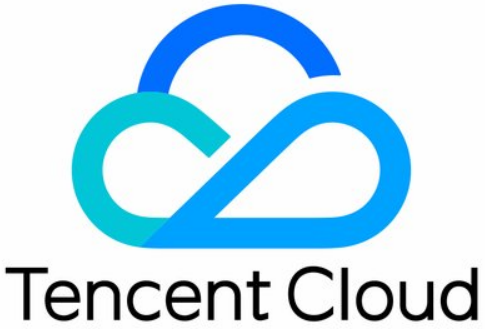
Serverless Cloud Function Development Guide



Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Development Guide
Concepts

Last updated: 2023-09-27 19:52:33

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 aforementioned concepts correspond to the following in typical project development:
> **Execution Method**: Corresponds to the main function of the project, which is the starting point of program execution.
> **Function Input Parameter**: This is the commonly understood function input parameter. However, in the SCF environment, the input parameter of the entry function is a fixed value set by the platform. For more details, see Function Input Parameter.
> **Function Return**: Corresponds to the return value of the main function in the project. Once the function returns, the code execution ends.

Execution Method

When the SCF platform invokes a cloud function, it first looks for the execution method as the entry point to execute the user's code. At this point, users need to set it in the format of **filename.executionMethodName**.
For instance, if the user sets the execution method as `index.handler`, the SCF platform will first look for the `index` file in the code package and start executing the `handler` method in that file.
Within the execution method, users can process the input parameters of the entry function and call any other methods in the code. The execution of an SCF function ends when the entry function is completed or an exception occurs during function execution.

Function input parameters

Function input parameters refer to the content passed to the function when it is triggered and called. Typically, function input parameters include two parts: **event** and **context**. However, depending on the development language and environment, the number of input parameters may vary. For more details, please refer to Development Language Description.

event

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.

Use 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 triggers function execution:
  You can customize a dict type parameter between the caller and the function code. The caller inputs data according to the defined format, and the function code obtains data according to the format.
  Example:
  Define a dict type data structure {"key":"XXX"}. When the caller inputs data {"key":"abctest"}, the function code can obtain the value abctest through event[key].
- Triggering function execution via triggers:
  SCF is integrated with various cloud services such as API Gateway, COS, and Ckafka. You can trigger function execution by binding the corresponding cloud service triggers to the function. When a trigger initiates a function, the event is passed to the function as an event parameter in a predefined, unchangeable format. You can write code according to this format and retrieve information from the event parameter.
  Example:
  When a function is triggered by COS, the specific information of the COS bucket and file is passed to the event parameter in JSON format. By parsing the event information in the function code, you can handle the triggering event.

context

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.

Use Instructions

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

Field	Description
memory_limit_in_mb	Function Memory Configuration
time_limit_in_ms	Function Execution Timeout Duration
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	Tencent Cloud Account APPID of the Function
tencentcloud_uin	Function's Tencent Cloud Account ID

> **Note**
> To ensure compatibility, the context retains the description methods of namespaces at different stages of SCF. The structure of the context will increase with the development iteration of the SCF platform.

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 across various development languages and environments, both the event input parameter and the context input parameter are uniformly encapsulated using the JSON data format.
- Different triggers pass different data structures when triggering functions. For more information, please refer to Function Trigger Description.
- 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	Solutions
Sync triggering	If triggered by API Gateway or the TencentCloud API for sync invocation, the function will be triggered synchronously. For a function triggered synchronously, the SCF platform will not return the trigger result during function execution. After the function is executed, the SCF platform will encapsulate the returned value into JSON format and return it to the invoker.

Async triggering	For a function that is triggered asynchronously, the SCF will return the triggering request ID after receiving the triggering event. After the function is executed, the returned value will be encapsulated into JSON format and stored in the log. 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.

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

Running 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 across various development languages and environments, the function return will be uniformly encapsulated in **JSON data format**. Upon receiving the return value of the function from the aforementioned running environment, the SCF platform will convert the returned data structure into JSON and return the JSON content to the caller.

> ⚠ **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).

Solutions

You can go to the Serverless Console and follow the steps below to perform exception handling tests:

