FunctionGraph

Getting Started

 Issue
 01

 Date
 2024-10-23





Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions

NUAWEI 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.

Contents

1 Creating a Function from Scratch and Executing the Function	1
2 Creating a Function Using a Template and Executing the Function	5
3 Creating an HTTP Function Using a Container Image and Executing the Function	tion 9
4 Creating an Event Function Using a Container Image and Executing the Func	tion 15

Creating a Function from Scratch and Executing the Function

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

Prerequisites

 Register with Huawei Cloud and complete real-name authentication.
 For details, see Registering a HUAWEI ID and Enabling Huawei Cloud Services and Real-Name Authentication.

If you already have a Huawei account and have completed real-name authentication, skip this step.

2. View free quota.

FunctionGraph offers a free tier every month, which you can share with your IAM users. For details, see **Free Tier**

If you continue to use FunctionGraph after the free quota is used up, your account goes into arrears if the balance is less than the bill to be settled. To continue using your resources, top up your account in time..

3. Grant the FunctionGraph operation permissions to the user.

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

Step 1: Create a Function

- Log in to the FunctionGraph console. In the navigation pane, choose Functions > Function List.
- 2. Click **Create Function** in the upper right corner and choose **Create from scratch**.
- 3. On the displayed page, enter **HelloWorld** for **Function Name** and retain the default values for other parameters by referring to **Figure 1-1**, and click **Create Function**. The following describes the parameters.
 - Function Type: Select Event Function.
 - **Region**: The default value is used. You can select other regions.

D NOTE

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

- **Project**: The default value is the same as the selected region.
- Function Name: enter HelloWorld.
- **Enterprise Project**: The default value is **default**. You can select the created enterprise project.

NOTE

Enterprise projects let you manage cloud resources and users by project.

- **Agency**: By default, no agency is used. You can select an existing agency.

D NOTE

Specify an agency if you want to delegate FunctionGraph to access other cloud services, such as LTS and VPC.

Runtime: Select a runtime to compile the function. Default: Node.js
 16.17. You can select another runtime.

Figure 1-1 Configuring basic information

Basic Information
* Function Type 🔞
Event Function HTTP Function
Processes event requests and can be triggered by APIG, OBS, and DIS events.
• •
Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.
* Function Name
HelloWorld
Enter 1 to 60 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Agency (2)
Use no agency C Create Agency
* Enterprise Project (2)
default C View Enterprise Project
Runtime ⑦
Node js 10.16 🔹

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

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

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

Step 2: Test the Function

{

}

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

```
"hello": "function"
```

Figure 1-2 Configuring a test event

Event Templates (18)		* Event	Name	test					
Search	Q	1	{	"helle"		at lan"			
Simple Message Notification (SMN)		3	}	nello	: Tu	ICCION			
Timer									
DMS (for Kafka)									
Kafka (OPENSOURCEKAFKA)									
DMS (for RabbitMQ)									
ommon Event									
Blank Template									
Login Security Analysis									
Image Classification									
Pornographic Image Analysis									
Speech Recognition									

Step 3: View the Execution Result

Click Test and view the execution result on the right.

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

Figure 1-3 Viewing the execution result

test	Test Deploy	
1631	<pre>v voor voor voor voor voor voor voor voo</pre>	 Execution Result X Calculation Result X C

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

Related Information

- For details about FunctionGraph concepts, see **Concepts**.
- For details about FunctionGraph pricing, see **FunctionGraph Pricing Details**.
- For details about the constraints and limitations of FunctionGraph, see Notes and Constraints.

2 Creating a Function Using a Template and Executing the Function

FunctionGraph provides multiple templates to automatically complete code and running environment configurations when you create a function, helping you quickly build applications. This section uses **context-class-introduction** as an example.

Prerequisites

1. Register with Huawei Cloud and complete real-name authentication.

For details, see **Registering a HUAWEI ID and Enabling Huawei Cloud Services** and **Real-Name Authentication**.

If you already have a Huawei account and have completed real-name authentication, skip this step.

2. View free quota.

FunctionGraph offers a free tier every month, which you can share with your IAM users. For details, see **Free Tier**

If you continue to use FunctionGraph after the free quota is used up, your account goes into arrears if the balance is less than the bill to be settled. To continue using your resources, top up your account in time..

