

Serverless Cloud Function

Development Guide

Product Documentation



Copyright Notice

©2013-2025 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by the Tencent corporate group, including its parent, subsidiaries and affiliated companies, as the case may be. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Development Guide

Basic Concepts

Testing a Function

Environment Variables

Dependency Installation

Using Container Image

Error Types and Retry Policies

Dead Letter Queue

Connecting SCF to Database

Automated Deployment

Cloud Function Status Code

Common Errors and Solutions

Development Guide

Basic Concepts

Last updated : 2024-12-02 19:46:29

Serverless Cloud Function (SCF) provides two deployment methods of code deployment and image deployment and supports two function types of event-triggered function and HTTP-triggered function. Different deployment methods and function types require different specifications during code development. This document describes the writing specifications and related concepts of event-triggered function in code deployment. For more information on [image deployment](#) and [HTTP-triggered function](#), please see the corresponding documents.

An SCF event-triggered function involves three basic concepts: execution method, function input parameter, and function return.

Note:

The above concepts correspond respectively to the following in general project development:

Execution method: corresponds to the main function of the project and is the starting point of program execution.

Function input parameter: refers to function input parameters in a normal sense. However, in the SCF environment, the input parameters of an entry function are fixed values. For more information, please see [Function Input Parameters](#).

Function return: corresponds to the returned value of the main function in the project. After the function returns, the code execution ends.

Execution Method

When the SCF platform invokes a function, it will first find an execution method as the entry point to execute your code. At this time, you need to set in the format of **filename.execution method name**.

For example, if the user-configured execution method is `index.handler`, the SCF platform will first look for the `index` file in the code package and find the `handler` method in the file to start execution.

In the execution method, you can process the input parameters of the entry function and call other methods in the code arbitrarily. In SCF, the completion of the execution of the entry function or the exception of the execution of the function marks the end of execution.

Function Input Parameters

Function input parameters refer to the content that is passed to the function when the function is triggered. Usually, there are two input parameters: `event` and `context`. However, the number of input parameters may vary by

programming language and environment. For more information, please see [Serverless Cloud Function](#).

event

context

Usage

The `event` parameter is of `dict` type and contains the basic information that triggers the function. It can be in a platform-defined or custom format. After the function is triggered, the event can be processed inside the code.

Instructions

There are two ways to trigger an SCF function:

1. Trigger by calling [TencentCloud API](#).
2. Trigger by binding a [trigger](#).

These two SCF trigger methods correspond to two event formats:

TencentCloud API:

You can freely define a parameter of `dict` type between the invoker and the function code, where the invoker passes in the data in the format agreed upon, and the function code gets the data in the format.

Sample:

You can define a data structure `{"key": "XXX"}` of `dict` type, and when the invoker passes in the data `{"key": "abctest"}`, the function code can get the value `abctest` through `event[key]`.

Trigger:

SCF can be connected with various Tencent Cloud services such as API Gateway, COS, and CKafka, so you can bind a corresponding Tencent Cloud service trigger to a function. When the function is triggered, the service will pass the event to SCF as the `event` parameter in a platform-predefined unchangeable format. You can write code based on this format and get information from the `event` parameter.

Sample:

When COS triggers a function, the specific information of the bucket and the file will be passed to the `event` parameter in [JSON](#) format. The processing of the triggering event can be completed by parsing the `event` information in the function code.

Usage

`context` is an input parameter provided by the SCF platform. It is passed to the execution method, by parsing which the code can get the runtime environment and related information of the current request.

Instructions

The fields and descriptions of the `context` input parameter provided by SCF are as follows:

Field	Description
<code>memory_limit_in_mb</code>	Configured function memory size

time_limit_in_ms	Timeout period for function execution
request_id	Function execution request ID
environment	Function namespace information
environ	Function namespace information
function_version	Function version information
function_name	Function name
namespace	Function namespace information
tencentcloud_region	Function region
tencentcloud_appid	Function's Tencent Cloud APPID
tencentcloud_uin	Function's Tencent Cloud account ID

Note:

To ensure compatibility, the descriptions of the namespace at different stages of the SCF function are retained in the context.

The content of the context structure may be increased as the SCF platform is iterated.

You can print the context information through the standard output statement in the function code. Take the `python` runtime environment as an example:

```
# -*- coding: utf8 -*-
import json
def main_handler(event, context):
    print(context)
    return("Hello World")
```

The following context information can be obtained:

```
{"memory_limit_in_mb": 128, "time_limit_in_ms": 3000, "request_id": "f03dc946-3df4-45a0-8e54-xxxxxxxxxxxx", "environment": "
{\\\\"SCF_NAMESPACE\\\\":\\\\"default\\\\"}", "environ":
"SCF_NAMESPACE=default;SCF_NAMESPACE=default", "function_version": "$LATEST",
"function_name": "hello-from-scf", "namespace": "default",
"tencentcloud_region": "ap-guangzhou", "tencentcloud_appid": "12xxxxx384",
"tencentcloud_uin": "10000xxxxx36"}
```

After understanding the basic usage of `event` and `context` input parameters, you should pay attention to the following points when writing function code:

To ensure uniformity for each programming language and environment, ``event`` and ``context`` should be uniformly encapsulated in the ``JSON`` data format.

Different triggers pass different data structures when triggering functions. For more information, please see [Trigger Overview](#).

If the function does not need any input, you can ignore the ``event`` and ``context`` parameters in your code.

Function Return

The SCF platform will get the returned value after the function is executed and handle according to different trigger type as listed below.

Trigger Type	Handling Method
Sync triggering	<p>If triggered by API Gateway or the TencentCloud API for sync invocation, the function will be triggered synchronously.</p> <p>For a function triggered synchronously, the SCF platform will not return the trigger result during function execution.</p> <p>After the function is executed, the SCF platform will encapsulate the returned value into JSON format and return it to the invoker.</p>
Async triggering	<p>For a function that is triggered asynchronously, the SCF will return the triggering request ID after receiving the triggering event.</p> <p>After the function is executed, the returned value will be encapsulated into JSON format and stored in the log.</p> <p>After the function execution is completed, you can query the log by the request ID in the return to get the returned value of the asynchronously triggered function.</p>

When the code in a function returns a specific value, it usually returns a specific data structure; for example:

Runtime Environment	Returned Structure Type
Python	Simple or dict data structure
Node.js	JSON Object
PHP	Array structure
Go	Simple data structure or struct with JSON description

To ensure uniformity for different programming languages and environments, the function return will be uniformly encapsulated in the JSON data format. For example, after SCF gets the returned value of the function in the above runtime environment, it will convert the returned data structure to JSON and return it to the invoker.

Note:

You should ensure that the returned value of the function can be converted to JSON format. If the object is returned directly and there is no JSON conversion method, SCF will fail when executing JSON conversion and prompt an error. For example, the returned value in the above runtime environment does not need to be converted to JSON format before it is returned; otherwise, the output string will be converted again.

Exception Handling

If an exception occurs during testing and executing a function, the SCF platform will handle the exception as much as possible and write the exception information into the log. Exceptions generated by function execution include caught exceptions (handled errors) and uncaught exceptions (unhandled errors).

Handling method

You can log in to the [SCF console](#) and follow the steps below to test exception handling:

1. Create a function and copy the following function code without adding any triggers.
2. Click **Test** in the console and select the "Hello World" test sample for testing.

This document provides the following three ways to throw exceptions, and you can choose how to handle exceptions in the code based on your actual needs.

Throw exceptions explicitly

Inherit the `Exception` class

Use the `Try` statement to capture errors

Sample

```
def always_failed_handler(event, context):  
    raise Exception('I failed!')
```

Description

This function will throw an exception during execution and return the following error message. The SCF platform will record this error message in the function log.

```
File "/var/user/index.py", line 2, in always_failed_handler  
raise Exception('I failed!')  
Exception: I failed!
```

Sample

```
class UserNameAlreadyExistsException(Exception): pass
```



```
def create_user(event):  
    raise UserNameAlreadyExistsException('The username already exists,  
    please change a name!')
```

Description

You can define how to handle errors in your code to ensure the robustness and scalability of your application.

Sample

```
def create_user(event):  
    try:  
        createUser(event[username], event[pwd])  
    except UserNameAlreadyExistsException, e: //catch error and do something
```

Description

You can define how to handle errors in your code to ensure the robustness and scalability of your application.

Returned error message

If exception handling and error capture are not performed in your code logic, the SCF platform will capture errors as much as possible such as when your function suddenly crashes and exits during execution. The platform will return a general error message if it cannot capture an error that occurs.

The table below lists some common errors in code execution:

Error Scenario	Error Message
<code>raise</code> is used to throw an exception	{File "/var/user/index.py", line 2, in always_failed_handler raise Exception('xxx') Exception: xxx}
The handler does not exist	{'module' object has no attribute 'xxx'}
The dependent module does not exist	{global name 'xxx' is not defined}
Timed out	{"time out"}

Log

The SCF platform stores all the records of function invocations and the outputs of the function code in logs. You can use the `printout` or `log` statement in the programming language to generate the output logs for debugging and troubleshooting. For more information, please see [Log Search Guide](#).

Notes

Because of the nature of SCF, you must write your function code in a **stateless** style. State characteristics in the lifecycle of a function such as local file storage will be destroyed after the function invocation ends.

Therefore, you are recommended to store persistent states in TDSQL, COS, TencentDB for Memcached, or other cloud storage services.

Development Process

For more information on the function development process, please see [Getting Started](#).

Testing a Function

Last updated : 2024-12-02 19:46:29

After creating a function, you can directly test it in the following ways to understand the function execution conditions and check the code execution process.

SCF console: [Creating Event-Triggered Function in Console](#)

Test Events and Templates

Functions are executed in an event-triggered method. Different triggers pass different event data structures when they trigger functions. The function testing method is to trigger the function by sending a simulated test event.

The SCF console provides the following event templates to simulate corresponding events:

Hello World event template: it contains simple data structure and content that can be used to trigger functions created by the hello world template.

COS file event template: it simulates file upload/deletion events in COS.

CMQ topic event template: it simulates message receiving events in a CMQ topic.

API Gateway event template: it simulates API request receiving events in API Gateway.

CKafka event template: it simulates message receiving events in a CKafka topic.

By clicking **Change** on the template management page in the console, you can change the currently used test template to another system-defined or custom template. For more information on message structures in event templates, please see [Trigger Event Message Structure Summary](#).

Custom Template Configuration and Usage

In addition to the system-provided event templates, you can create more custom templates. By clicking **Configure** on the template management page in the console, you can modify an existing template and save it as a custom template, or directly enter a test event designed by yourself and save it as a custom template.

Notes

When using the test event template feature, you need to pay attention to the following:

The test event template name can contain letters, digits, hyphens, and underscores and must begin with a letter.

On the same page, the created custom test templates can be deleted if they are no longer needed.

Up to five custom test templates can be configured for one single function. After the limit is reached, to configure a new one, please first delete an old one that is no longer in use.

Environment Variables

Last updated : 2024-12-02 19:46:29

When creating or editing a function, you can add, delete, or modify environment variables for the function runtime environment by modifying the environment variables in the configuration.