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

Example

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

Note

This function will throw an exception during execution, returning 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!
```

Inherit the `Exception` class

Example

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

Explanation

You can define your own error handling methods within the code, ensuring the robustness and scalability of your application.

Use the `Try` statement to capture errors

Example

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

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

Returns error message

When the user's code logic does not handle exceptions or capture errors, the SCF platform will attempt to capture errors as much as possible. For instance, if a user function suddenly crashes during execution and the platform is unable to capture the error, the system will return a generic error message.
The table below displays some common errors that occur during code execution:

Error Scenario	Error Message
`raise` 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.

Precautions

Given the characteristics of SCF, you must write your function code in a **stateless** style. Local file storage and other state features within the function lifecycle will be destroyed after the function call ends. Therefore, it is recommended to store persistent states in relational databases like TencentDB, object storage like COS, cloud database 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: 2023-09-27 19:53:07

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 VS Code Plugin: Cloud Debugging
- SCF console: Cloud Test

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.

Precautions

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: 2023-09-27 19:53:37

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 on Function Service in the left sidebar.
2. During the process of creating a function, or when editing an existing function, you can add environment variables in the "Environment Variables" section. Environment variables typically appear in the form of `key-value` pairs. Please enter the required environment variable key in the first input box and the required environment variable value in the second input box. Note that the key and value must start with a letter and can only contain letters, numbers, and "_", with a length of not less than 2 and not more than 64 bytes.

Adding an environment variable locally

1. When developing locally, locate the function that requires environment variable configuration in the serverless.yml file and add the Environment configuration item as shown below:

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

# Component parameter configuration
inputs:
name: scfdemo # Cloud function name, default is ${name}-${stage}-${app}
  namespace: default
# 1. Default format: Create a specifically named COS bucket and upload.
  src: ./src
type: event # Function type, default is event (Event type), web (HTTP-triggered
function)
handler: index.main_handler # Entry point (effective when the function type is
event type)
runtime: Nodejs10.15 # Runtime environment, default is Nodejs10.15
region: ap-guangzhou # Function's 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 Variables
variables: # Environment Variable Objects
    TEST1: value1
    TEST2: value2
```