3. Grant the FunctionGraph operation permissions to the user.

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

Step 1: Create a Function

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

Figure 2-1 Selecting a template

Scenario	All	Basic fun	ction usage	Data	processing	Data s	synchronizatio	n	File process	ing N	lessage proces	sing Pic	cture process	ing Text rea	cognition	Voice anal	ysis		
Service	Ali VBS	APIG	CTS	DIS	DMS	ECS	EVS	FRS	Functi	onGraph	IMAGE	KAFKA	LTS	Moderation	OBS	OCR	OMS	SIS	SMN
Runtime	All	Node.js 1	2.13	Node.js 6.1	0 Nod	le.js 8.10	PHP 7.3	P	ython 2.7	Python 3	1.6								
Basic function context-cla	n usage ass-intro Python	duction	Config	ure															
Service	ဗုံနှံ Functi	onGraph																	
Obtains a ten View details	nporary cre	edential (such	ı as AK, SK,	, an															

- 4. Set **Function Name** to **context**, select any agency from the **Agency** dropdown list, retain default values for other parameters, and click **Create Function**.
 - Templates: name of the selected template. To change the template, click Reselect on the right.
 - **Region**: The default value is used. You can select other regions.

D NOTE

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

- **Project**: The default value is the same as the selected region.
- Function Name: enter context.
- Enterprise Project: The default value is default. You can select the created enterprise project.

NOTE

Enterprise projects let you manage cloud resources and users by project.

- **Agency**: By default, no agency is used. You can select an existing agency.

D NOTE

Specify an agency if you want to delegate FunctionGraph to access other cloud services, such as LTS and VPC.

Runtime: Select a runtime to compile the function. Default: Python 2.7.
 You can select another runtime.

NOTE

If no agency is configured, the following message will be displayed when the function is triggered:
 Failed to access other services because no temporary AK, SK, or token has been obtained.
 Please set an agency.

Figure 2-2 Setting basic information

Basic Information	
Function Template	
context-class-introduction-python Reselect	
* Region	
۰ ۲	
Regions are geographic areas isolated from each other. Resource access, select the nearest region.	s are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource
* Function Name	
context	
Enter 1 to 60 characters, starting with a letter and ending with a let	tter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Agency 🕜	
agency 🔻	C Create Agency
* Enterprise Project ⑦	
default 💌	C View Enterprise Project
Runtime ③	
Python 2.7 v	

Step 2: Test the Function

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

Figure 2-3 Configuring a test event

Configure Test Event										
Create new test event Edit saved test event										
Event Templates (18)	* Event Name test									
Search Q	1 {	_								
Simple Message Notification (SMN)	3 }									
Timer										
DMS (for Kafka)										
Kafka (OPENSOURCEKAFKA)										
DMS (for RabbitMQ)										
Common Event										
Blank Template										
Login Security Analysis										
Image Classification										
Pornographic Image Analysis										
Speech Recognition										
	Create Cancel									

Step 3: View the Execution Result

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

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

×

D NOTE

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

Related Information

- For details about FunctionGraph concepts, see **Concepts**.
- For details about FunctionGraph pricing, see **FunctionGraph Pricing Details**.
- For details about the constraints and limitations of FunctionGraph, see **Notes** and **Constraints**.

3 Creating an HTTP Function Using a Container Image and Executing the Function

This section uses the creation of an HTTP function using a container image as an example to describe how to create and test a container image function. In this example, implement an HTTP server in the image and listen on **port 8000** for requests. **(Do not change port 8000 in the examples provided in this section.)** HTTP functions support only APIG triggers.

Prerequisites

1. Register with Huawei Cloud and complete real-name authentication.

For details, see **Registering a HUAWEI ID and Enabling Huawei Cloud Services** and **Real-Name Authentication**.

If you already have a Huawei account and have completed real-name authentication, skip this step.

2. View free quota.

FunctionGraph offers a free tier every month, which you can share with your IAM users. For details, see **Free Tier**

If you continue to use FunctionGraph after the free quota is used up, your account goes into arrears if the balance is less than the bill to be settled. To continue using your resources, top up your account in time..

3. Grant the FunctionGraph operation permissions to the user.

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

Step 1: Create an Image

Take the Linux x86 64-bit OS as an example. (No system configuration is required.)

