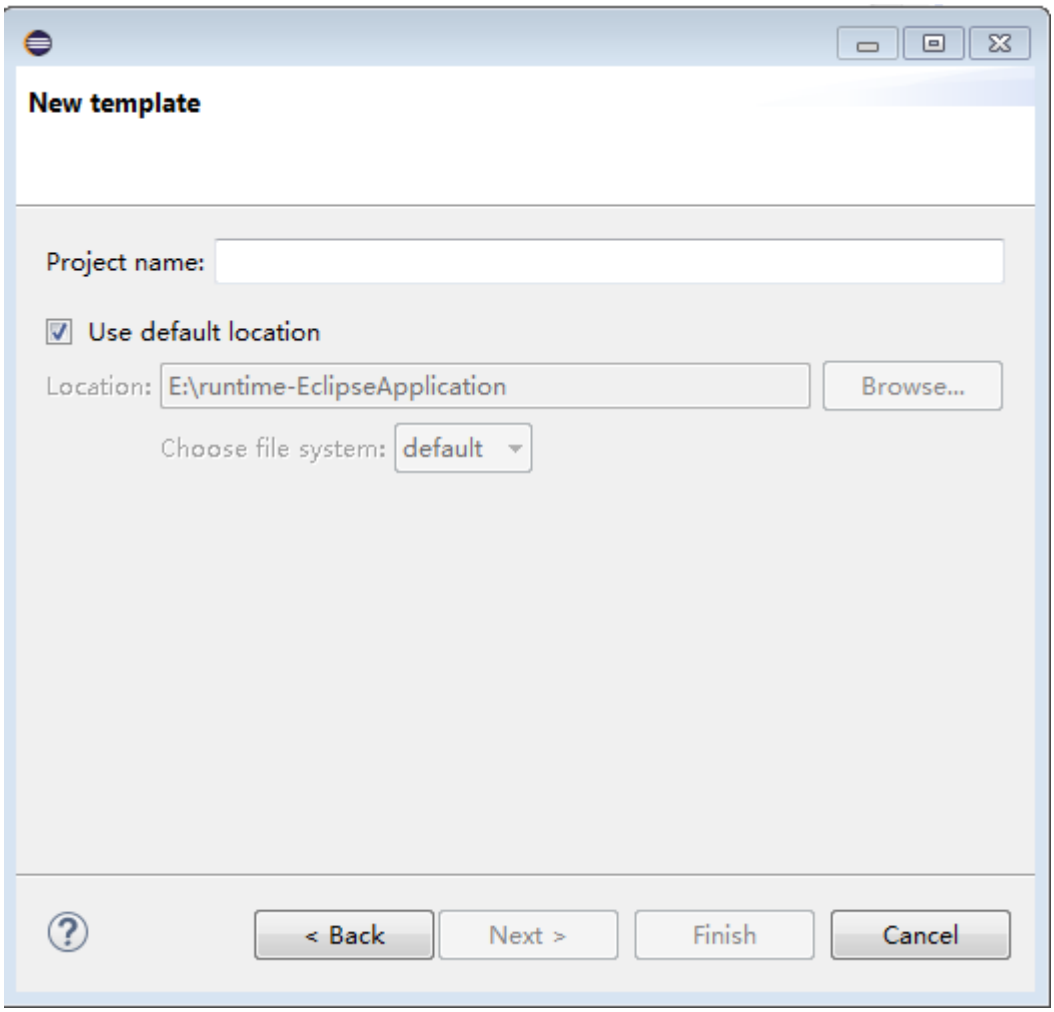
Step 4 Choose **FunctionGraph** > **Default Java project**, as shown in **Figure 9-3**.

Figure 9-3 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 9-4**.