2. Save the serverless.yml file and execute the `scf deploy` command in the command line window to deploy to the cloud.

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

Configured environment variables will be set in the runtime environment of the function during execution. They can be obtained and used in the code by reading system environment variables. Please note that **environment variables cannot be read locally**.
Assuming the key of the environment variable configured for the function is `key`, the following are example codes for reading and printing this environment variable value in various 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');
```

Use case

- **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 start 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 (to use a timer trigger)
PYTHONDONTWRITEBYTECODE	x
PYTHONPATH	/var/user:/opt
CLASSPATH	/var/runtime/java x:/var/runtime/java x/lib/*:/opt (where x is either 8 or 11)
NODE_PATH	/var/user:/var/user/node_modules:/var/lang/node x/lib/node_modules:/opt:/opt/node_modules (where x is either 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 (This built-in variable is available in the Node.js 16.13 environment for event functions, but not for HTTP-triggered Functions)
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 (where x is either 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

Error Types And Retry Policies

Last updated: 2023-09-28 16:07:25

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 has no sufficient 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.

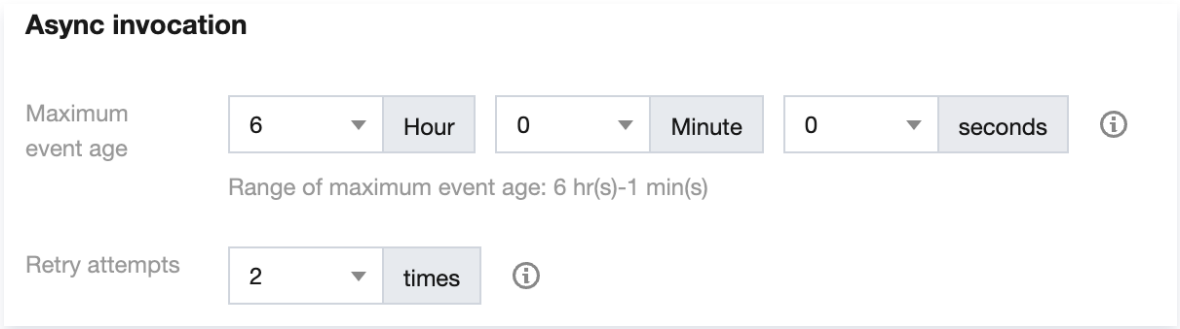
Sync invocation

Sync invocation includes: Sync invocation of Cloud API Trigger, API Gateway Trigger, CKafka Trigger, and CLB Trigger.
Since the error information is directly returned to the user during the sync invocation, the platform will not automatically retry when an error occurs in the sync invocation. The retry policy (whether to retry and how many times to retry) is determined by the caller.

> **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

Async invocation includes: Cloud API Trigger async invocation, COS Trigger, Timer Trigger, and CMQ Topic Trigger, etc. For specific trigger invocation types, please refer to the relevant Trigger Documentation.
You can modify and customize the default **Retry Count** and **Max Wait Time** configurations according to your business needs. These configurations are only applicable to async invocation scenarios.



- **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 Types	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: 2023-09-27 19:55:35

Scenario

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:

- The user code continues to fail after two attempts to retry due to a runtime error.
- Excessive errors and system errors continue to retry beyond a 24-hour period.