The configured environment variables will be configured into the OS environment when the function is executed. The function code can read the system environment variables to obtain the specific values and use them in the code.

Adding an environment variable

Adding an environment variable in the console

1. Log in to the [Serverless console](#) and click **Function Service** on the left sidebar.
2. When creating or editing a function, you can add environment variables in "Environment Variable". Environment variables usually appear as `key-value` pairs. Enter the required environment variable key in the first input box and the required value in the second one. Note that the value of `key` or `value` can contain 2-64 bytes of letters, digits, and underscores and must begin with a letter.

Adding an environment variable locally

For local development, you can configure the `Environment` environment variable directly under the function in `serverless.yml` and run the `scf deploy` command to deploy it to the cloud as shown below:

```
component: scf # Component name, which is required. It is `scf` in this example
name: scfdemo # Component instance name, which is required

# Component parameter configuration
inputs:
  name: scfdemo # Function name, which is `${name}-${stage}-${app}` by default
  namespace: default
  # 1. Default format. Create a specifically named COS bucket and upload it
  src: ./src
  type: event # Function. Valid values: event - event-triggered (default), web - HTTP
  handler: index.main_handler # Entry (valid if the function is event-triggered)
  runtime: Nodejs10.15 # Runtime environment, which is Nodejs10.15 by default
  region: ap-guangzhou # Function region
  description: This is a function in ${app} application.
  memorySize: 128 # Memory size in MB
  timeout: 20 # Function execution timeout period in seconds
  initTimeout: 3 # Initialization timeout period in seconds
  environment: # Environment variable
    variables: # Environment variable object
```

```
TEST1: value1
TEST2: value2
```

Viewing an environment variable

After configuring environment variables for the function, you can query the specific configured environment variables by viewing the function configuration, which are displayed in the form of `key=value`.

Using an environment variable

The configured environment variables will be configured into the runtime environment when the function is executed. The code can read the system environment variables to get the specific values and use them in the code. It should be noted that **environment variables cannot be read locally**.

Assume that the key of the configured environment variable for a function is `key`. The following is the sample codes for reading and printing the value of this environment variable in different runtime environments.

In a Python runtime environment, the way to read the environment variables is as follows:

```
import os
value = os.environ.get('key')
print(value)
```

In a Node.js runtime environment, the way to read the environment variables is as follows:

```
var value = process.env.key
console.log(value)
```

In a Java runtime environment, the way to read the environment variables varies by temporary authorized fields and other fields:

For temporary authorized fields (including `TENCENTCLOUD_SESSIONTOKEN`, `TENCENTCLOUD_SECRETID`, and `TENCENTCLOUD_SECRETKEY`), the way to read the environment variables is as follows:

```
System.out.println("value: " + System.getProperty("key"));
```

For other fields, the way to read the environment variables is as follows:

```
System.out.println("value: " + System.getenv("key"));
```

In a Go runtime environment, the way to read the environment variables is as follows:

```
import "os"
```

```
var value string
value = os.Getenv("key")
```

In a **PHP runtime environment**, the way to read the environment variables is as follows:

```
$value = getenv('key');
```

Overview

Variable value extraction: Values that may change in the business can be extracted into environment variables, eliminating the need to modify the code according to business changes.

External storage of encrypted information: Keys related to authentication and encryption can be extracted from the code into environment variables, avoiding security risks caused by the presence of relevant keys hard-coded in the code.

Environment differentiation: The configuration and database information for different development stages can be extracted into the environment variables, so that in different stages of development and release, you only need to modify the environment variable values and execute the development environment database and release environment database separately.

Use Limits

The following Use Limits apply to the environment variables of functions:

The key must begin with a letter (**[a-zA-Z]**) and can only contain alphanumeric characters and underscores (**[a-zA-Z0-9_]**).

The keys of reserved environment variables cannot be modified, including:

Keys beginning with SCF_, such as SCF_RUNTIME.

Keys beginning with QCLOUD_, such as QCLOUD_APPID.

Keys beginning with TENCENTCLOUD_, such as TENCENTCLOUD_SECRETID.

Built-in Environment Variables

The **Key** and **Value** of built-in environment variables in the current runtime environment are as shown in the table below:

Environment Variable Key	Specific Value or Value Source
TENCENTCLOUD_SESSIONTOKEN	{Temporary SESSION TOKEN}

TENCENTCLOUD_SECRETID	{Temporary SECRET ID}
TENCENTCLOUD_SECRETKEY	{Temporary SECRET KEY}
_SCF_SERVER_PORT	28902
TENCENTCLOUD_RUNENV	SCF
USER_CODE_ROOT	/var/user/
TRIGGER_SRC	Timer (if a timer trigger is used)
PYTHONDONTWRITEBYTECODE	x
PYTHONPATH	/var/user:/opt
CLASSPATH	/var/runtime/java x:/var/runtime/java x/lib/*:/opt (`x` is 8 or 11)
NODE_PATH	/var/user:/var/user/node_modules:/var/lang/node x/lib/node_modules:/opt:/opt/node_modules (`x` is 16, 14, 12, 10, 8, or 6)
PHP_INI_SCAN_DIR	/var/user/php_extension:/opt/php_extension
_	/var/lang/python3/bin/python x (`x` is 37, 3, or 2)
PWD	/var/user
LOGNAME	qcloud
LANG	en_US.UTF8
LC_ALL	en_US.UTF8
USER	qcloud
HOME	/home/qcloud
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SHELL	/bin/bash
SHLVL	3
LD_LIBRARY_PATH	/var/runtime/java x:/var/user:/opt (`x` is 8 or 11)
HOSTNAME	{host id}
SCF_RUNTIME	Function runtime

SCF_FUNCTIONNAME	Function name
SCF_FUNCTIONVERSION	Function version
TENCENTCLOUD_REGION	Region
TENCENTCLOUD_APPID	Account APPID
TENCENTCLOUD_UIN	Account UIN
TENCENTCLOUD_TZ	Time zone, which is UTC currently

Dependency Installation

Last updated : 2024-12-02 19:46:29

Built-in Dependencies

SCF runtime environments have provided some dependent libraries, which can be queried in the corresponding runtime development guide:

[Node.js](#)

[Python](#)

[PHP](#)

[Go](#)

Installing Dependent Libraries

You can save the dependent libraries of the SCF code in the code package and upload it to the cloud for use by SCF. SCF supports the following runtimes and usage methods:

Node.js runtime

The Node.js runtime supports installing dependent libraries in the following three ways:

Upload from local

Online Dependency Installation

Use dependency management tools

You can use a dependency manager (such as npm) to locally install the dependent library, package and upload it with the function code.

Note:

Place the entry point file of the function in the root directory of the `zip` package. If you package and upload the entire folder as a `zip` package, the function creation will fail because the entry point file cannot be found in the unzipped root directory.

This document takes installing the `lodash` library as an example.

1. Run the `mkdir test-package` command in the local terminal to create a directory for storing the function code and dependency libraries.
2. Run the following command to install the `lodash` dependency library in this directory.

```
cd test-package
npm install lodash
```

3. Create the function entry file `index.js` in this directory and import the `lodash` library in the code.

```
'use strict';
const _ = require('lodash');
exports.main_handler = async (event, context) => {
  console.log("Hello World")
  console.log(event)
  console.log(event["non-exist"])
  console.log(context)
  return event
};
```

4. Compress the function code and dependent library to a zip package. Upload the zip package and create a function via the [SCF console](#).

4.1 Log in to the [SCF console](#) and click **Functions** on the left sidebar.

4.2 Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

4.3 Enter the basic information of the function on the **Create function** page.

The screenshot shows the 'Create' page in the SCF console. It has three tabs: 'Template', 'Create from scratch' (selected), and 'Use TCR image'. The 'Basic configurations' section includes: 'Function type' (Event-triggered function selected), 'Function name' (hello), 'Region' (Guangzhou), 'Runtime environment' (Nodejs 12.16), and 'Time zone' (UTC). The 'Function codes' section shows 'Submitting method' (Local ZIP file selected) and 'Execution' (index.main_handler). There is a text area for 'Function codes' and an 'Upload' button. At the bottom, there is a checkbox for 'I have read and agree to TENCENT CLOUD TERMS OF SERVICE'.

Creation method: Select **Create from scratch**.

Runtime environment: Select **Node.js 12.16**.

Submitting method: Select **Local ZIP file**.

4.4 Click **Complete**.

The Node.js runtime provides an online dependency installation feature, which can install dependencies online according to the dependency information configured in `package.json`. For more information, see [Online Dependency Installation](#).

The SCF online editor [Serverless Web IDE](#) supports Terminal capability that has a built-in `npm` package management tool.

Note:

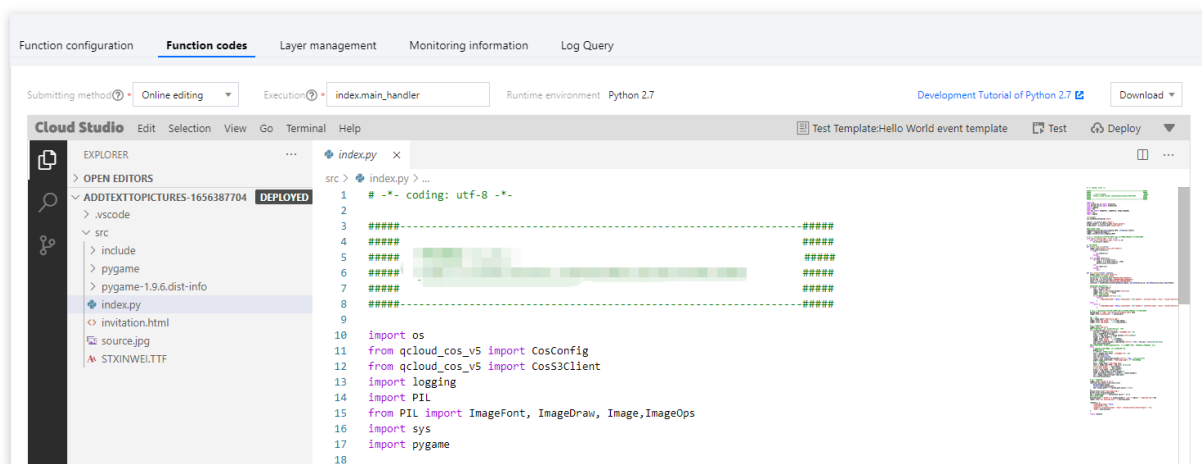
Serverless Web IDE has a delay in supporting newer versions of runtime environments. If the console does not open up the Serverless Web IDE for the corresponding runtime environment, package and upload the dependency library together with the code for dependency installation or install the dependency online.

This document takes installing the `lodash` library in the terminal as an example:

1. Log in to the [SCF console](#) and click **Functions** on the left sidebar.
2. In the function list, click a function name to enter the function details page.
3. On the **Function management** page, select **Function codes** > **Edit codes** to view and edit the function.
4. Select **Terminal** > **New terminal** on the topbar of the IDE to open the terminal window.
5. Run the following command in the terminal to install the `lodash` dependency library:

```
cd src # Install the dependent library in the same level of directory as the
function's entry point file, namely, enter the `src` directory and install it.
npm install lodash # You can view the npm version through the terminal
```

6. After the installation is completed, view `package.json` and `node_modules` in the file tree on the left side of the IDE.
7. After you click **Deploy**, the dependency library can be packaged and uploaded to the cloud together with the function code.