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

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

```
const express = require('express');
const PORT = 8000;
const app = express();
app.use(express.json());
app.post('/*', (req, res) => {
    console.log('receive', req.body);
    res.send('Hello FunctionGraph, method POST');
});
app.listen(PORT, () => {
    console.log(`Listening on http://localhost:${PORT}`);
});
```

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

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

- **name**: project name
- version: project version
- main: application entry file
- dependencies: all available dependencies of the project in npm
- 4. Create a Dockerfile. FROM node:12.10.0

}

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

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

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

RUN cd \${HOME} && npm install

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

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

USER \${USER_NAME} WORKDIR /\${HOME}

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

- **FROM**: Specify base image **node:12.10.0**. The base image is mandatory and its value can be changed.
- ENV: Set environment variables HOME (/home/custom_container), GROUP_NAME and USER_NAME (custom_container), USER_ID and GROUP_ID (1003). These environment variables are mandatory and their values can be changed.
- RUN: Use the format RUN <*Command>*. For example, RUN mkdir -m 550 \${HOME}, which means to create the home directory for user \$ {*USER_NAME*} during container building.
- **USER**: Switch to user *\${USER_NAME}*.
- WORKDIR: Switch the working directory to the /\${HOME} directory of user *\${USER_NAME}*.
- COPY: Copy main.js and package.json to the home directory of user \$
 {USER_NAME} in the container.
- EXPOSE: Expose port 8000 of the container. Do not change this parameter.
- **ENTRYPOINT**: Run the **node main.js** command to start the container. Do not change this parameter.

NOTE

- 1. You can use any base image.
- In the cloud environment, UID 1003 and GID 1003 are used to start the container by default. The two IDs can be modified by choosing Configuration > Basic Settings > Container Image Override on the function details page. They cannot be root or a reserved ID.
- 3. Do not change port 8000 in the example HTTP function.
- 4. If the basic image of the Alpine version is used, run the **addgroup** and **adduser** commands.
- 5. Build an image.

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

docker build -t custom_container_http_example:latest .

Step 2: Perform Local Verification

- 1. Start the Docker container. docker run -u 1003:1003 -p 8000:8000 custom_container_http_example:latest
- 2. Open a new Command Prompt, and send a message through port 8000. You can access all paths in the root directory in the template code. The following uses **helloworld** as an example.

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

The following information is returned based on the module code: Hello FunctionGraph, method POST

3. Check whether the following information is displayed: receive {"message":"HelloWorld"}

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

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

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

Step 3: Upload the Image

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

SWR	My Images 🕥	L Upload Through Client	L Upload Through SWR
Deshiboard My Images	Private Images Shared Images		

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

Step 4: Create a Function

- In the left navigation pane of the management console, choose Compute > FunctionGraph. On the FunctionGraph console, choose Functions > Function List from the navigation pane.
- 2. Click **Create Function** in the upper right corner and choose **Container Image**.
- 3. Set the basic information.
 - Function Type: Select HTTP Function.
 - **Region**: The default value is used. You can select other regions.

D NOTE

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

- **Project**: The default value is the same as the selected region.
- Function Name: Enter custom_container_http.
- **Enterprise Project**: The default value is **default**. You can select the created enterprise project.

D NOTE

Enterprise projects let you manage cloud resources and users by project.

- Agency: Select an agency with the SWR Admin permission. If no agency is available, create one by referring to section "Creating an Agency".
- **Container Image**: Enter the image uploaded to SWR in **step 3**.
- 4. (Optional) Override the container image.
 - CMD: container startup command. Example: /bin/sh. If no command is specified, the entrypoint or CMD in the image configuration will be used.

- **Args**: container startup parameter. Example: **-args,value1**. If no argument is specified, CMD in the image configuration will be used.
- User ID: Enter the user ID.
- **Group ID**: Enter the user group ID.
- 5. After the configuration is complete, click **Create Function**.

Step 5: Test the Function

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

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

Step 6: View the Execution Result

}

{

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

Figure 3-1 Execution result

```
Execution Result ×
 Execution successful
 Function Output
     "body": "SGVsbG8gRnVuY3Rpb25HcmFwaCwgbWV0aG9kIFBPU1Q=",
     "headers": {
         "Content-Length": [
             "32"
         1.
         "Content-Type": [
             "text/html; charset=utf-8"
         Ъ
         "Date": [
             "Wed, 02 Nov 2022 11:06:38 GMT"
         1,
         "Etag": [
             "W/\"20-uygbC2IEf2PxTTMC0H1BL5d/vwI\""
         1,
          "X-Powered-By": [
            "Express"
         1
     },
     "statusCode": 200,
     "isBase64Encoded": true
 3
 Log Output
 2022-11-02T11:06:38Z Start invoke request '7309717e-f597-4368-a7fe-89ac3d9b5df5', version: latest
 receive { message: 'helloworld' }
 2022-11-02T11:06:38Z Finish invoke request '7309717e-f597-4368-a7fe-89ac3d9b5df5', duration: 31.563ms, billing duration: 32ms, memory
 used: 10.566MB, billing memory: 128MB
 Summary
                       7309717e-f597-4368-a7fe-89ac3d9b5df5
 Request ID
 Memory Configured 128 MB
                     34.247 ms
Execution Duration
```