Figure 9-4 Setting the project name and directory



----End

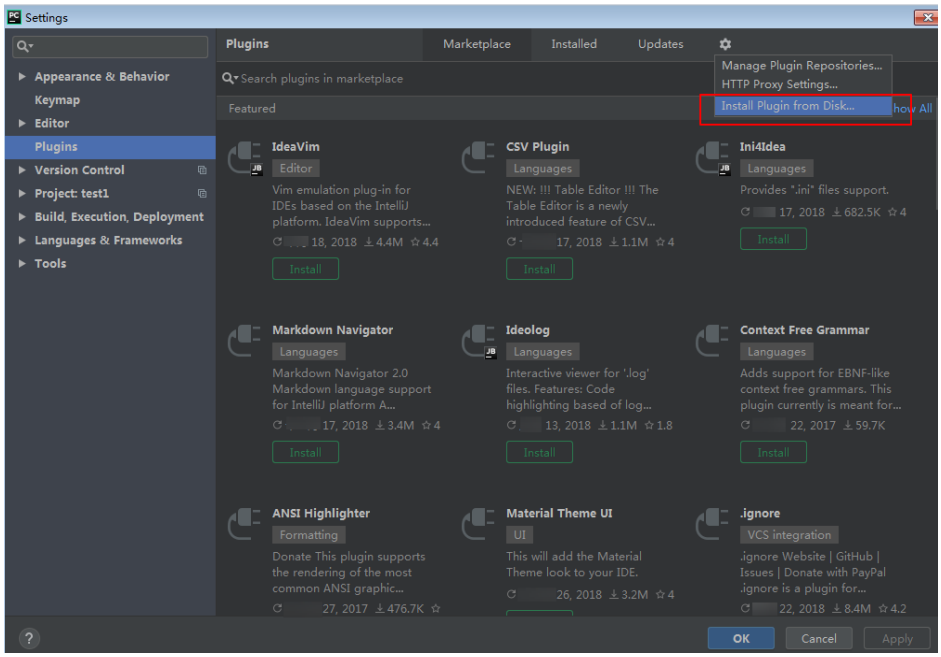
9.2 PyCharm Plug-in

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

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

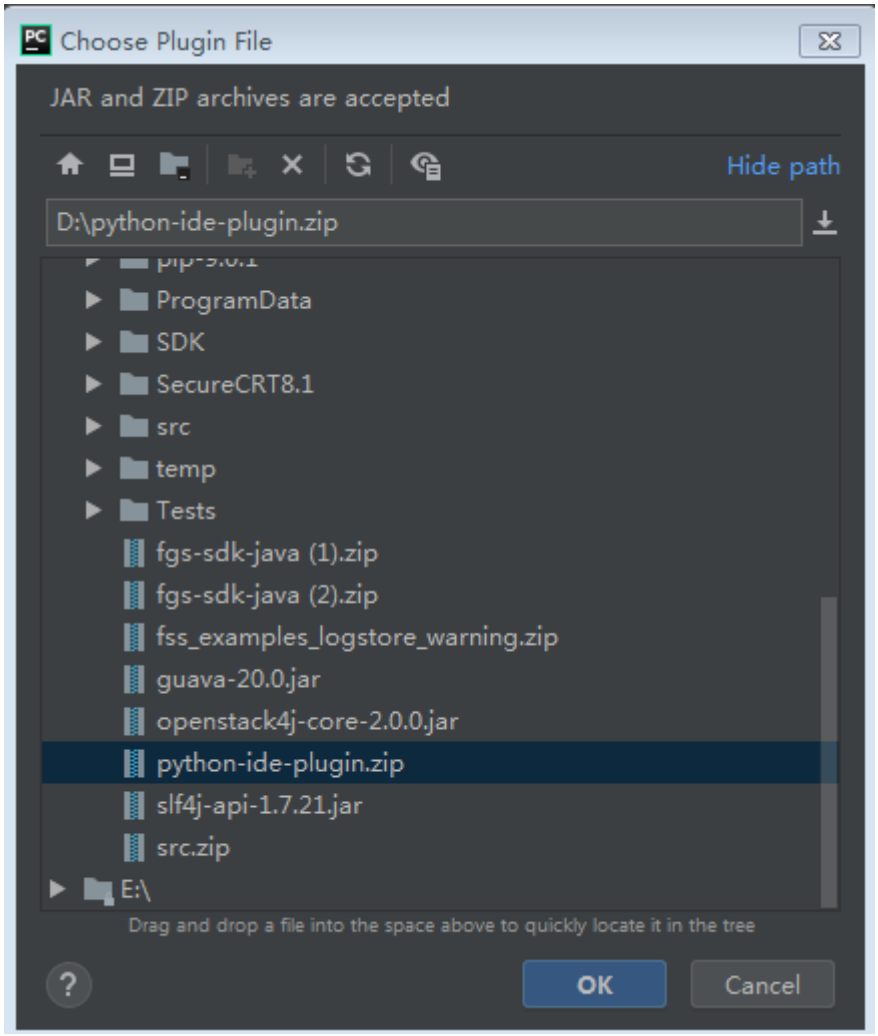
Step 2 Run JetBrains PyCharm. Choose **File** > **Settings**, choose **Plugins** in the left pane, and then click **Install Plugin from Disk** in the upper right corner, as shown in Figure 9-5.

