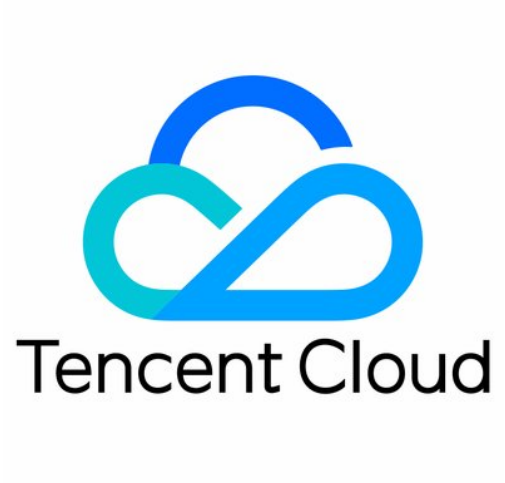
Serverless Cloud Function

Product Introduction

Product Documentation



Contents

Product Introduction

Overview

Last updated：2024-04-19 15:12:30

Tencent Cloud Serverless Cloud Function (SCF) is a serverless execution environment that enables you to build and run applications without having to purchase and manage servers. It is an ideal computing platform for use cases such as real-time file processing and data processing. Simply code in a supported language and set the execution conditions, and your code can be run on the Tencent Cloud infrastructure elastically and securely.

Evolution of Computing Resources

With the development of cloud services and the high abstraction of computing resources, Tencent Cloud provides a wide variety of computing resource options from physical servers to cloud-based functions at different abstraction levels.

Cloud Physical Machine (CPM): scaling is at the physical machine level. You enjoy the physical computing resources exclusively, which provides the best security.

Cloud Virtual Machine (CVM): it virtualizes hardware devices, and scaling is at the virtual machine level. You share the physical machine resources with other tenants, and you can configure CVM metrics on your own, which is simpler than deployment and iteration.

Tencent Kubernetes Engine (TKE): it virtualizes operating systems, and scaling is at the service level. The test and production environments are exactly the same, making testing and deployment very convenient.

Serverless Cloud Function (SCF): it virtualizes runtime environments, and scaling is at the function level. This is the smallest unit of existing computing resources, which features full automation, one-click deployment, and high scalability, and is an ideal choice for lightweight service deployment.

Serverless

Serverless does not mean that there is no server. It is just that you do not need to care about the underlying resources, log in to a server, or optimize the server. You only need to care about the core code snippets while skipping the complex and cumbersome basic work. These code snippets are completely triggered by events or requests, and the platform automatically adjusts service resources in parallel based on the requests. The serverless architecture has near-infinite scalability, and no resources will be executed when idle. The code is executed in a stateless manner, which easily enables fast iterations and rapid deployment.

SCF Overview

SCF is a serverless execution environment provided by Tencent Cloud. All you need to do is write simple and single-purpose functions and associate them with events generated by your Tencent Cloud infrastructure and services. When using SCF, all you need to do is write the code in a programming language (Python, Node.js, PHP, Go, Java, and Custom Runtime) supported by the platform. The underlying computing resources and tasks are fully managed by Tencent Cloud, including maintenance of server CPUs, memories, networks, and other configurations/resources, code deployment, elastic scaling, load balancing, security upgrade, and resource execution monitoring. However, this means that you cannot log in to or manage servers or customize the system or environment.

SCF is automatically deployed in multiple AZs in the same region with extremely high fault tolerance achieved. When a function is executed, scaling will be made based on the request load with no manual configuration or intervention required, helping meet the needs for service availability and stability in different scenarios. From several requests per day to thousands of requests per second, SCF can automatically scale at the underlying layer. You only need to pay for running functions, and if no function is running, no fees will be incurred.

You can customize the timing of running a function, such as when a file is uploaded to a COS bucket, when a file is deleted, when a message in CKafka is used, or when an application is called through an SDK. You can also configure the function to run regularly. Therefore, SCF can be used as a data processing trigger for the COS service to easily implement IFTTT logic. Scheduled automated tasks can also be flexibly built to free you from manual operations and easily construct an elastic and controllable software architecture.

SCF Features

Serverless helps you get rid of the tedious development and configuration work, so that you can only care about the writing of business code logic, without any infrastructure construction, management, and OPS overheads. This service model lowers the threshold of R&D and improves the efficiency of business construction; therefore, it has gained the recognition of high numbers of enterprises and developers.