10.566 MB

35 ms

Memory Used

Billed Duration

•

.

•

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

Related Information

• For details about FunctionGraph concepts, see Concepts.

Function Output: displays the return result of the function.

Log Output: displays the execution logs of the function.

Summary: displays key information of the logs.

- For details about FunctionGraph pricing, see **FunctionGraph Pricing Details**.
- For details about the constraints and limitations of FunctionGraph, see Notes and Constraints.

4 Creating an Event Function Using a Container Image and Executing the Function

This section uses the creation of an event function using a container image as an example to describe how to create and test a container image function. You need to implement an HTTP server in the image and listen to port **8000** to receive requests. By default, the request path **/init** is the function initialization entry. Implement it as required. The request path **/invoke** is the function execution entry where trigger events are processed. For details about request parameters, see section "Supported Event Sources".

Prerequisites

1. Register with Huawei Cloud and complete real-name authentication.

For details, see **Registering a HUAWEI ID and Enabling Huawei Cloud Services** and **Real-Name Authentication**.

If you already have a Huawei account and have completed real-name authentication, skip this step.

2. View free quota.

FunctionGraph offers a free tier every month, which you can share with your IAM users. For details, see **Free Tier**

If you continue to use FunctionGraph after the free quota is used up, your account goes into arrears if the balance is less than the bill to be settled. To continue using your resources, top up your account in time..

3. Grant the FunctionGraph operation permissions to the user.

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

Step 1: Create an Image

Take the Linux x86 64-bit OS as an example. (No system configuration is required.)

1. Create a folder. mkdir custom_container_event_example && cd custom_container_event_example 2. Implement an HTTP server to process **init** and **invoke** requests and give a response. Node.js is used as an example.

Create the **main.js** file to introduce the Express framework and implement a function handler (method **POST** and path **/invoke** and an initializer (method **POST** and path **/init**).

```
const express = require('express');
const PORT = 8000;
const app = express();
app.use(express.json());
app.post('/init', (req, res) => {
    console.log('receive', req.body);
    res.send('Hello init\n');
});
app.post('/invoke', (req, res) => {
    console.log('receive', req.body);
    res.send('Hello invoke\n');
});
app.listen(PORT, () => {
    console.log(`Listening on http://localhost:${PORT}`);
```

});

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

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

- name: project name
- version: project version
- main: application entry file
- dependencies: all available dependencies of the project in npm
- 4. Create a Dockerfile. FROM node:12.10.0

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

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

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

RUN cd \${HOME} && npm install

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

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

USER \${USER_NAME} WORKDIR /\${HOME}

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

- FROM: Specify base image node:12.10.0. The base image is mandatory and its value can be changed.
- ENV: Set environment variables HOME (/home/custom_container), GROUP_NAME and USER_NAME (custom_container), USER_ID and GROUP_ID (1003). These environment variables are mandatory and their values can be changed.
- RUN: Use the format RUN <*Command>*. For example, RUN mkdir -m 550 \${HOME}, which means to create the home directory for user \$ {*USER_NAME*} during container building.
- **USER**: Switch to user *\${USER_NAME}*.
- WORKDIR: Switch the working directory to the /\${HOME} directory of user \${USER_NAME}.
- COPY: Copy main.js and package.json to the home directory of user \$
 {USER_NAME} in the container.
- EXPOSE: Expose port 8000 of the container. Do not change this parameter.
- **ENTRYPOINT**: Run the **node /home/tester/main.js** command to start the container.

NOTE

- 1. You can use any base image.
- In the cloud environment, UID 1003 and GID 1003 are used to start the container by default. The two IDs can be modified by choosing Configuration > Basic Settings > Container Image Override on the function details page. They cannot be root or a reserved ID.
- 3. If the basic image of the Alpine version is used, run the **addgroup** and **adduser** commands.
- 5. Build an image.