- The accumulation of messages in the asynchronous queue reaches its maximum 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 the request. Each request returns a unique ID. The `RequestId` 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.

Instructions

Creating DLQ

> **Note**
> SCF currently supports both the topic and queue modes of CMQ as dead letter queues, which you can configure as needed. The dead letter queue of a function alias follows the main version's dead letter queue, which is the first dead letter queue selected and configured when creating an alias in the console.

1. Log in to the CMQ Message Queue Console to create your dead letter queue. The topic mode of CMQ supports two filtering schemes: tag filtering and routing matching. To ensure that your subscribers can receive all error information, please set the tag filter to **empty** and the BindingKey filter to **"#"** when adding subscribers.
2. Log in to the Serverless Console to create your function.
3. Configure the dead letter queue. You can configure the dead letter queue on the "Create Function" page or the "Function Configuration" 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** from the left navigation bar.
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 dead letter queue.

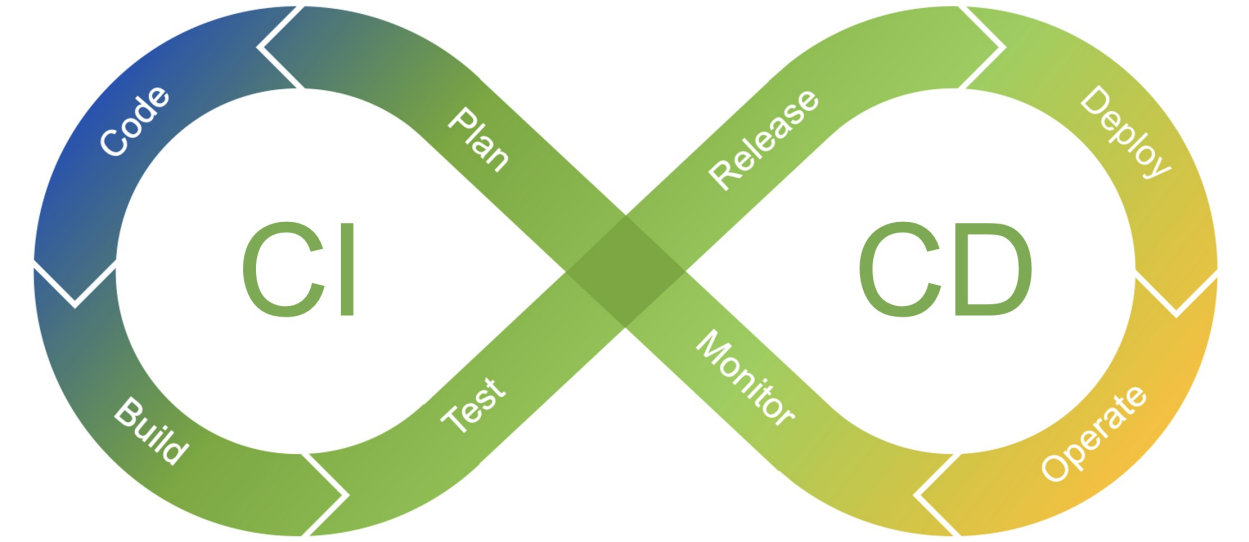
Automated Deployment

Last updated: 2023-09-27 19:57:06

Feature 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



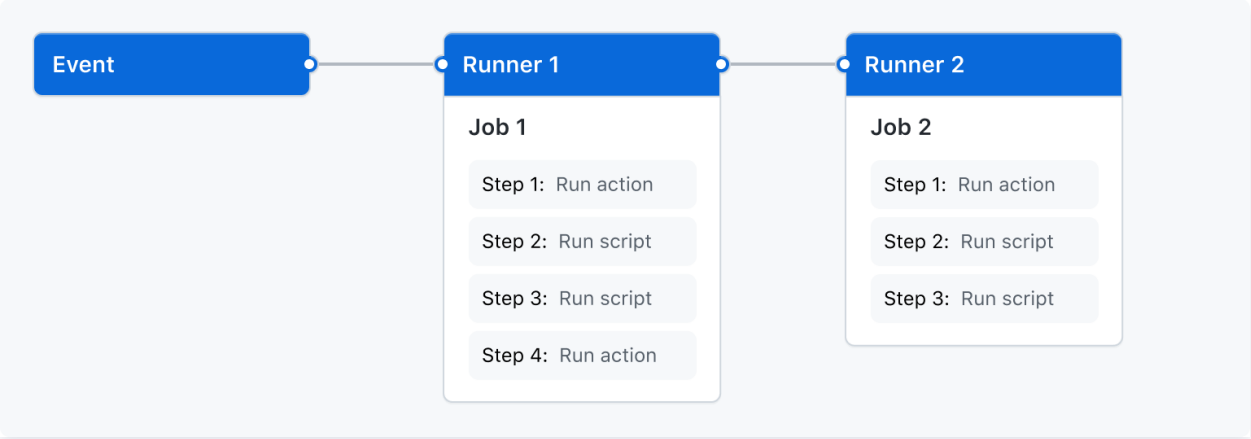
CI/CD strengths

- Reduced release cycle
- Reduced risks
- Improved code quality
- More efficient feedback loop
- Visual process

This article illustrates how to quickly set up CI/CD for Serverless projects using Serverless Cloud Framework, taking GitHub, Jenkins, and CODING as examples.

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.



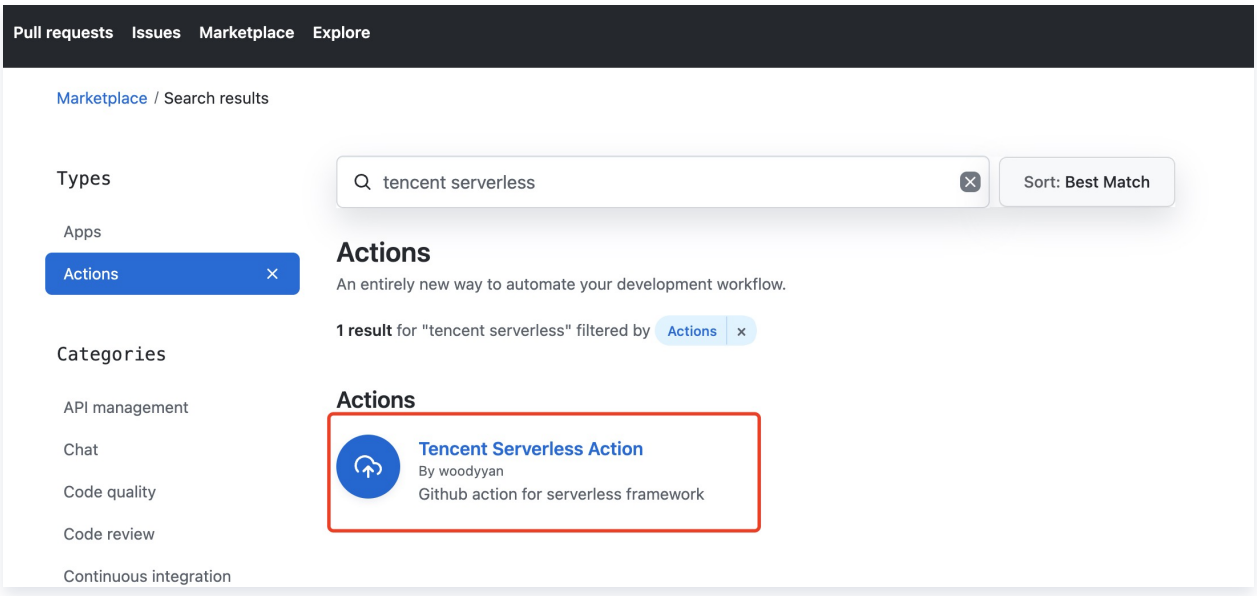
Preparations

- The SCF project has been hosted in GitHub.
- The project should include the `serverless.yml` configuration file required for executing the Serverless Cloud Framework.
- To use an HTTP-triggered function, place the scf_bootstrap file in the root directory of your project.

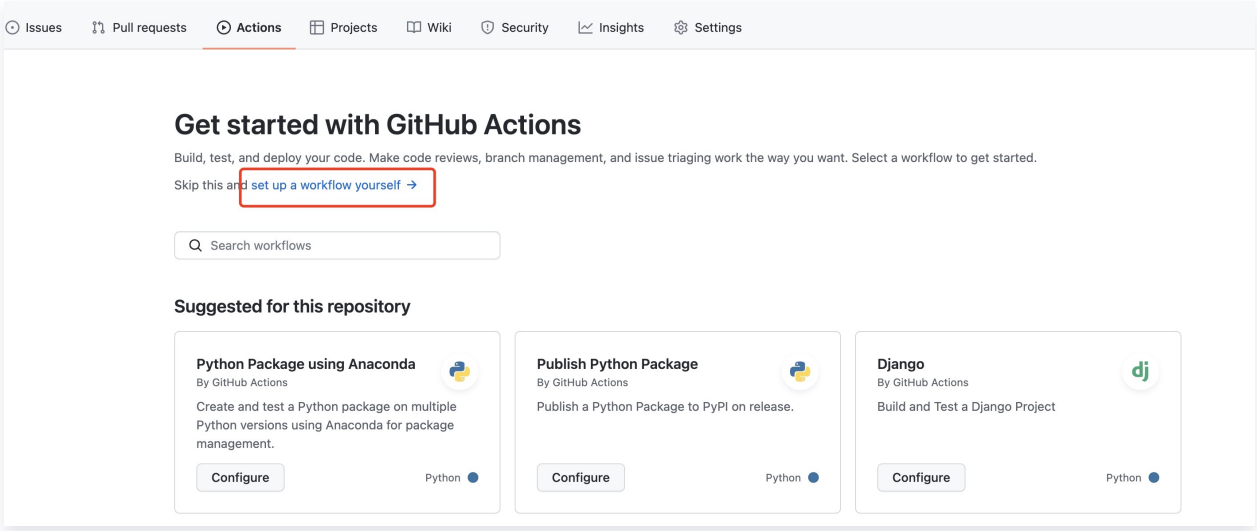
Instructions

> **Note**
> SCF has released Tencent Serverless Action in GitHub.

1. Search for "Tencent Serverless Action" in the GitHub Marketplace.



2. In Actions, select **Set up a workflow yourself** as shown below:



3. Directions:

- If you are familiar with the usage of Action, you can use the command provided below, which encapsulates the steps of installing Serverless Cloud Framework and executing the deployment command.

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

- If you are not familiar with the usage of Action, you can choose different yml syntax based on different languages. This article provides examples of yml syntax for Python, Java, and NodeJS for your reference.

Python