Python runtime

The Python runtime supports installing dependent libraries in the following two ways.

Upload from local

Use dependency management tools

You can use a dependency manager (such as pip) to locally install the dependent library, package and upload it with the function code.

Note:

Place the entry point file of the function in the root directory of the `zip` package. If you package and upload the entire folder as a `zip` package, the function creation will fail because the entry point file cannot be found in the unzipped root directory.

Due to runtime environment differences, confirm that the installed dependency version is adapted to the function runtime environment.

The function runtime environment is CentOS 7, and you need to install the dependencies in the same environment. If not, an error where the dependencies cannot be found may occur while running the function after upload.

If some dependencies involve a dynamic link library, such as Pandas in Python 3.6, you need to manually copy the relevant dependency package to the dependency installation directory before packaging and uploading them. For more information, see [Installing Dependency with Docker](#). You can also use the online IDE for installation.

This document takes installing the `numpy` library as an example:

1. Run the `mkdir test-package` command in the local terminal to create a directory for storing the function code and dependency libraries.
2. Run the following command to install the `numpy` dependency library in this directory.

```
cd test-package
pip install numpy -t . # Check whether the pip version you are using is adapted
to the function runtime environment
```

3. Create the function entry file `index.py` in this directory and import the `numpy` library in the code.

```
# -*- coding: utf8 -*-
import json
import numpy
def main_handler(event, context):
    print("Received event: " + json.dumps(event, indent = 2))
    print("Received context: " + str(context))
    print("Hello world")
    return("Hello World")
```

4. Compress the function code and dependent library to a zip package. Upload the zip package and create a function via the [SCF console](#).

4.1 Log in to the [SCF console](#) and click **Functions** on the left sidebar.

4.2 Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

4.3 Enter the basic information of the function on the **Create function** page.

Create

Template
Use demo template to create a function or application

Create from scratch
Start from a Hello World sample

Use TCR image
Create a function based on a TCR image

Basic configurations

Function type * ☒ Event-triggered function
Triggers functions by JSON events from Cloud API and other triggers [here](#)

☐ HTTP-triggered Function
Triggers functions by HTTP requests, which is applicable to web-based scenarios [here](#)

Function name * helloworld-1672390755
2 to 60 characters ([a-z], [A-Z], [0-9] and [-_]). It must start with a letter and end with a digit or letter.

Region * Guangzhou

Runtime environment * Python 3.6

Time zone * UTC

Function codes

Submitting method * ☐ Online editing ☒ Local ZIP file ☐ Local folder ☐ Upload a ZIP pack via COS

Execution * index.main_handler

Function codes * Upload
Please upload a code package in zip format. The max file size is 50M. If the zip is larger than 10M, only the entry file is displayed.

☐ I have read and agree to [TENCENT CLOUD TERMS OF SERVICE](#)

Creation method: Select **Create from scratch**.

Runtime environment: Select **Python 3.6**.

Submitting method: Select **Local ZIP file**.

4.4 Click **Complete**.

The SCF online editor [Serverless Web IDE](#) supports Terminal capability that has a built-in `pip` package management tool.

Note:

Serverless Web IDE has a delay in supporting newer versions of runtime environments. If the console does not open up the Serverless Web IDE for the corresponding runtime environment, package and upload the dependency library together with the code for dependency installation or install the dependency online.

This document takes installing the `numpy` library in the terminal as an example:

1. Log in to the [SCF console](#) and click **Functions** on the left sidebar.
2. In the function list, click a function name to enter the function details page.
3. On the **Function management** page, select **Function codes** > **Edit codes** to view and edit the function.
4. Select **Terminal** > **New terminal** on the topbar of the IDE to open the terminal window.
5. Run the following command in the terminal to install the `numpy` dependency library:

```
cd src # Install the dependent library in the same level of directory as the
function's entry point file, namely, enter the `src` directory and install it.
pip install numpy -t . # You can view the pip version through the terminal to
check whether it is adapted to the function runtime environment
```

6. After the installation is completed, view the installed dependency libraries in the file tree on the left side of the IDE.
7. After you click **Deploy**, the dependency library can be packaged and uploaded to the cloud together with the function code.

Note:

You can run `pip freeze > requirements.txt` to generate the `requirements.txt` files for all dependencies in the local environment.

Run `pip install -r requirements.txt -t .` in the terminal of the IDE to install the dependency package according to the configuration in `requirements.txt`.

PHP runtime

Note:

The PHP versions supported by SCF are 5.6, 7.2, 7.4 and 8.0. Different minor versions of PHP may be incompatible. Check the version number first before installing dependencies.

Install custom libraries

Install custom extensions

You can use a dependency manager (such as “composer”) to locally install the dependent library, package and upload it with the function code.

Note:

Place the entry point file of the function in the root directory of the `zip` package. If you package and upload the entire folder as a `zip` package, the function creation will fail because the entry point file cannot be found in the unzipped root directory.

This document takes installing the `requests` library for PHP 7.2 as an example:

1. Run the `mkdir test-package` command in the local terminal to create a directory for storing the function code and dependency libraries.
2. Create `Composer.json` under `test-package` and specify the dependency library and version to be installed.

```
{
  "require": {
    "requests": ">=1.0"
  }
}
```

3. Run the following command to install the `requests` dependency library in this directory.

```
cd test-package
composer install
```

4. Create the function entry file `index.php` in this directory and import the `requests` library in the code.

```
<?php
```

```
require 'vendor/autoload.php';  
function main_handler($event, $context) {  
    return "hello world";  
}  
?>
```

5. Compress the function code and dependent library to a zip package. Upload the zip package and create a function via the [SCF console](#).

5.1 Log in to the [SCF console](#) and click **Functions** on the left sidebar.

5.2 Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

5.3 Enter the basic information of the function on the **Create function** page.

Creation method: Select **Create from scratch**.

Runtime environment: Select **Php7.2**.

Submitting method: Select **Local ZIP file**.

5.4 Click **Complete**.

Create the extension folder `php_extension` in a directory at the same level as the function entry file, add the custom extension file `.so` and configuration file `php.ini`, and package and upload them together with the function code.

This document uses installing the custom extension `swoole.so` for PHP 7.2 as an example.

1. Run the `mkdir test-package` command in the local terminal to create a directory for storing the function code and dependency libraries.
2. Run the following command to create the folder `php_extension` in `test-package` and place the configuration file `php.ini` and extension file `.so` corresponding to the extension in this directory. The directory structure is as follows:

Note:

The extension folder `php_extension` and configuration file `php.ini` are given fixed names. If other names are used, the extension may fail to load.

The extension folder `php_extension`, the configuration file `php.ini`, and the custom extension file `.so` need to have executable permissions.

```
|__php_extension
| |__php.ini
| |__swoole.so
|__index.php
```

3. Custom extensions can be loaded from the code or layers. If an extension is uploaded as a layer, make sure that the unzipped directory structure of the uploaded zip file is as follows:

```
|__php_extension
| |__swoole.so
```

4. `php.ini` writing method:

The extension is in the code directory:

```
extension=/var/user/php_extension/swoole.so
```

This extension is in the layer directory:

```
extension=/opt/php_extension/swoole.so
```

5. Create the function entry file `index.php` in this directory. Check whether the extension is loaded successfully through the `extension_loaded()` function, and if so, `true` will be returned; otherwise, `false` will be returned.

```
<?php
function main_handler($event, $context) {
    var_dump(extension_loaded('swoole'));
    return "hello world";
}
?>
```

6. Compress the function code and dependent library to a zip package. Upload the zip package and create a function via the [SCF console](#).

1. Log in to the [SCF console](#) and click **Functions** on the left sidebar.

6.1 Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

6.2 Enter the basic information of the function on the **Create function** page.

Create

Template
Use demo template to create a function or application

Create from scratch
Start from a Hello World sample

Use TCR image
Create a function based on a TCR image

Basic configurations

Function type * ☒ Event-triggered function
Triggers functions by JSON events from Cloud API and other triggers [here](#)

☐ HTTP-triggered Function
Triggers functions by HTTP requests, which is applicable to web-based scenarios [here](#)

Function name *
2 to 60 characters ([a-z], [A-Z], [0-9] and [-_]). It must start with a letter and end with a digit or letter.

Region *

Runtime environment *

Time zone * ⓘ

Function codes

Submitting method * ☐ Online editing ☒ Local ZIP file ☐ Local folder ☐ Upload a ZIP pack via COS

Execution * ⓘ

Function codes *
[Upload](#)

Please upload a code package in zip format. The max file size is 50M. If the zip is larger than 10M, only the entry file is displayed.

☐ I have read and agree to [TENCENT CLOUD TERMS OF SERVICE](#)

[Complete](#) [Cancel](#)

Creation method: Select **Create from scratch**.

Runtime environment: Select **Php7.2**.

Submitting method: Select **Local ZIP file**.

6.3 Click **Complete**.

Java runtime

You can use a dependency manager (such as maven) to locally install the dependent library, package and upload it with the function code.

1. Run the `mkdir test-package` command in the local terminal to create a directory for storing the function code and dependency libraries.

2. Create `pom.xml` in this directory and configure the dependency information in `pom.xml`.

3. Run the `mvn package` command in the root directory of the project folder, and the compilation output is as follows:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building java-example 1.0-SNAPSHOT
[INFO] -----
[INFO]
...
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.785 s
[INFO] Finished at: 2017-08-25T10:53:54+08:00
[INFO] Final Memory: 17M/214M
[INFO] -----
```

4. Compress the function code and dependent library to a jar package. Upload the jar package and create a function via the [SCF console](#).

4.1 Log in to the [SCF console](#) and click **Functions** on the left sidebar.

4.2 Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

4.3 Enter the basic information of the function on the **Create function** page.

Create

Template
Use demo template to create a function or application

Create from scratch
Start from a Hello World sample

Use TCR image
Create a function based on a TCR image

Basic configurations

Function type * ☒ Event-triggered function
Triggers functions by JSON events from Cloud API and other triggers[here](#)

☐ HTTP-triggered Function
Triggers functions by HTTP requests, which is applicable to web-based scenarios[here](#)

Function name *
2 to 60 characters ([a-z], [A-Z], [0-9] and [-_]). It must start with a letter and end with a digit or letter.

Region *

Runtime environment *

Time zone * ⓘ

Function codes

Submitting method * ☐ Online editing ☒ Local ZIP file ☐ Local folder ☐ Upload a ZIP pack via COS

Execution * ⓘ

Function codes *
Please upload a code package in zip/jar format. The max file size is 50M. If the zip is larger than 10M, only the entry file is displayed.

☐ I have read and agree to [TENCENT CLOUD TERMS OF SERVICE](#)

Creation method: Select **Create from scratch**.

Runtime environment: Select **Java 8**.

Submitting method: Select **Local ZIP file**.

4.4 Click **Complete**.

Go runtime

Instructions: upload the final binary file when packaging.

Compile the dependent library of the Go runtime with codes to generate a binary file. Upload the binary file and create a function via the [SCF console](#).

1. Log in to the [SCF console](#) and click **Functions** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Enter the basic information of the function on the **Create function** page.

Create

Template
Use demo template to create a function or application

Create from scratch
Start from a Hello World sample

Use TCR image
Create a function based on a TCR image

Basic configurations

Function type *

☒ Event-triggered function
Triggers functions by JSON events from Cloud API and other triggers[here](#)

☐ HTTP-triggered Function
Triggers functions by HTTP requests, which is applicable to web-based scenarios[here](#)

Function name *

helloworld

2 to 60 characters ([a-z], [A-Z], [0-9] and [-_]). It must start with a letter and end with a digit or letter.

Region *

Guangzhou

Runtime environment *

Go 1

Time zone *

UTC

Function codes

Submitting method *

☐ Online editing ☒ Local ZIP file ☐ Local folder ☐ Upload a ZIP pack via COS

Execution *

main

Function codes *

Upload

Please upload a code package in zip format. The max file size is 50M. If the zip is larger than 10M, only the entry file is displayed.

☐ I have read and agree to [TENCENT CLOUD TERMS OF SERVICE](#)

Creation method: Select **Create from scratch**.

Runtime environment: Select **Go 1**.

Submitting method: Select **Local ZIP file**.

4. Click **Complete**.

Using Container Image

Last updated : 2024-12-02 19:46:29

Overview

The SCF container image feature has been launched, so you can use images for development. This document describes how to install an image and use it for development.

Prerequisites

[Docker](#) has been installed in the development environment.

Directions

Getting image

The SCF image is based on CentOS 7.7.1908 and available as a [public image](#) in TKE. You can search for `scf-repo` on the public image page to [view the image information](#).

1. Run the following command to pull the image:

```
# Pull the SCF source image
docker pull ccr.ccs.tencentyun.com/scf-repo/scf-runtimes-image:latest
```

Note:

If the command prompts a permission error and cannot be executed normally, add `sudo` before the command and try again.

2. You can view the runtime contained in the current image in the `/scf/lang/` directory.

As the SCF source image contains all runtimes, it is relatively large. Please refer to the following table and choose a runtime image according to your needs.

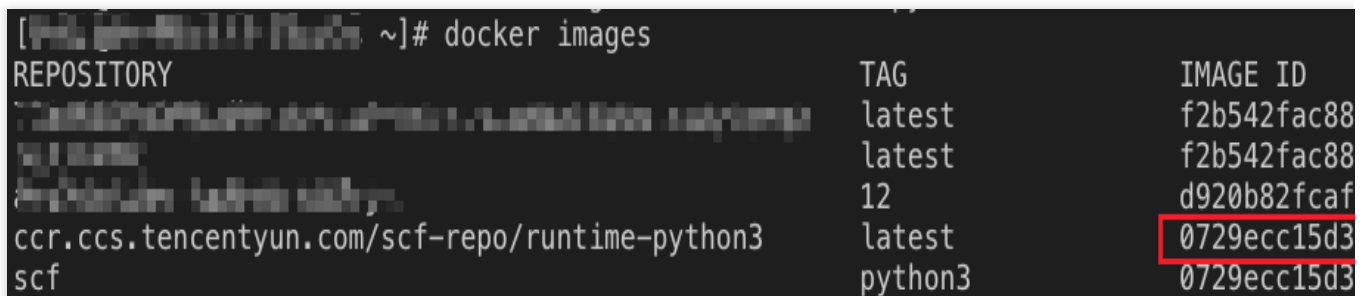
Runtime	Address
SCF	ccr.ccs.tencentyun.com/scf-repo/scf-runtimes-image:latest
Go 1.8	ccr.ccs.tencentyun.com/scf-repo/runtime-go1:latest
Python 2.7	ccr.ccs.tencentyun.com/scf-repo/runtime-python2:latest
Python 3.6	ccr.ccs.tencentyun.com/scf-repo/runtime-python3:latest

PHP 5.6	ccr.ccs.tencentyun.com/scf-repo/runtime-php5:latest
PHP 7.2	ccr.ccs.tencentyun.com/scf-repo/runtime-php7:latest
Java 8	ccr.ccs.tencentyun.com/scf-repo/runtime-java8:latest
Node.js 6.10	ccr.ccs.tencentyun.com/scf-repo/runtime-node6:latest
Node.js 8.9	ccr.ccs.tencentyun.com/scf-repo/runtime-node8:latest
Node.js 10.15	ccr.ccs.tencentyun.com/scf-repo/runtime-node10:latest
Node.js 12.16	ccr.ccs.tencentyun.com/scf-repo/runtime-node12:latest

3. This document uses the `scf:python3` tag as an example. Run the following command to retag the image:

```
docker pull ccr.ccs.tencentyun.com/scf-repo/runtime-python3:latest
# Run this command to find the IMAGE ID and copy it
docker images
# docker tag IMAGE_ID REPOSITORY:TAG
docker tag 0729ecc15d37 scf:python3
```

The execution result is as shown below:



```
[root@ip-10.10.10.10 ~]# docker images
REPOSITORY                                TAG                IMAGE ID
ccr.ccs.tencentyun.com/scf-repo/runtime-python3  latest            f2b542fac88
ccr.ccs.tencentyun.com/scf-repo/runtime-python3  latest            f2b542fac88
ccr.ccs.tencentyun.com/scf-repo/runtime-python3  12                d920b82fcac
ccr.ccs.tencentyun.com/scf-repo/runtime-python3  latest            0729ecc15d3
scf                                             python3            0729ecc15d3
```

Note:

If you don't want to tag the image, you need to replace `scf:python3` in the sample with `ccr.ccs.tencentyun.com/scf-repo/runtime-python3:latest`.

Using image for dependency installation

This document uses the NodeJieba dependency in the Node.js 12 environment as an example to describe how to install dependencies with an image.

Getting Node.js 12 image

Run the following command to pull the image:

```
docker pull ccr.ccs.tencentyun.com/scf-repo/runtime-node12:latest
```

```
# Run this command to find the IMAGE ID and copy it
docker images
docker tag d64a665357b6 scf:node12
```

Starting container and mounting directory

Run the following command to start the container and mount the local directory to a directory in the container (if the directory does not exist, it will be created automatically). This document uses mounting the

`/path/to/your_project` directory to the `/tmp/your_project` directory in the container as an example.

```
docker run -it -v /path/to/your_project:/tmp/your_project scf:node12 /bin/bash
```

Installing dependencies in container

1. After starting the container, run the `cd` command to enter the directory in the container. Then, run the `npm` command to install NodeJieba in this directory as shown below:

```
cd /tmp/your_project
npm install nodejieba --save
```

2. The dependency will be installed in the local `/path/to/your_project` directory. Run the `exit` command to exit the container as shown below:

```
# Exit the container
exit
```

Following the above steps, you can install dependencies through the container image and then redeploy the code to SCF. For the Node.js language, [online dependency installation](#) is also supported, so that dependencies will be automatically installed upon upload.

Using image for development

This document uses Python 3.6 as an example to describe how to use a container for development and testing.

Getting Python 3.6 image

Run the following command to pull the image:

```
docker pull ccr.ccs.tencentyun.com/scf-repo/runtime-python3:latest
# Run this command to find the IMAGE ID and copy it
docker images
docker tag d64a665357b6 scf:python3
```

Starting container and mounting directory

1. Run the following command to start the container and mount the local project directory to a directory in the container (if the directory does not exist, it will be created automatically):

```
docker run -it -v /path/to/your_project:/tmp/your_project scf:node12 /bin/bash
```

2. Run the `docker exec` command to enter the container for development as shown below:

```
docker ps
# Get the CONTAINER ID
docker exec -it CONTAINER_ID /bin/bash
```

Saving image

Run the following command to submit changes to the local image for subsequent use:

```
# Get the container ID
docker ps
# Save the image locally
# docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
docker commit db47b8e66e64 scf:myimage
```


Error Types and Retry Policies

Last updated : 2024-12-02 19:46:29

A function invocation may fail for various reasons. Different **error types** and **invocation methods (sync or async invocation)** all affect the retry policy. You can configure a [dead letter queue \(DLQ\)](#) to collect error event information and analyze causes of failures.

Error Types

A function invocation may fail for various reasons. The errors can be divided into the following types:

Invocation error

An invocation error occurs before the function is actually executed. It will occur in the following cases:

Invocation request error. For example, the data structure of the event passed in is too large, an input parameter does not meet the requirements, or the function does not exist.

Invoker error. This error generally occurs when the invoker does not have required permissions.

Overrun error. The number of concurrent invocations exceeds the [maximum concurrency](#) limit.

Execution error

An execution error occurs during the actual execution of a function. It will occur in the following cases:

User code execution error. This type of errors occurs during the execution of user code; for example, the function code throws an exception, or the format of the returned result is exceptional.

Runtime error. During function execution, the runtime is responsible for pulling and executing user code. A runtime error refers to errors detected and reported by the runtime, such as function execution timeout (for the timeout period, see [Quota Limits](#)) and code syntax error.

System error

It refers to errors of the function platform, such as internal error.

Retry Policy

Different **error types** and **invocation methods (sync or async invocation)** all affect the retry policy.

Sync invocation

Types of sync invocation include sync invocation by [TencentCloud API trigger](#), [API Gateway trigger](#), [CKafka trigger](#), and [CLB trigger](#).

In sync invocation, the error message will be directly returned; therefore, when an error occurs in sync invocation, the platform will not automatically retry, and the retry policy (i.e., whether to retry and the number of retries) will be determined by the invoker.

Note:

A CKafka trigger will create a backend module as a consumer that can connect to a CKafka instance and consume messages. After obtaining the message, the backend module will synchronously invoke the triggered function. Since the backend module of the CKafka trigger is maintained by SCF, the retry policy for sync invocation will also be controlled by SCF.

For execution errors (including user code errors and runtime errors), the CKafka trigger will retry according to the configured retry times, which is 10,000 by default.

For overrun errors and system errors, the CKafka trigger will continue to retry in an exponential backoff manner until it succeeds.

Async invocation

Types of async invocation include async invocation by [TencentCloud API trigger](#), [COS trigger](#), [scheduled trigger](#), [CMQ topic trigger](#), etc. For specific trigger invocation types, see [Trigger Overview](#).

You can modify and customize the default **retry attempts** and **maximum waiting time** in the function configuration according to your business needs. This configuration is only applicable to async invocations.

Async Invocation

Maximum event age

6

hr(s)

0

min(s)

0

s

Range of maximum event age: 6 hr(s)-1 min(s)

Retry Attempts

2

times

Retry Attempts: the number of times the function retries when an error is returned. This parameter is only applicable to the policy configuration for **execution errors**. The default value is 2 retries.

Maximum Event Age: the maximum time that the function keeps events in the async event queue. This parameter is applicable to the retry configuration of all async invocations. The default value is 6 hours, and the maximum queue length can reach up to 100,000 events.

Async invocation retry policies for different types of errors:

Error Type	Retry Policy
System error	The function request execution status code is 500. When an error of this type occurs, the SCF platform will retry for the configured maximum event age (which is 6 hours by default) at intervals of one minute. If a DLQ is configured, events that

	still fail after the maximum event age elapses will be sent to it for further processing on your own; otherwise, they will be discarded by the SCF platform.
Overrun error	The function request execution status code is 432. When an error of this type occurs, the SCF platform will retry for the configured maximum event age (which is 6 hours by default) at intervals of one minute. If a DLQ is configured, events that still fail after the maximum event age elapses will be sent to it for further processing on your own; otherwise, they will be discarded by the SCF platform.
Execution errors(except system errors and overrun errors, all other errors are execution errors)	When an error of this type occurs, the SCF platform will retry for the configured number of retries at intervals of one minute. While automatically retrying, the function can still handle new triggering events normally. If a DLQ is configured, events that still fail after retries for the configured number of times or exceed the maximum waiting time will be passed to it; otherwise, the events will be discarded by the SCF platform.

Note:

1. Due to the differences in execution mechanisms, the retry and dead letter queue configurations don't work for errors during the execution of [asynchronously executed functions](#).
2. How to judge whether the maximum waiting time is exceeded: if event retry time - event initial trigger time is greater than the maximum waiting time, the maximum waiting time is exceeded.

Dead Letter Queue

Last updated : 2024-12-02 19:46:29

Overview

A dead letter queue (DLQ) is a message queue under your account used to collect error event information and analyze causes of failures. If you have configured a DLQ for a function, an event will be sent to the DLQ if:

It still fails after the SCF platform retries it twice due to a **user code execution error**

It still fails after the SCF platform retries it for more than 24 hours due to an **overrun error** or **system error**

Message retention in the [async queue](#) reaches the upper limit.

Note:

The DLQ feature is currently in beta test. If you want to try it out, please [submit a ticket](#) to apply for the activation of CMQ.

DLQ Message Attributes

RequestId: (string) unique identifier of the event call request

ErrorCode: (numeric) error code status

ErrorMessage: (string) error message

When the DLQ delivers a message to CMQ, it encapsulates the attribute information and event information in a JSON request body in the following format:

```
{
  "RequestId": "b615b896-d197-47d7-8919-xxx",
  "ErrorCode": -1,
  "ErrorMessage": "Traceback (most recent call last):\\n File
\\\"/var/user/index.py\\\", line 5, in main_handler\\n if 'key1' in
event.keys():\\nNameError: global name 'event' is not defined",
  "Body": {
    "AppId": xxx,
    "Uin": "xxx",
    "SubAccountUin": "xxx",
    "RequestSource": "TRIGGER_TIMER",
    "FunctionName": "tabortest",
    "Namespace": "default",
    "Qualifier": "$DEFAULT",
    "InvocationType": "RequestResponse",
    "ClientContext": "
{\\\"Type\\\":\\\"Timer\\\",\\\"TriggerName\\\":\\\"tabortimer\\\",\\\"Time\\\":\\\"2020-
```

```

10-10T01:22:00Z\\",\\"Message\\":\\"\\\\"}\\",
  "LogType": "",
  "TimeStampForInvoker": "160229310xxx",
  "RequestId": "b615b896-d197-47d7-8919-xxx",
  "PushTime": "2020-10-10T09:22:00.061824591+08:00",
  "RetryNum": 2,
  "Ttl": 0
}
}

```

Structure	Content
AppId	APPID.
Uin	Root account ID.
SubAccountUin	Sub-account ID of the creator (this field may return null, indicating that no valid values can be obtained).
RequestSource	Trigger request source.
FunctionName	Function name.
Namespace	Namespace.
Qualifier	Version/Alias of the trigger function.
ClientContext	Parameters used to run the function, which are passed in JSON format. For the maximum parameter length, please see Limits .
TimeStampForInvoker	The millisecond timestamp when the function is invoked.
RequestId	Unique ID of request. Each request returns a unique ID. The <code>RequestId</code> is required to troubleshoot issues.
PushTime	Time when the message is pushed to CMQ.
RetryNum	Number of retries (0-2).
Ttl	Retention time of the async queue event.

Directions

Creating DLQ

Note:

SCF currently supports a CMQ topic or queue as the DLQ for your choice.

The DLQ of a function alias will follow the DLQ of the primary version, i.e., the first DLQ selected and configured when the alias is created in the console.

1. Log in to the [CMQ console](#) and create a DLQ.

CMQ topics support filtering by tag or route match. To ensure that your subscribers can receive all error messages, when adding a subscriber, please leave the tag filter **empty** and enter **"#"** for the `BindingKey` filter.

2. Log in to the [Serverless console](#) and create a function.

3. Configure the DLQ.

You can configure the DLQ on the **Create Function** or **Configure Function** page.

Monitoring DLQ

When using a DLQ, permission errors, incorrect resource configurations, or message sizes exceeding the size limit of the target queue or topic will cause DLQ delivery failures. You can query the "number of failed deliveries to DLQ" in the function monitoring information.

1. Log in to the [Serverless console](#) and select **Function Service** on the left sidebar.

2. Select the region of the function for which to monitor the DLQ at the top of the page and click the target function in the list to enter the function details page.

3. On the function details page, click **Monitoring information** to view the number of failed deliveries to the DLQ.

Connecting SCF to Database

Last updated : 2025-03-25 10:17:54

Overview

You can quickly connect to your local or TencentDB databases by writing code in SCF. This document describes how to use an existing SDK to connect to a [TencentDB for MySQL](#) database in the SCF function code and perform operations such as insertion and query in the database. [TDSQL-C](#) and [TDSQL for MySQL](#) databases can also be connected, and you can perform relevant operations as needed.

Note:

You can also use Serverless Framework components to deploy databases and functions. For more information, see [Serverless Application Center](#).

Prerequisites

You have registered a [Tencent Cloud account](#) and completed [identity verification](#).

Interconnect network environments:

For **self-built databases (non-TencentDB databases)**, you need to **enable public network access** first before you can connect to them; otherwise, the connection may fail due to the lack of network connectivity.

For **TencentDB databases**, it is necessary to ensure that the function and database are in the same VPC.

Directions

You can follow the steps below to connect to and manage your TencentDB database in the function code.

Creating VPC

Note:

You can skip this step for self-built databases.

Follow the steps below to create a VPC and subnet. For more information, see [Building Up an IPv4 VPC](#).

1. Log in to the [VPC console](#).
2. Select the region of the VPC at the top and click **+Create**.
3. In the **Create VPC** pop-up window, enter the VPC information, initial subnet name, and region based on the following information as shown below:

Create VPC

VPC information

RegionSouth China(Guangzhou)

Name

IPv4 CIDR Block

10.0

0

.0.0/16

ⓘ Cannot be modified after creation

For better usage of VPC, it's recommended to have a proper [network structure](#).

Advanced Options

▶

Subnet Information

Subnet Name

IPv4 CIDR Block

10.0.0.0/24

IP地址剩余253个

Availability Zone

Guangzhou Zone 1

ⓘ

Associated route table

Default

ⓘ

Advanced Options

▶

OK

Close

4. Click **OK**.

Creating database instance

Note:

You can skip this step for self-built databases.

The following steps take [TDSQL-C](#) as an example to describe how to quickly create a MySQL database.

Note:

For other types of databases, see corresponding product documents:

[TencentDB for MySQL](#)

[TencentDB for PostgreSQL](#)

[TencentDB for Redis](#)

©2013-2025 Tencent Cloud International Pte. Ltd.

Page 40 of 64

1. Log in to the TDSQL-C purchase page, select the deployment region, AZ, database specification, and other information, and click **Buy Now**.
2. After the purchase is completed, you will be redirected to the cluster list. After the status of the cluster becomes **Running**, it can be used normally as shown below:

Cluster ID/Name	Cluster ...	Compatible Databa...	AZ	read-write address	readonly address	Billing...	Project	Expiration Time	Instan...	Operation
cynosdbmysql-nr9lccde	Running	MySQL 5.7	Shangha...	(Private) 10.0.0.12:3306	--	Serverless	DEFAULT PR...	--	1	Log In Manage

3. Click the cluster ID to enter the cluster details page. You can modify configurations, manage accounts, set security groups, and perform other operations for your database cluster as shown below.

Basic Info	
Cluster Name: cynosdbmysql-	Cluster ID: cynosdbmysql-
Status: Running	Network: fortdsq/fortdsq Change Network
Project: DEFAULT PROJECT Switch to Another Project	Tag:
Region/AZ: East China (Shanghai)/Shanghai Zone 2	

Configuration Info	
Compatible Database: MySQL 5.7	Database Storage (Used/Total): 0 B / 5T (Max storage supported by the current specification: 5T)
Database Version: 2.0.14 Upgrade	

Connection Info	
Private Read-Write Address:	Public Read-Write Address: Enable

Billing Info	
Compute Billing Mode: Serverless	Storage Billing: Pay as You Go
Creation Time: 2021-11-03 16:52:13	Expiration Time: --

Creating a function

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Write your business code and connect to the database through an existing SDK or the SCF DB SDK for MySQL tool encapsulated by SCF by following the normal way of connecting to the database. Here, the Node.js function is used as an example. For other languages, see the [function code samples](#) below.

Note:

To use an existing SDK, you need to install the dependency package first. For more information, see [Dependency Installation](#).

```

exports.main_handler = async (event, context, callback) => {
  var mysql      = require('mysql2');
  var connection = mysql.createConnection({
    host      : process.env.HOST,
    user      : process.env.USER,
    password  : process.env.PASSWORD
  });
  connection.connect();
  connection.query('SELECT 1 + 1 AS solution', function (error, results, fields) {
    if (error) throw error;
    console.log('The solution is: ', results[0].solution);
  });
  connection.end();
}

```

3. Enter the **Function Configuration** page of the function and configure the function as shown below:

3.1 Add an **environment variable** and enter the information by referring to the table below:

Environment Variable	key	value	
	HOST	Please enter the value of environment variable	×
	USER	Please enter the value of environment variable	×
	PASSWORD	Please enter the value of environment variable	×


key	value
HOST	Database address
USER	Database username
PASSWORD	Database password

3.2 Enable VPC and select the same VPC and subnet as those of the database as shown below:

VPC ☒ Enable ⓘ

Default-VPC | vpc-1imv05se | 172.17.0.0/16 ▼

Default-Subnet | subnet-1ijzdkj | 172.17.0.0/20 ▼

 [Create VPC](#)

4. After completing the configuration, save it and invoke your function to connect to and manage your database.

Function code samples

You can refer to the following code samples to create functions and configure corresponding [environmental variables](#):

Python

Node.js

PHP

Java

In Python, you can use the built-in **pymysql** dependency package in the SCF environment to connect to the database.

The sample code is as follows:

```
# -*- coding: utf8 -*-
from os import getenv
import pymysql
from pymysql.err import OperationalError
mysql_conn = None
def __get_cursor():
    try:
        return mysql_conn.cursor()
    except OperationalError:
        mysql_conn.ping(reconnect=True)
        return mysql_conn.cursor()
def main_handler(event, context):
    global mysql_conn
    if not mysql_conn:
        mysql_conn = pymysql.connect(
            host      = getenv('DB_HOST', '<YOUR DB HOST>'),
            user      = getenv('DB_USER', '<YOUR DB USER>'),
            password   = getenv('DB_PASSWORD', '<YOUR DB PASSWORD>'),
            db        = getenv('DB_DATABASE', '<YOUR DB DATABASE>'),
            port      = int(getenv('DB_PORT', '<YOUR DB PORT>')),
            charset    = 'utf8mb4',
            autocommit = True
        )

    with __get_cursor() as cursor:
        cursor.execute('select * from employee')
        myresult = cursor.fetchall()
        print(myresult)
        for x in myresult:
            print(x)
```

Node.js allows you to use a connection pool for connection, which supports automatic reconnection to effectively avoid connection unavailability due to connection release by the SCF underlying layer or database. The sample code is as follows:

Note:

Before using a connection pool, you need to install the **mysql2** dependency package first. For more information, see [Dependency Installation](#).