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

docker build -t custom_container_event_example:latest .

Step 2: Perform Local Verification

- 1. Start the Docker container. docker run -u 1003:1003 -p 8000:8000 custom_container_event_example:latest
- 2. Open a new Command Prompt, and send a message through port 8000 to access the **/init** directory specified in the template code. curl -XPOST -H 'Content-Type: application/json' localhost:8000/init

The following information is returned based on the module code: Hello init 3. Open a new Command Prompt, and send a message through port 8000 to access the **/invoke** directory specified in the template code. curl -XPOST -H 'Content-Type: application/json' -d '{"message":"HelloWorld"}' localhost:8000/invoke

The following information is returned based on the module code:

Hello invoke

4. Check whether the following information is displayed: Listening on http://localhost:8000

receive {}
receive { message: 'HelloWorld' }



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



Step 3: Upload the Image

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

SWR	My Images ①	1 Uplaad Through Client
Dashboard My Images	Private Images Shared Images	

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

Step 4: Create a Function

- In the left navigation pane of the management console, choose Compute > FunctionGraph. On the FunctionGraph console, choose Functions > Function List from the navigation pane.
- 2. Click **Create Function** in the upper right corner and choose **Container Image**.
- 3. Set the basic information.
 - Function Type: Select Event Function.
 - **Region**: The default value is used. You can select other regions.

D NOTE

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

- Project: The default value is the same as the selected region.
- Function Name: Enter custom_container_event.
- **Enterprise Project**: The default value is **default**. You can select the created enterprise project.

D NOTE

Enterprise projects let you manage cloud resources and users by project.

- **Agency**: Select an agency with the **SWR Admin** permission. If no agency is available, create one by referring to section "Creating an Agency"
- Container Image: Enter the image uploaded to SWR. Example: swr. {Region ID}.myhuaweicloud.com/{Organization name}] {Image name}: {Image tag}
- 4. (Optional) Override the container image.
 - **CMD**: container startup command. Example: **/bin/sh**. If no command is specified, the entrypoint or CMD in the image configuration will be used.
 - **Args**: container startup parameter. Example: **-args,value1**. If no argument is specified, CMD in the image configuration will be used.
 - User ID: Enter the user ID.
 - **Group ID**: Enter the user group ID.
- 5. After the configuration is complete, click **Create Function**.
- 6. On the function details page, choose **Configuration** > **Lifecycle**, and enable **Initialization**. The **init** API will be called to initialize the function.

Step 5: Test the Function

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

"message": "HelloWorld"

Step 6: View the Execution Result

}

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

Figure 4-1 Execution result

```
Execution Result ×
 Execution successful
 Function Output
     "body": "SGVsbG8gRnVuY3Rpb25HcmFwaCwgbWV0aG9kIFBPU1Q=",
     "headers": {
         "Content-Length": [
             "32"
         1.
         "Content-Type": [
             "text/html; charset=utf-8"
         Ъ
         "Date": [
             "Wed, 02 Nov 2022 11:06:38 GMT"
         1,
         "Etag": [
             "W/\"20-uygbC2IEf2PxTTMC0H1BL5d/vwI\""
         1,
          "X-Powered-By": [
            "Express"
         1
     },
     "statusCode": 200,
     "isBase64Encoded": true
 3
 Log Output
 2022-11-02T11:06:38Z Start invoke request '7309717e-f597-4368-a7fe-89ac3d9b5df5', version: latest
 receive { message: 'helloworld' }
 2022-11-02T11:06:38Z Finish invoke request '7309717e-f597-4368-a7fe-89ac3d9b5df5', duration: 31.563ms, billing duration: 32ms, memory
 used: 10.566MB, billing memory: 128MB
 Summary
                       7309717e-f597-4368-a7fe-89ac3d9b5df5
Request ID
 Memory Configured 128 MB
                     34.247 ms
Execution Duration
```

Related Information

Memory Used

NOTE

Billed Duration

•

•

.

10.566 MB

"Querying Function Logs".

35 ms

• For details about FunctionGraph concepts, see Concepts.

Function Output: displays the return result of the function.

Log Output: displays the execution logs of the function.

Summary: displays key information of the logs.

• For details about FunctionGraph pricing, see **FunctionGraph Pricing Details**.

A maximum of 2 KB logs can be displayed. For more log information, see section

 For details about the constraints and limitations of FunctionGraph, see Notes and Constraints.