Figure 9-5 Installing the plug-in



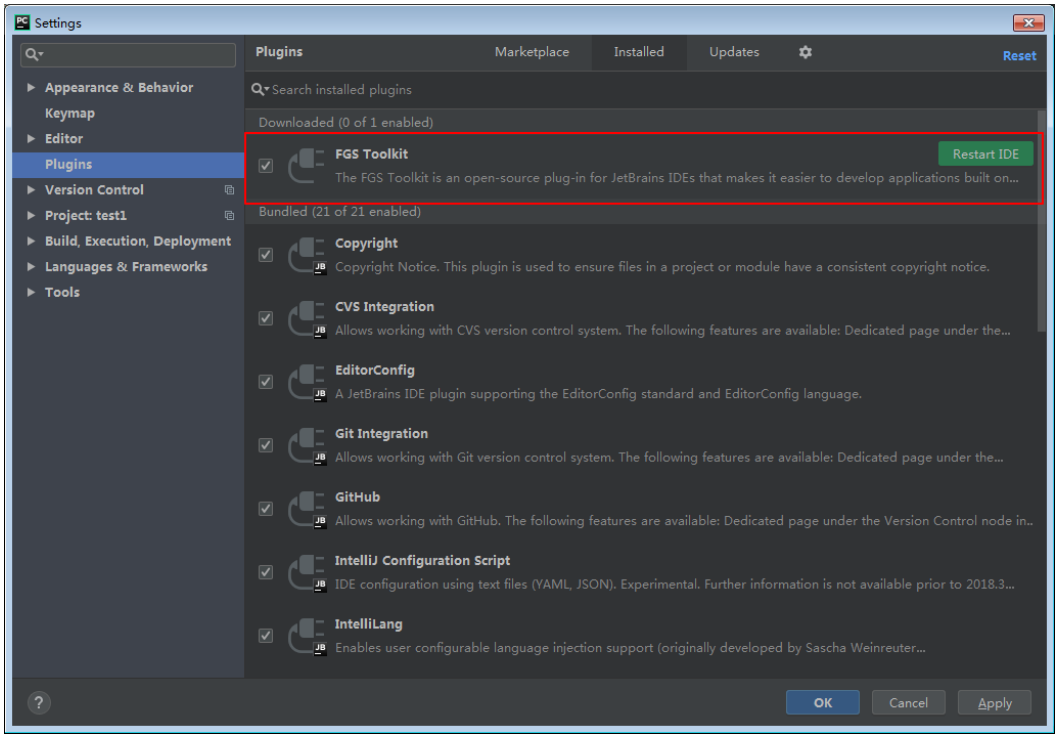
Step 3 Select the plug-in package you want to install, and click **OK**, as shown in **Figure 9-6**.

Figure 9-6 Selecting a plug-in package



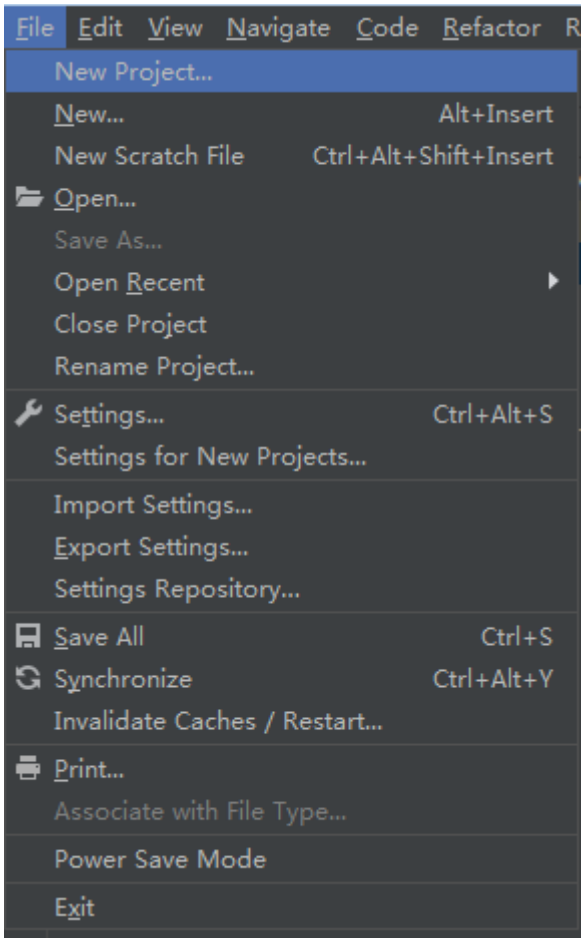
Step 4 In the plug-in list, select the desired plug-in and click **Restart IDE**, as shown in **Figure 9-7**.