```
'use strict';
const DB_HOST      = process.env[`DB_HOST`]
const DB_PORT      = process.env[`DB_PORT`]
const DB_DATABASE  = process.env[`DB_DATABASE`]
const DB_USER      = process.env[`DB_USER`]
const DB_PASSWORD  = process.env[`DB_PASSWORD`]
const promisePool = require('mysql2').createPool({
  host      : DB_HOST,
  user      : DB_USER,
  port      : DB_PORT,
  password  : DB_PASSWORD,
  database  : DB_DATABASE,
  connectionLimit : 1
}).promise();

exports.main_handler = async (event, context, callback) => {
  let result = await promisePool.query('select * from employee');
  console.log(result);
}
```

In PHP, you can use the **pdo_mysql** or **mysqli** dependency package for data connection. The sample code is as follows:

```
<?php
function handler($event, $context) {
  try{
    $pdo = new PDO('mysql:host= getenv("DB_HOST");dbname= getenv("DB_DATABASE"),get
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  }catch(PDOException $e){
    echo 'Databases connection failed: '.$e->getMessage();
    exit;
  }
}

<?php
function main_handler($event, $context) {
  $host = "";
  $username = "";
  $password = "";

  // Create a connection
```

```
$conn = mysqli_connect($servername, $username, $password);

// Test the connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";

mysqli_close($conn);
echo "Disconnected";
}
?>
```

1. Please install the following dependencies as instructed in [Dependency Installation](#).

```
<dependencies>
  <dependency>
    <groupId>com.tencentcloudapi</groupId>
    <artifactId>scf-java-events</artifactId>
    <version>0.0.2</version>
  </dependency>
  <dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>3.2.0</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.11</version>
  </dependency>
</dependencies>
```

2. Use HikariCP for connection. The sample code is as follows:

```
package example;

import com.qcloud.scf.runtime.Context;
import com.qcloud.services.scf.runtime.events.APIGatewayProxyRequestEvent;
import com.qcloud.services.scf.runtime.events.APIGatewayProxyResponseEvent;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;
public class Http {
    private DataSource dataSource;
    public Http() {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl("jdbc:mysql://" + System.getenv("DB_HOST") + ":" + System.
        config.setUsername(System.getenv("DB_USER"));
        config.setPassword(System.getenv("DB_PASSWORD"));
        config.setDriverClassName("com.mysql.jdbc.Driver");
        config.setMaximumPoolSize(1);
        dataSource = new HikariDataSource(config);
    }

    public String mainHandler(APIGatewayProxyRequestEvent requestEvent, Context con
        System.out.println("start main handler");
        System.out.println("requestEvent: " + requestEvent);
        System.out.println("context: " + context);

        try (Connection conn = dataSource.getConnection(); PreparedStatement ps = c
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                System.out.println(rs.getInt("id"));
                System.out.println(rs.getString("first_name"));
                System.out.println(rs.getString("last_name"));
                System.out.println(rs.getString("address"));
                System.out.println(rs.getString("city"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        APIGatewayProxyResponseEvent apiGatewayProxyResponseEvent = new APIGatewayP
        apiGatewayProxyResponseEvent.setBody("API GW Test Success");
        apiGatewayProxyResponseEvent.setIsBase64Encoded(false);
        apiGatewayProxyResponseEvent.setStatusCode(200);

        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "text");
        headers.put("Access-Control-Allow-Origin", "*");
        apiGatewayProxyResponseEvent.setHeaders(headers);

        return apiGatewayProxyResponseEvent.toString();
    }
}
```

SCF DB SDK for MySQL

For ease of use, the SCF team encapsulated the code related to connection pools in Node.js and Python as SCF DB SDK for MySQL. With this SDK, you can connect to [MySQL](#), [TDSQL-C](#), or [TDSQL for MySQL](#) databases and performs operations such as insertion and query.

SCF DB SDK for MySQL has the following features:

It can automatically initialize the database client from environment variables.

It can maintain a persistent database connection globally and handle reconnection after disconnection.

The SCF team will continuously check issues to ensure that the database connection is available, so you don't need to pay attention to connection issues.

The sample code is as follows:

Node.js SDK

Python SDK

```
'use strict';
const database = require('scf-nodejs-serverlessdb-sdk').database;
exports.main_handler = async (event, context, callback) => {
  let pool = await database('TESTDB2').pool()
  pool.query('select * from coffee', (err, results) => {
    console.log('db2 callback query result:', results)
  })
  // no need to release pool

  console.log('db2 query result:', result)
}
```

```
from serverless_db_sdk import database
def main_handler(event, context):
    print('Start Serverless DB SDK function')
    connection = database().connection(autocommit=False)
    cursor = connection.cursor()
    cursor.execute('SELECT * FROM name')
    myresult = cursor.fetchall()
    for x in myresult:
        print(x)
```

Note:

1. Python 3.6, Python 2.7, Node.js 12.16, and Node.js 10.15 have built-in SCF DB SDK for MySQL, so no additional installation is required.

2. For other Node.js versions, please refer to [Dependency Installation](#) to install `scf-nodejs-serverlessdb-sdk`.
3. For specific usage of the SDK for Node.js, see [SCF DB SDK for MySQL](#).

FAQs

How do I manage database connections more efficiently under the operating mechanism of SCF?

Each SCF request actually runs on a container that can be reused for a period of time when there are continuous requests. A database connection is better to be established when the container is initialized, i.e., corresponding to the global part of the function code. After the database connection is established, it can be reused during the existence of the container and will be closed when the container is released. Please avoid frequent database connections and disconnections inside the entry function, as they affect the performance. To ensure the database connection availability, a connection check can be performed inside the entry function.

We recommend you use the database connection pool for container database connection management and set the minimum number of connections to 1.

How do I perform database connection management in high-concurrency scenarios?

In high function concurrency scenarios, the number of concurrent connections may exceed the maximum number of database connections. You can refer to the following solutions for handling:

Increase the maximum number of database connections.

Set the maximum dedicated concurrency quota for functions and limit the number of concurrent function connections to be less than the maximum number of database connections.

TencentDB for MySQL provides the [database proxy](#) feature. Requests arriving at the proxy address are all relayed through the proxy cluster to access the source and replica nodes of the database. Read/Write separation is implemented, so that read requests are forwarded to read-only instances, which lowers the load of the source database.

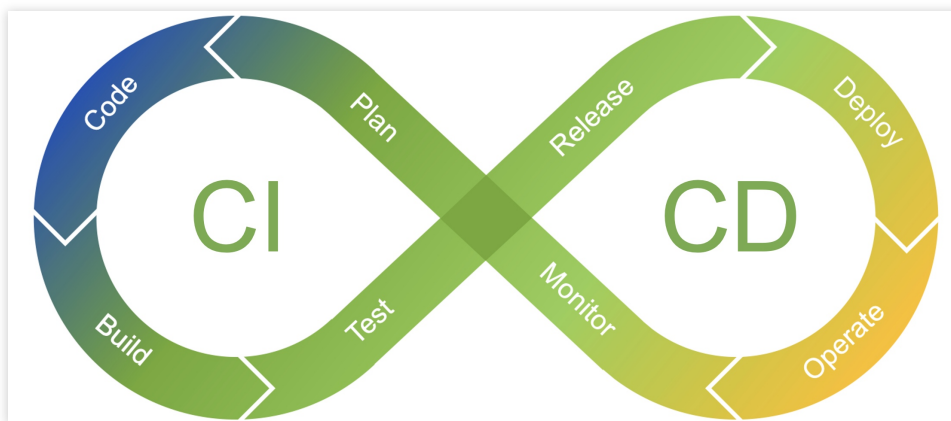
Automated Deployment

Last updated : 2024-12-02 19:46:29

Overview

As agile development and DevOps get more popular, CI/CD has become an indispensable best practice by almost all developers. It aims to deliver practical software programs more quickly.

CI/CD



CI/CD strengths

Reduced release cycle

Reduced risks

Improved code quality

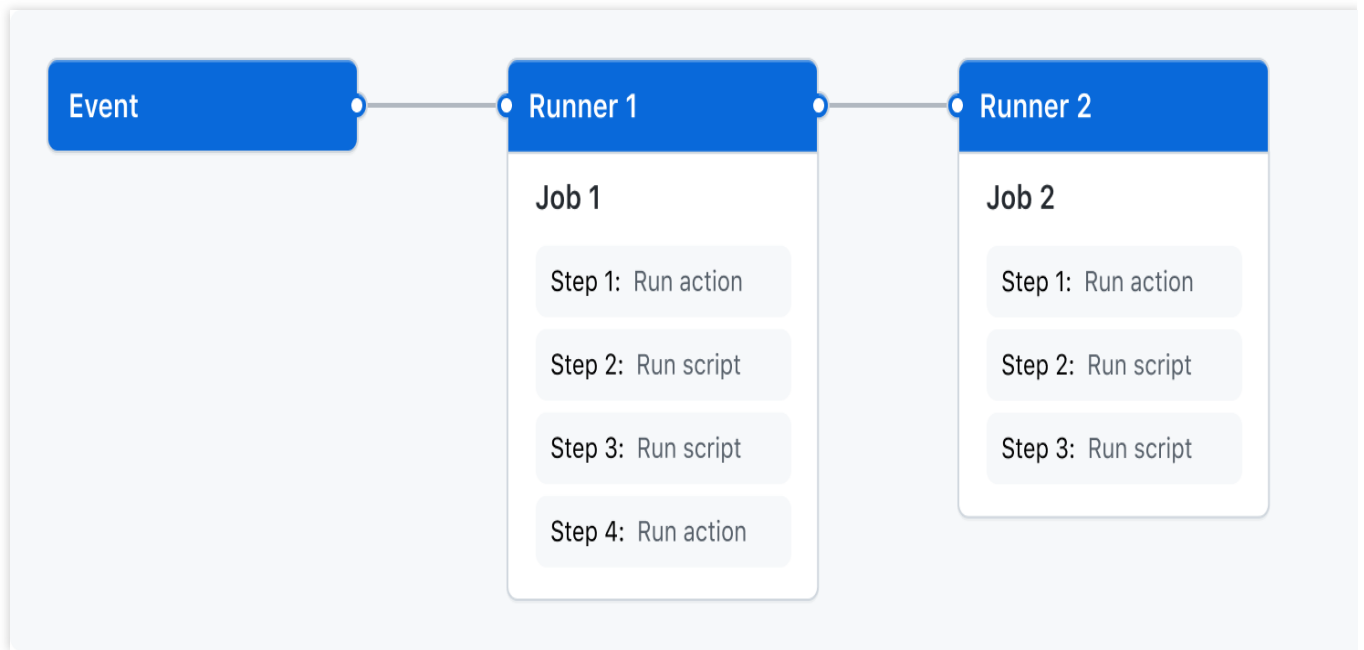
More efficient feedback loop

Visual process

This document uses GitHub, Jenkins, and CODING as examples to describe how to use [Serverless Framework CLI](#) to quick build CI/CD for your SCF project.

Automated Deployment Based on GitHub

[GitHub Actions](#) is an automated software development workflow launched by GitHub. It uses actions to execute any tasks, including CI/CD.



Prerequisites

The SCF project has been hosted in GitHub.

The project needs to contain the `serverless.yml` configuration file used for execution in Serverless Framework CLI.

To use an HTTP-triggered function, place the `scf_bootstrap` file in the root directory of your project.

Directions

Note:

SCF has released [Tencent Serverless Action](#) in GitHub.

1. Search for **Tencent Serveless Action** in GitHub.

Pull requests Issues Marketplace Explore

Marketplace / Search results

Types

Apps

Actions

Categories

API management

Chat

Code quality

Code review

Continuous integration


Q tencent serverless X Sort: Best Match

Actions

An entirely new way to automate your development workflow.

1 result for "tencent serverless" filtered by Actions x

Actions

 **Tencent Serverless Action**
By woodyyan
Github action for serverless framework

2. On the **Actions** tab, select **Set up a workflow yourself** as shown below.

Issues Pull requests Actions Projects Wiki Security Insights Settings

Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself →](#)

Q Search workflows

Suggested for this repository

Python Package using Anaconda
By GitHub Actions

Create and test a Python package on multiple Python versions using Anaconda for package management.

Configure Python

Publish Python Package
By GitHub Actions

Publish a Python Package to PyPI on release.

Configure Python

Django
By GitHub Actions

Build and Test a Django Project

Configure Python

3. How to use:

If you are familiar with the action usage, you can use the following command, which encapsulates the steps of installing Serverless Framework and running the deployment command.