```
# Execute the current workflow when code is pushed to the main branch
# For more configuration information: https://docs.github.com/cn/actions/getting-started-with-github-actions
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 Variables
        STAGE: dev # Your deployment environment
        SERVERLESS_PLATFORM_VENDOR: tencent # The default for Serverless
overseas is AWS, configured for Tencent
        TENCENT_SECRET_ID: ${{ secrets.TENCENT_SECRET_ID }} # Your Tencent
Cloud account Secret ID, please configure in Settings-Secrets
        TENCENT_SECRET_KEY: ${{ secrets.TENCENT_SECRET_KEY }} # Your Tencent
Cloud account Secret Key, please configure in Settings-Secrets
```

Java

```
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 field of
the pom.xml
          settings-path: ${{ github.workspace }} # location for the settings.xml file
      - name: Build with Gradle # Use this for Gradle projects
        uses: gradle/gradle-build-
action@937999e9cc2425eddc7fd62d1053baf041147db7
        with:
          arguments: build
      - name: Build with Maven # Use this for Maven projects
        run: mvn -B package --file pom.xml
      - name: Create zip folder # This step is only for Java Web functions, used to store
jar and scf_bootstrap files. For Java event functions, it is sufficient to specify the Jar
```

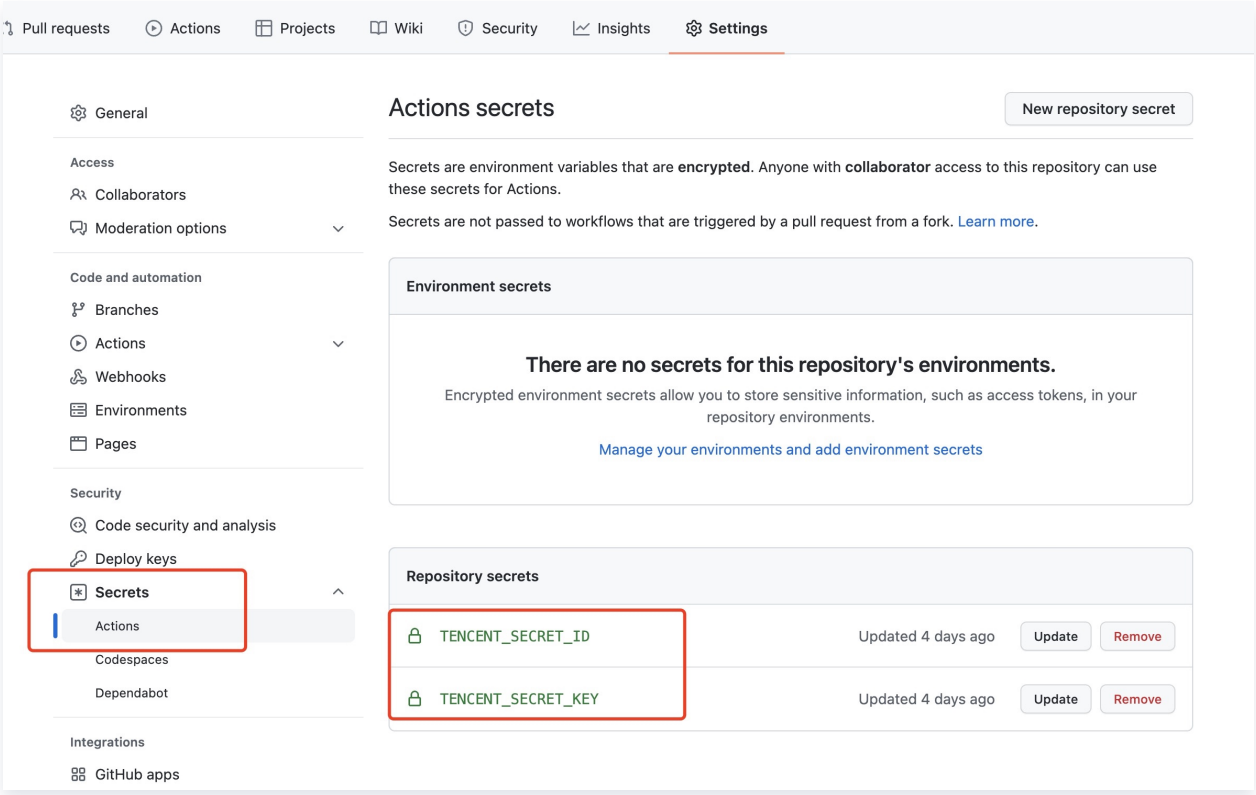
```
directory in Serverless.yml.
        run: mkdir zip
      - name: Move jar and scf_bootstrap to zip folder # This step is only for Java Web functions, used to move jar and scf_bootstrap files. For Java event functions, it is sufficient to specify the Jar directory in Serverless.yml. Note that if it is compiled with Maven, please modify the jar path below to /target.
        run: cp ./build/libs/XXX.jar ./scf_bootstrap ./zip
      - name: deploy serverless
        uses: woodyyan/tencent-serverless-action@main
        env: # Environment Variables
          STAGE: dev # Your deployment environment
          SERVERLESS_PLATFORM_VENDOR: tencent # The default for Serverless overseas is AWS, configured for Tencent
          TENCENT_SECRET_ID: ${{ secrets.TENCENT_SECRET_ID }} # Your Tencent Cloud account Secret ID
          TENCENT_SECRET_KEY: ${{ secrets.TENCENT_SECRET_KEY }} # Your Tencent Cloud account Secret Key
```

NodeJS