Figure 9-7 Restarting the IDE



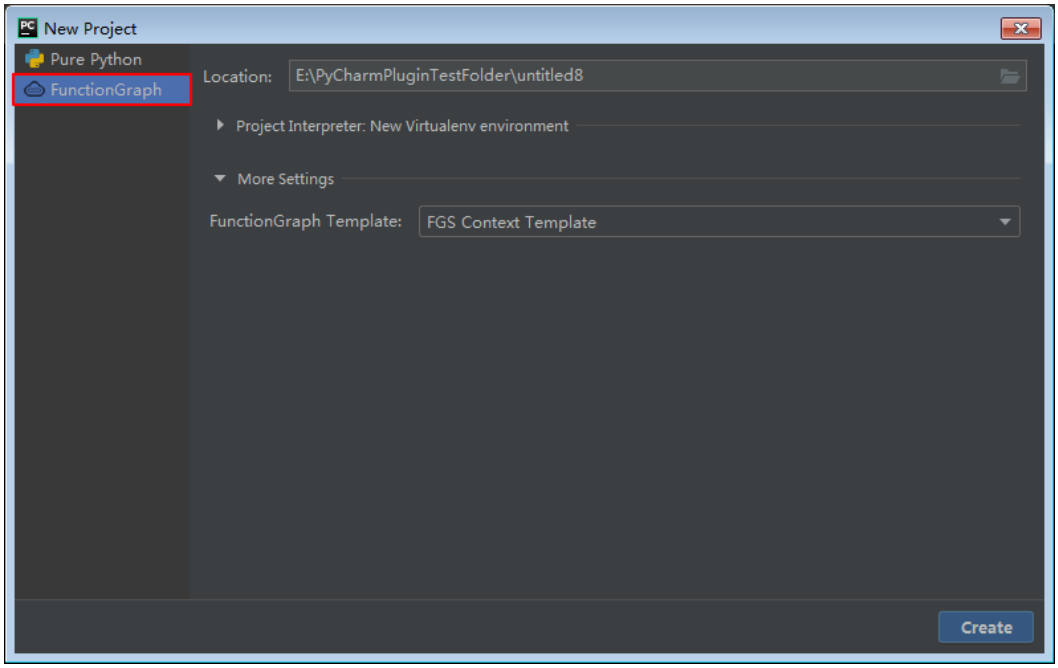
Step 5 Choose **File** > **New Project**, as shown in **Figure 9-8**.