```
- name: serverless scf deploy
  uses: woodyyan/tencent-serverless-action@main
```

If you are new to actions, you can select one of the following YAML code samples based on your programming language (Python, Java, or Node.js):

Python

Java

NodeJS

```
# When the code is pushed to the `main` branch, the current workflow will be executed
# For more information on configuration, visit https://docs.github.com/en/actions/guides
name: deploy serverless scf
on: # Configuration of the event and branch listened on
  push:
    branches:
      - main
jobs:
  deploy:
    name: deploy serverless scf
    runs-on: ubuntu-latest
    steps:
      - name: clone local repository
        uses: actions/checkout@v2
      - name: deploy serverless
        uses: woodyyan/tencent-serverless-action@main
    env: # Environment variable
      STAGE: dev # Your deployment environment
      SERVERLESS_PLATFORM_VENDOR: tencent # The serverless platform is `aws` by default
      TENCENT_SECRET_ID: ${ secrets.TENCENT_SECRET_ID } # `secret ID` of your Tencent Cloud account
      TENCENT_SECRET_KEY: ${ secrets.TENCENT_SECRET_KEY } # `secret key` of your Tencent Cloud account

name: deploy serverless scf
on: # Configuration of the event and branch listened on
  push:
    branches:
      - main
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
```

```
steps:
  - uses: actions/checkout@v2
  - name: Set up JDK 11
    uses: actions/setup-java@v2
    with:
      java-version: '11'
      distribution: 'temurin'
      server-id: github # Value of the distributionManagement/repository/id file
      settings-path: ${github.workspace} # location for the settings.xml file
  - name: Build with Gradle # Use this for a Gradle project
    uses: gradle/gradle-build-action@937999e9cc2425eddc7fd62d1053baf041147db7
    with:
      arguments: build
  - name: Build with Maven # Use this for a Maven project
    run: mvn -B package --file pom.xml
  - name: Create zip folder # This step is used to store JAR and `scf_bootstrap`
    run: mkdir zip
  - name: move jar and scf_bootstrap to zip folder # This step is used to move
    run: cp ./build/libs/XXX.jar ./scf_bootstrap ./zip
  - name: deploy serverless
    uses: woodyyan/tencent-serverless-action@main
    env: # Environment variable
      STAGE: dev # Your deployment environment
      SERVERLESS_PLATFORM_VENDOR: tencent # The serverless platform is `aws` by
      TencentSecretId: ${secrets.TENCENT_SECRET_ID} # `secret ID` of your
      TencentSecretKey: ${secrets.TENCENT_SECRET_KEY} # `secret key` of y

# When the code is pushed to the `main` branch, the current workflow will be executed
# For more information on configuration, visit https://docs.github.com/cn/actions/g
name: deploy serverless scf
on: # Configuration of the event and branch listened on
  push:
    branches:
      - main
jobs:
  deploy:
    name: deploy serverless scf
    runs-on: ubuntu-latest
    steps:
      - name: clone local repository
        uses: actions/checkout@v2
      - name: install dependency
        run: npm install
      - name: build
        run: npm build
      - name: deploy serverless
```

```
uses: woodyyan/tencent-serverless-action@main
env: # Environment variable
  STAGE: dev # Your deployment environment
  SERVERLESS_PLATFORM_VENDOR: tencent # The serverless platform is `aws` by
  TENCENT_SECRET_ID: ${ secrets.TENCENT_SECRET_ID } # `secret ID` of your
  TENCENT_SECRET_KEY: ${ secrets.TENCENT_SECRET_KEY } # `secret key` of y
```

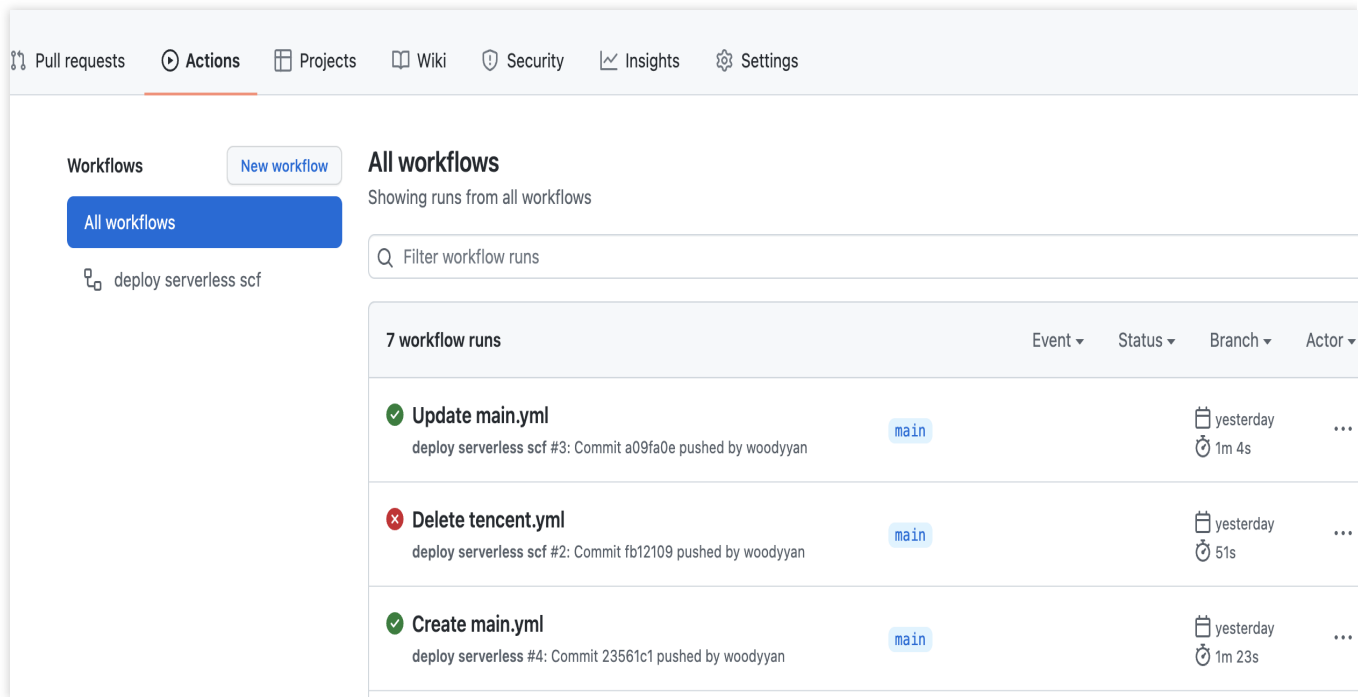
`TENCENT_SECRET_ID` and `TENCENT_SECRET_KEY` are required during the deployment. You need to configure such variables in `Secrets` in the GitHub code repository settings as follows:

The screenshot shows the GitHub repository settings page for a repository. The left sidebar contains navigation links: Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Settings' tab is selected. On the left sidebar of the settings page, the 'Secrets' option is highlighted with a red box. The main content area is titled 'Actions secrets' and includes a 'New repository secret' button. Below this, there are two sections: 'Environment secrets' and 'Repository secrets'. The 'Environment secrets' section states 'There are no secrets for this repository's environments.' The 'Repository secrets' section contains a table with two entries, both highlighted with red boxes:

Secret Name	Updated	Actions
TENCENT_SECRET_ID	Updated 4 days ago	Update Remove
TENCENT_SECRET_KEY	Updated 4 days ago	Update Remove

You can get the Tencent Cloud secret ID and key from [CAM](#).

4. After the configuration, every time the code is pushed, the deployment process will be automatically triggered, and you can view the execution result and error logs on the **Actions** tab in real time as shown below:



In addition, you can add testing to the process for steps such as security check and release based on your project needs.

Automated Deployment Based on Jenkinsfile

Jenkinsfile is commonly used on Jenkins and CODING platforms. After configuring the Jenkinsfile, you can complete automated deployment on such platforms.

Prerequisites

You have hosted your SCF project onto platforms such as CODING, GitHub, GitLab, and Gitee.

The project needs to contain the `serverless.yml` configuration file used for execution in Serverless Framework CLI.

To use an HTTP-triggered function, place the `scf_bootstrap` file in the root directory of your project.

Directions

This document provides Jenkinsfile code in three programming languages: Python, Java, and Node.js. Carefully read the comments.