```
# Execute the current workflow when code is pushed to the main branch
# For more configuration information: https://docs.github.com/cn/actions/getting-started-with-github-actions
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 Variables
          STAGE: dev # Your deployment environment
```

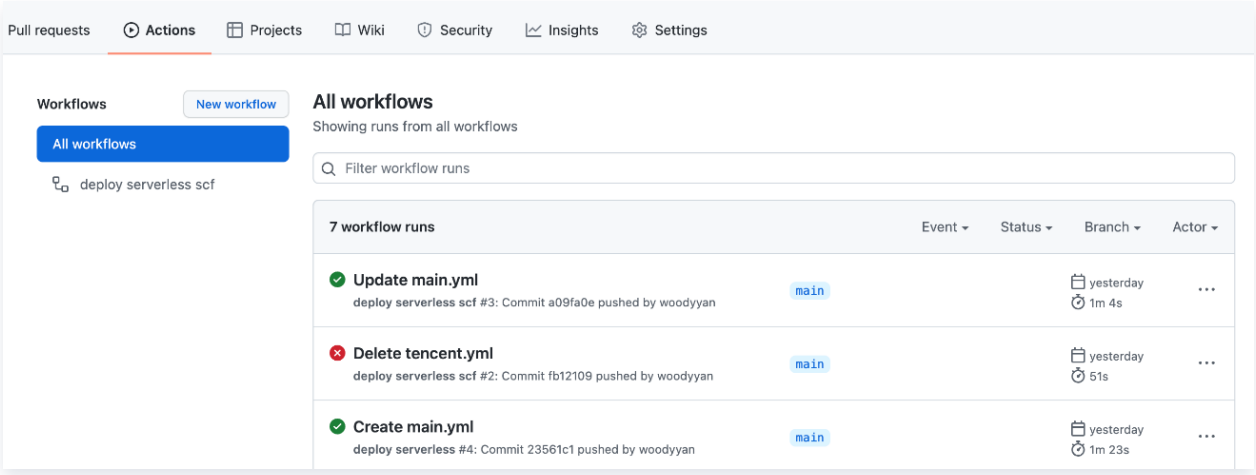
```
        SERVERLESS_PLATFORM_VENDOR: tencent # The default for Serverless
overseas is AWS, configured for Tencent
        TENCENT_SECRET_ID: ${{ secrets.TENCENT_SECRET_ID }} # Your Tencent
Cloud account Secret ID, please configure in Settings-Secrets
        TENCENT_SECRET_KEY: ${{ secrets.TENCENT_SECRET_KEY }} # Your Tencent
Cloud account Secret Key, please configure in Settings-Secrets
```

During deployment, you will need Tencent Cloud's `TENCENT_SECRET_ID` and `TENCENT_SECRET_KEY`. You need to configure these variables in the Secrets settings of your Github code repository, as shown below:



You can obtain the Tencent Cloud Secret ID and Secret KEY in Tencent Cloud's Access Management.

4. Once configured, each code push will automatically trigger the deployment process. You can view real-time execution results and error logs in Actions, as shown below:



In addition, you can also add testing, security checks, and release steps to the process as per the project requirements.

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.

Preparations

- You have hosted your SCF project onto platforms such as CODING, GitHub, GitLab, and Gitee.
- The project should include the `serverless.yml` configuration file required for executing the Serverless Cloud Framework.
- If you are using a Web function, please place the `scf_bootstrap` file in the root directory of your project.

Instructions

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

```
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
            userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId:
env.CREDENTIALS_ID]]])
      }
    }
```