Figure 9-8 Creating a project



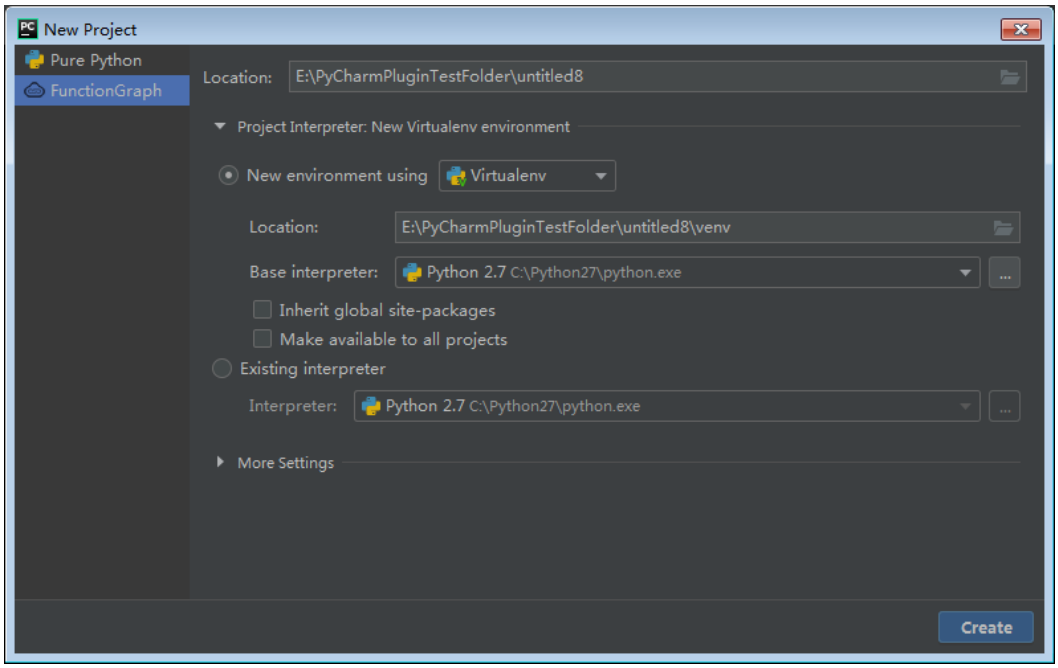
Step 6 On the displayed **New Project** page, choose **FunctionGraph**, as shown in **Figure 9-9**.

Figure 9-9 FunctionGraph



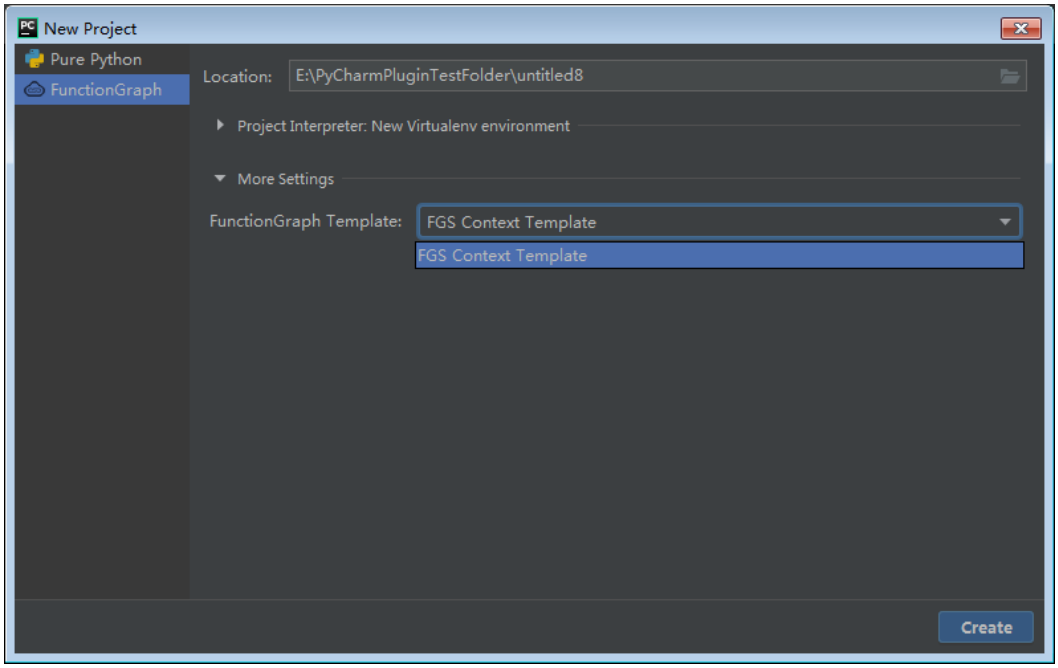
Step 7 Select the path in which the project will be stored in **Location**, and select a Python version in **Base interpreter**, as shown in **Figure 9-10**.

Figure 9-10 Selecting a version



Step 8 Select a template you want to create in the **More Settings** area, as shown in **Figure 9-11**.

Figure 9-11 Selecting a template



NOTE

Currently, only the Python 2.7 context template is supported.

Step 9 Click **Create**.

----End