```
pipeline {
  agent any
  stages {
    stage('check out') {
      steps {
```

```

        checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
            userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTI
    }
}

stage('Package'){ // This stage is only used for a Java project
steps{
    container("maven") {
        echo 'Package start'
        sh "mvn package" // This line is used for a Java Maven project
            sh "./gradlew build" // This line is used for a Java Gradle pro
            sh "mkdir zip" // This line is used to store JAR and `scf_boots
        sh "cp ./build/libs/XXX.jar ./scf_bootstrap ./zip" // This line is used t
    }
}

stage('Install dependency') {
    steps {
        echo 'Installing dependency...'
        sh 'npm i -g serverless'
        sh 'npm install' // This line is used for a Node.js project
        echo 'Dependency installed.'
    }
}

stage('deploy') {
    steps {
        echo 'deploying...'
        withCredentials([
            cloudApi(
                credentialsId: "${env.TENCENT_CLOUD_API_CRED}",
                secretIdVariable: 'TENCENT_SECRET_ID',
                secretKeyVariable: 'TENCENT_SECRET_KEY'
            ),
        ]) {
            // Generate the credential file
            sh 'echo "TENCENT_SECRET_ID=${TENCENT_SECRET_ID}\\nTENCENT_SECRET_KEY=
            // Deploy
            sh 'sls deploy --debug'
            // Remove the credential
            sh 'rm .env'
        }
        echo 'deployment complete'
    }
}
}
}
}

```

You can use the above Jenkinsfile to quickly configure CI/CD on platforms such as Jenkins and CODING.

Note:

You can get Tencent Cloud `TENCENT_SECRET_ID` and `TENCENT_SECRET_KEY` required during the deployment from [CAM](#).

Cloud Function Status Code

Last updated : 2024-12-02 19:46:29

If an error code is returned after the function is executed, you can find the cause and solution for the error code by referring to the following table.

Status Code and Status Message	Description	Solution
200 Success	Successful	-
400 InvalidParameterValue	The request event passed in by the event execution function is not of the JSON type.	Make modifications as instructed in Introduction and InvokeFunction and try again.
401 InvalidCredentials	Permission authentication failed.	Your account does not have the permission to manipulate this function. Make modifications as instructed in the authorization description in Permission Management Overview and try again.
402 ServiceSuspended	The service is temporarily suspended.	Your SCF service is temporarily suspended. You can refer to Payment Overdue to make changes and try again later.
404 InvalidSubnetID	The subnet ID in the network configuration of the function is exceptional.	Check whether the network configuration of the function is correct and whether the subnet ID is valid.
405 ContainerStateExited	The container exits.	Check your image or bootstrap file to see whether it can be properly started locally. If so, check whether the use limits of SCF are followed; for example, RootFS is read-only and only <code>/tmp</code> is writable. Local debugging command: <pre>docker run -itd --read-only -v /tmp:/tmp</pre>
406 RequestTooLarge	The <code>event</code> input parameter of the function,	The request event size exceeds the quota limit, which is 6 MB for sync request events or 128 KB for async ones. Adjust the request event size accordingly and try again.

	i.e., the request event size of the function, exceeds the quota limit .	
407 The size of response exceeds the upper limit (6MB)	The size of function response exceeds the upper limit of 6 MB.	Please adjust it and try again.
410 InsufficientBalance	Insufficient account balance	The SCF service is suspended because the Tencent Cloud account has overdue payments. Top up and try again.
429 ResourceLimit	The container request rate is too high and exceeds the limit due to concurrency surges.	<p>The default maximum speed of elastic concurrency expansion (function burst) for each account is 500 concurrent instances per region per minute. During a sudden concurrency surge, if there are not enough containers to carry the requests, a large number of container request actions will be triggered, and this message will be returned when the account limit is exceeded.</p> <p>After assessing the function concurrency, configure provisioned concurrency for the function and prepare containers in advance to avoid sudden concurrency surges from causing the container request speed to exceed the limit.</p> <p>If assessment shows that the provisioned concurrency cannot meet the needs of your business scenario, you can purchase a function package to increase the function burst in the region.</p>
430 User code exception caught	A user code execution error occurs.	Please check the code error stack information in the invocation log provided by the SCF console, make modifications, and try again.
432 ResourceLimitReached	The account-level or region-level concurrency limit is reached.	<p>For a function with a reserved quota configured, if the function concurrency exceeds the quota, <code>Function [xxx] concurrency exceeded reserved quota xxx MB</code> will be returned. You can assess your business needs and increase the quota or refer to Concurrency Overrun.</p> <p>For a function with no reserved quota configured, if the concurrency quota actually used by the function exceeds the region-level unused concurrency quota, <code>Function [xxx] concurrency exceeded region unreserved quota xxx MB</code> will be returned. You can assess your business needs</p>

		and configure a reserved quota for the function. If the remaining available quota in the corresponding region cannot meet your business needs, you can purchase a function package to increase the total concurrency quota in the region.
433 TimeLimitReached	Function execution is not completed after the execution timeout period elapses.	Please check whether a large number of time-consuming operations exists in the service code. Set a longer timeout period on the Function configuration page. If the current timeout period has been set to the maximum value, you can create an async function as instructed in Async Execution to get a function execution duration of up to 24 hours. This status code will trigger instance repossession .
434 MemoryLimitReached	The memory limit is reached.	Please check the code logic to see whether there is a memory leak. Please increase the memory configuration on the Function Configuration page, or apply for a large memory on the Function Memory Configuration page to get up to 120 GB of function execution memory. This status code will trigger instance repossession .
435 FunctionNotFound	The function is not found.	Please check whether the input parameters match the information of the function to be invoked. Please check whether the function exists when it is invoked and whether there is any deletion action that causes the function to be invoked after deletion.
436 InvalidParameterValue	The parameter passed in for <code>invoke</code> does not conform to the specification.	The parameter does not conform to the specification. Modify it as instructed in Introduction and try again.
437 HandlerNotFound	The function package is loaded incorrectly.	Please check whether the compressed package is in normal status. The function execution entry file is not found. Please make sure that the entry file is in the root directory of the decompressed code package. Check the entry file and execution method in the code package.
438 FunctionStatusError	The function is exceptional or the SCF service is suspended.	The function is invoked in an exceptional state. Please wait for the function status to become normal and try again. The SCF service is suspended because the Tencent Cloud account has overdue payments. Top up and try again.

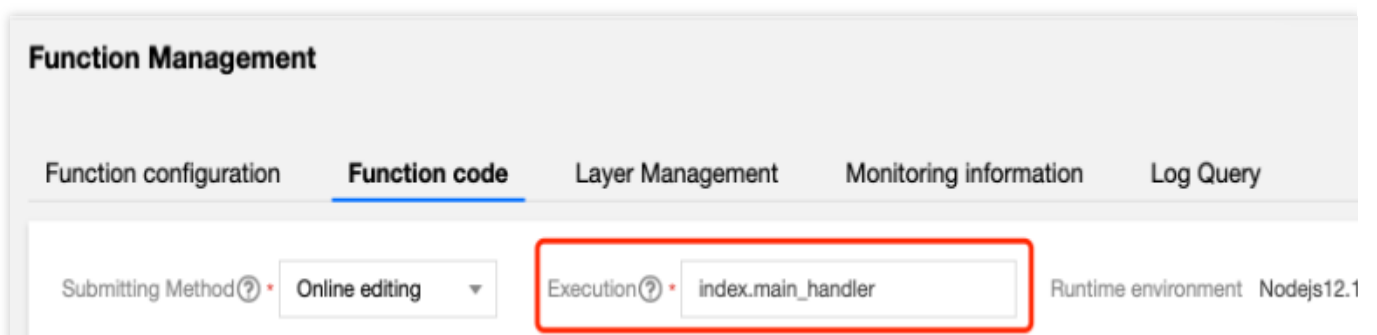
439 User process exit when running	The user process exits accidentally.	Based on the error message, find out the cause, fix the function code, and try again. This status code will trigger instance repossession .
441 UnauthorizedOperation	CAM authentication fails.	Check whether the CAM authentication parameters for the function invoker are passed correctly. For more information, see the authorization description in Permission Management Overview .
442 QualifierNotFound	The specified version is not found.	The function version does not exist. Check the function version and try again.
443 UserCodeError	A user code execution error occurs.	Based on the error log on the console, check the error stack of the code and see whether the code can be executed properly.
444 PullImageFailed	Image pull fails.	Check the integrity and validity of the selected image and try again; for example, check whether it can be downloaded locally. If the problem persists, submit a ticket .
445 ContainerInitError	Container start fails.	Container start fails. Check whether your bootstrap file has been uploaded successfully and ensure that the invocation path is correct. For an image deployment-based function, check whether the <code>Command</code> or <code>Args</code> parameter passed in the console is in the correct format. For more information, see Usage . For a code deployment-based function, please check whether your bootstrap file has been uploaded successfully and ensure that the invocation path is correct.
446 PortBindingFailed	Port listening fails.	The container initialization duration exceeds the initialization timeout period . Please check whether the listening port is 9000. Please check whether all the files in the code package or container image are required files. Appropriate streamlining can improve the initialization speed of the container. Please check whether there are any exceptions or time-consuming business logic in the initialization code. You can appropriately increase the initialization timeout period and try again.
447 PullImageTimeOut	Image pull times out.	It may be a timeout caused by a large image or network jitters. Minimize the image or increase the initialization timeout period and try again. If the problem persists, submit a ticket .
449	There are no	If the resource type is high-spec CPU or GPU, it can be used

InsufficientResources	resources available at the resource specification selected by this function in the specified region.	with the provisioned concurrency. If the problem persists, submit a ticket .
450 InitContainerTimeout	Container start times out.	The container start duration exceeds the initialization timeout period . Minimize the code or increase the initialization timeout period and try again.
499 RequestCanceled	The function execution request is canceled.	For an asynchronously executed function, if the user cancels the function execution request, this message will be returned. For an HTTP-triggered function, if the timeout period of an API Gateway trigger is less than the sum of the initialization duration and execution duration of the function, this message will be returned. Please check whether there is any exceptionally time-consuming business logic in the code or increase the backend timeout period of the API and try again.
500 InternalError	Internal error	An internal error occurs. Try again later. If the problem persists, submit a ticket .

Concepts

Execution method

The execution method specifies which function in which file is executed first when the cloud function is invoked.



For Go programming, use the **FileName** format, such as `main`.

For Python, Node.js, or PHP programming, use the **FileName.FunctionName** format, such as

`index.main_handler`.

Note that **FileName** does not include the file name extension, and **FunctionName** is the name of the entry **function**. Ensure that the file name extension matches the programming language. For example, for Python programming, the file name extension is `.py`, and for Node.js programming, the file name extension is `.js`. For more information, see "Execution Method" in [Basic Concepts](#).

For Java programming, use the **package.class::method** format, such as `example.Hello::mainHandler`.

For custom runtime, you can ignore the above patterns, and write the execution method in your custom language.

Common Errors and Solutions

Last updated : 2024-12-02 19:46:29

Common errors and solutions:

Error Code	Solution
InvalidParameter.FunctionName	The value of FunctionName is invalid. Please modify it as instructed in API documentation and try again.
InvalidParameterValue.Action	The requests API does not exist. Please modify it as instructed in API documentation and try again.
InvalidParameterValue.CosBucketRegion	The value of CosBucketRegion is invalid. Please modify it as instructed in COS Regions and Access Endpoints and try again.
InvalidParameterValue.DeadLetterConfig	The value of DeadLetterConfig is invalid. The value of Type should be CMQ-TOPIC, CMQ-QUEUE, topic or queue, and Name cannot be left empty.
InvalidParameterValue.Enable	The value of Enable is invalid. It should be OPEN or CLOSE.
InvalidParameterValue.Memory	The value of Memory is invalid. The function runtime memory defaults to 128M. You can set it to 64M, or 128M - 3072M (in increments of 128M).
InvalidParameterValue.OrderBy	The value of OrderBy is invalid. Please modify it as instructed in the API documentation and try again.
InvalidParameterValue.RoutingConfig	The value of RoutingConfig is invalid. Please refer to API documentation .