```
        stage('Package'){ // This stage is only for Java projects
       steps {
        container("maven") {
         echo 'Package start'
         sh "mvn package" // This line is for Java Maven projects
                  sh "./gradlew build" // This line is for Java Gradle projects
                  sh "mkdir zip" // This line is only for Java Web functions, used to store jar
and scf_bootstrap files. For Java event functions, it is sufficient to specify the Jar
directory in the Serverless.yml.
         sh "cp ./build/libs/XXX.jar ./scf_bootstrap ./zip" // This line is only for Java Web
functions, used to move jar and scf_bootstrap files. For Java event functions, it is
sufficient to specify the Jar directory in the Serverless.yml. Note that if it is Maven
compilation, please modify the jar path below to /target.
       }
         }
    }
    stage('Install Dependencies') {
      steps {
        echo 'Installing dependencies...'
        sh 'npm i -g scf'
        sh 'npm install' // This line is for NodeJS projects
        echo 'Dependency installation completed.'
      }
    }
    stage('Deployment') {
      steps {
        echo 'Deploying...'
        withCredentials([
          cloudApi(
            credentialsId: "${env.TENCENT_CLOUD_API_CRED}",
            secretIdVariable: 'TENCENT_SECRET_ID',
            secretKeyVariable: 'TENCENT_SECRET_KEY'
          ),
        ]) {
            // Generate credentials file
            sh 'echo
"TENCENT_SECRET_ID=${TENCENT_SECRET_ID}\nTENCENT_SECRET_KEY=${TENCEN
T_SECRET_KEY}" > .env'
            // Deployment
            sh 'scf deploy --debug'
            // Remove credentials
            sh 'rm .env'
        }
        echo 'Deployment completed'
      }
```