Various development tools and languages supported for smooth development

The Tencent Cloud Serverless team works in many ways to provide convenient tools or capabilities that can meet the needs in a wide variety of development scenarios; for example:

You can use Serverless Cloud Framework to create, debug, and package projects in your local development environment and then quickly deploy them online.

With the aid of visual operations based on the VS Code plugin and IDE, online/offline management of functions and code writing and debugging can be performed at one single place. The VS Code IDE and plugin also enables local management, development, and debugging as well as online release of functions.

Web IDE is supported; therefore, you can develop and debug projects in real time in the console, which delivers the same experience as local development and debugging and makes it easier for you to view and adjust code.

SCF supports Python, Node.js, Go, PHP, Java, and Custom Runtime, so you can select a custom runtime environment as needed.

Multiple deployment methods for various environments

SCF supports deployment through the console, command line, SDK/API, web IDE, and image.

Diversified triggers to support more business scenarios

Triggering methods of SCF include API, SDK, and events in other Tencent Cloud services such as COS and API Gateway, which enrich the use cases.

Automated and flexible execution for better invocations

SCF can automatically scale according to the call volume, which is imperceptible to users, perfectly fits the invocation curve, and saves resources and costs to the greatest extent.

Pay-as-You-Go billing accurate to the millisecond level

SCF supports billing actually used resources at a time granularity of 1 ms, which can significantly reduce your costs compared to the time granularity of 100 ms.

Related Concepts

Last updated：2024-04-19 15:12:30

Tencent Cloud Serverless Cloud Function (SCF) is a function-as-a-service (FaaS) product that provides a serverless FaaS computing platform. Therefore, when combined with the trigger event source, an SCF functions can be triggered by an event generated by the trigger source.

Serverless

The origin of the serverless architecture concept can be explained in the article Serverless Architecture by Mike Roberts at Martin Fowler's blog website.
Serverless does not mean that computation can be performed without a server; instead, it means that developers can use relevant resources without caring about the underlying servers.
More broadly, cloud services that can be used directly without configuring or understanding the underlying servers can also be seen as serverless to some extent.
In the SCF product, we are targeting the computation in serverless scenarios. SCF provides FaaS capabilities in serverless mode.

FaaS

Function-as-a-service (FaaS) provides the capability to execute stateless, transient, event-triggered code directly in the cloud.
FaaS is different from traditional application architectures. It provides an event-triggered operation method, where a function is not always running but triggered by an event when the event occurs, and the event is only handled once during one execution. Therefore, in the code of the function, only the handling flow for one event needs to be considered, and high-concurrency handling of many events is supported by the multi-instance function concurrency implemented by the platform.
In order to support high concurrency, the SCF platform features automatic elastic scalability, which will launch more instances to handle event requests when high volumes of requests are received and will reduce function instances (down to zero) when less or no requests are received. Therefore, in order to match the automatic scaling capability, the function code needs to be developed in a stateless manner, that is, the relevant state data is not retained in the running memory of the function or depended on when the function is executed multiple times. The state data of the function can be stored in external persistent storage services such as Cloud Memcached, TencentDB, and COS.

Triggers and trigger sources

Anything that can generate an event and trigger the execution of a function can be referred to as a trigger or trigger source. The trigger triggers function execution by passing the event to the function after it generates the event itself. The trigger can trigger the function synchronously or asynchronously according to its own characteristics. When the function is triggered synchronously, the trigger will wait for the function to complete and return the execution result; when the function is triggered asynchronously, the trigger will only trigger the function and ignore the execution result. When connecting with other Tencent Cloud products or services, SCF has some special ways of implementing such as push mode and pull mode.
Push mode: a trigger actively pushes the event to the SCF platform and triggers function execution.
Pull mode: the SCF platform pulls an event through the pull module from a trigger and triggers function execution.