```
    }
  }
}
```

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: 2023-09-27 19:58:50

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	The execution is successful.	-
400 InvalidParameterValue	The request event passed in by the event execution function is not of the JSON type.	Please refer to the Function Execution API documentation and the Synchronous Invocation API documentation for modifications and retry.
401 InvalidCredentials	The verification fails.	Your account does not have the permission to operate this function. Please refer to the Overview of Permission Management for instructions on granting permissions and retry.
402 ServiceSuspended	The service is temporarily suspended.	Your function service has been temporarily suspended, causing the service to stop. Please refer to the Manual Service Recovery instructions for modifications and retry.
403 Invalid scf response format	When using the integrated response, the returned data structure was not successfully parsed by the API Gateway.	When the API Gateway backend configuration has enabled integrated responses, please modify according to the Integrated Response Return Data Structure of the API Gateway Trigger and retry.
404 InvalidSubnetID	The subnet ID in the network configuration of the function is exceptional.	Please verify if the network configuration information of the function is correct and whether the subnet ID is valid.

405 ContainerStateExited	The container exits.	Please check your image or startup file to see if it can be started normally locally. If it can be started locally, please ensure it complies with the usage restrictions of SCF, such as read-only RootFS, and only /tmp is writable. For more details, please refer to the Troubleshooting Document. The local debugging command reference is as follows: `docker run -itd --read-only -v /tmp:/tmp`
406 RequestTooLarge	The input parameter event of the function call, that is, the size of the function's request event, exceeds the quota limit.	The request event size exceeds the quota limit, which is 6 MB for sync request events or 128 KB for async ones. Please adjust the request event size accordingly and try again.
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	The account balance is insufficient.	The SCF service is suspended because the Tencent Cloud account is in arrears. Please top up and try again.
429 ResourceLimit	The container request rate is too high and exceeds the limit due to concurrency surges.	The maximum expansion speed of elastic concurrency for each account (function burst) is set to 500 per minute by default in each region. During a concurrency surge, if there are not enough containers to handle the load, a large number of container requests will be triggered. This message will be returned once the account limit is exceeded. After evaluating the function concurrency, configure provisioned concurrency for the function to prepare containers in advance, preventing the container request rate from

		exceeding the limit due to concurrency surges. If, after evaluation, the provisioned concurrency cannot meet the business requirements, you can purchase a function package to enhance the function burst in the region.
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	This message will be returned when the concurrency exceeds the limit. The quota for the account region has been exceeded. The reservation has been exceeded.	For functions configured with a maximum reserved concurrency quota, if the function's concurrency exceeds the maximum reserved quota, the message "Function [xxx] concurrency exceeded reserved quota xxx MB" will be returned. You can adjust the function's maximum reserved quota based on business needs or refer to the Guide to Resolving Concurrency Overruns document. For functions that have not been configured with a maximum reserved concurrency quota, if the actual concurrency used by the function exceeds the remaining unoccupied concurrency quota in the region, the message "Function [xxx] concurrency exceeded region unreserved quota xxx MB" will be returned. You can evaluate your business needs and configure a maximum reserved concurrency quota for the function. If the remaining available quota in the region is insufficient to meet your business needs, you can purchase a function package to increase the total concurrency quota in the region.
433 TimeLimitReached	This message will be returned when the function does not complete execution within the configured execution timeout period.	Check whether a large number of time-consuming operations exist in the service code. Adjust the execution timeout period on the function configuration page. If the current setting is already at its maximum, you can refer to the Asynchronous Execution document to create an asynchronous

		execution function, which can provide a maximum function execution time of 24 hours. This status code will trigger Instance Recycling.
434 MemoryLimitReached	The memory limit is reached.	Check the code logic to see if memory leak exists. Increase the memory configuration on the function configuration page, or apply for a larger memory specification on the function memory configuration page, to obtain a maximum function execution memory of 120GB. This status code will trigger Instance Recycling.
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	Invalid parameter. The arguments passed to invoke do not conform to the standard.	The parameter does not conform to the standard. Please refer to API Document for modification 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. Please confirm the entry file in the code package and the execution method.
438 FunctionStatusError	The function is abnormal or the SCF service is suspended.	The function is invoked in an abnormal state. Wait for the function status to become normal and try again. The SCF service is suspended because the Tencent Cloud account is in arrears. Please top up and try again.

439 User process exit when running	The user process exists accidentally.	Based on the error message, please find out the cause, fix the function code, and try again. This status code will trigger Instance Recycling.
441 UnauthorizedOperation	CAM authentication fails.	Please ensure that the CAM authentication information for the function invocation role is correctly passed. For more information on granting permissions, see Overview of Permission Management.
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.	Please verify the integrity and validity of the selected image and try again. If the image can be downloaded normally locally and the issue persists, please contact Online Customer Service or submit a ticket.
445 ContainerInitError	Container start fails.	Container start fails. Please check whether your bootstrap file has been uploaded successfully and ensure that the invocation path is correct. For image deployment functions, please ensure that the Command or Args parameters passed in from the console are in the correct format. For more details, refer to How to Use Image Deployment Functions. 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.	Container initialization exceeds the Initialization Timeout. Please verify if your listening port is set to 9000. Please check whether all the files in the code package or container image are required

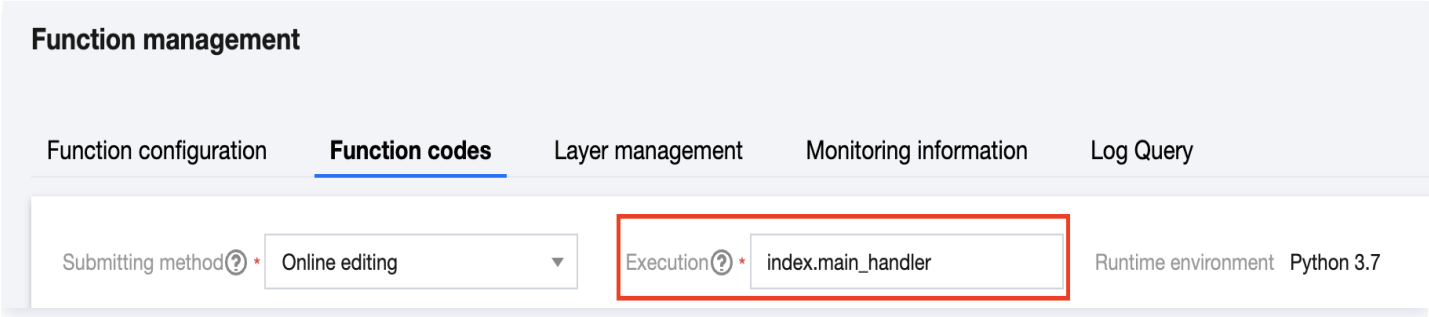
		files. Appropriate streamlining can improve the initialization speed of the container. Please check whether there are any exceptions or time-consuming business logic in the initialization code. You can appropriately increase the initialization timeout period and try again.
447 PullImageTimeOut	Image pull times out.	The timeout may be caused by a large image size or network jitter. It is recommended to retry after minimizing the image or increasing the Initialization Timeout. If the problem persists, please contact Online Customer Service or submit a ticket.
449 InsufficientResources	There are no resources available at the resource specification selected by this function in the specified region.	If the resource type is a large CPU specification or GPU, it can be used in combination with presets. If the problem persists, please submit a ticket.
450 InitContainerTimeout	Container start times out.	The container start time exceeds the Initialization Timeout. Please minimize the code or increase the initialization timeout and try again.
452 NetworkSuspended	The function has not been called for a long time, and the network is in a frozen state.	Redeploy the function code or update the function configuration.
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	An internal error occurs.	An internal error has occurred. Please try again later. If the issue persists, please contact our online customer service or submit a ticket.

Concepts

Execution Method

The execution method indicates which function from which file should be initiated when invoking the cloud function. As illustrated below:



- 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`.
  - This execution method **first part points to the filename without suffix in the code package, and the second part points to the entry function name in the file.** Ensure that the file name suffix in the code package matches the language environment, such as `.py` files for Python environment, and `.js` files for Node.js environment. For more details on execution methods, please refer to Detailed Explanation of Execution Methods.
- For Java programming, use the **package.class::method** format, such as `example.Hello::mainHandler`.
- For the Custom Runtime environment, a non-fixed format can be used based on the custom programming language implementation.

Common Errors And Solutions

Last updated: 2023-09-27 21:00:07

Common errors and solutions:

Error Code	Solution
InvalidParameter.FunctionName	The FunctionName value does not conform to the standard. Please refer to the API documentation for corrections and try again.
InvalidParameterValue.Action	The requested API does not exist. Please refer to the API documentation for corrections and try again.
InvalidParameterValue.CosBucketRegion	The CosBucketRegion value does not conform to the standard. Please refer to the COS Region and Access Domain for corrections 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 Memory value is invalid. The default memory size for function execution is 128MB, with options ranging from 64MB to 3072MB in increments of 128MB. Please correct and try again.
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 the API Documentation.