Trigger events

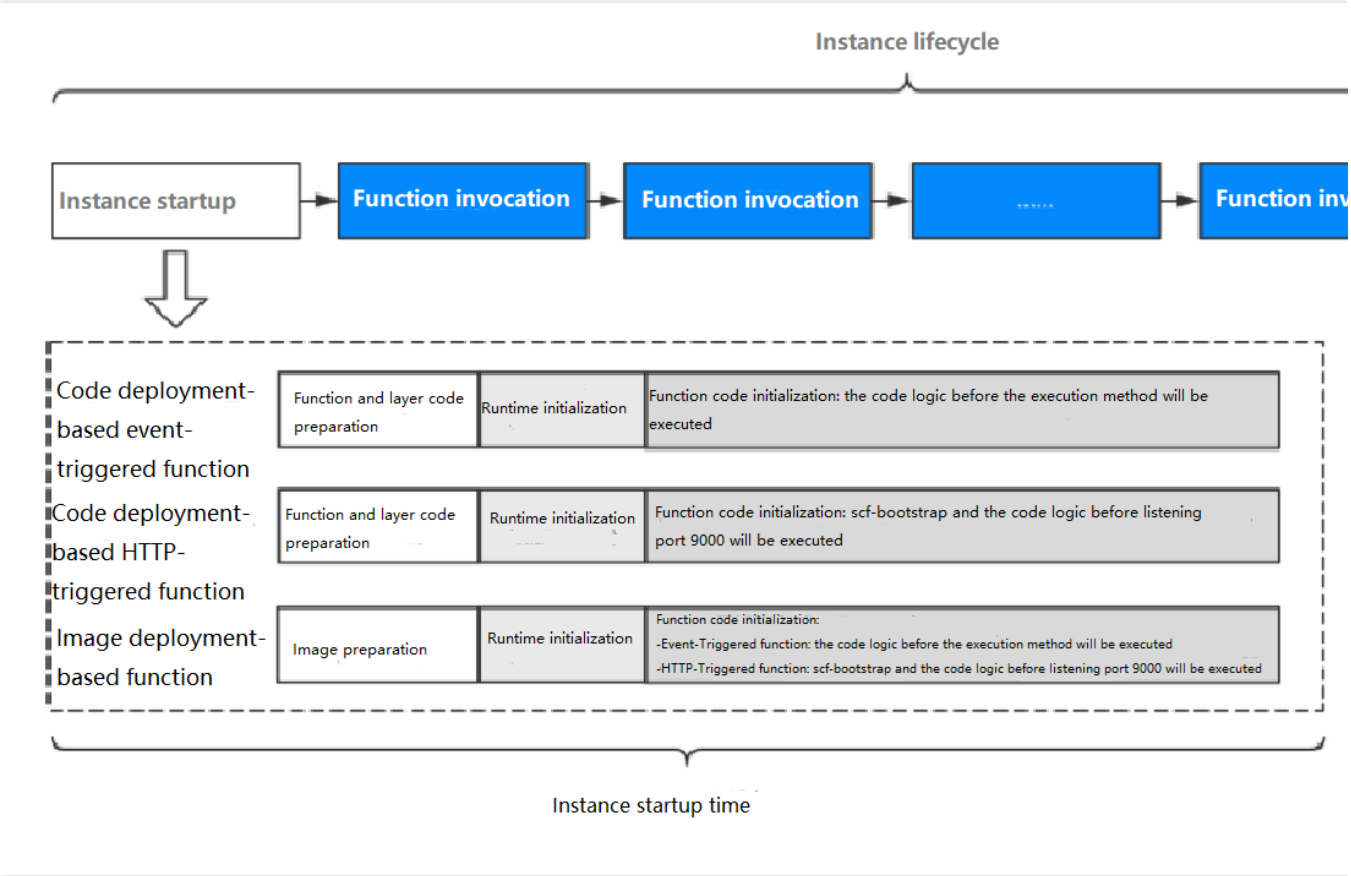
A trigger passes an event to a function when the function is triggered. The event is represented by a specific data structure in JSON format and passed to the function as the event input parameters of the function.
The JSON data content of the trigger event will be converted to corresponding data structures or objects in different language environments, and you do not need to perform the conversion from JSON structures to data structures in the code on your own.For example, in Python environment, the JSON data content will be converted to a complex dict object, that is, the event input parameters of the function is a complex Python dict object. In Go or Java, the input parameters have to be an object that can match the event data structure. For more information on specific implementations, please see Development Language Descriptions.

How It Works

Last updated：2024-04-19 15:12:30

Instance Model of Function Runtime

SCF will execute a function for you when the function receives a triggering request. Instance is the resource for SCF to execute the request. SCF will allocate resources based on the function configuration information (such as memory size) and launch one or multiple instances to process the function request. The SCF platform is responsible for the creation, management, and deletion of all function runtime instances, and you have no permissions to manage them. The lifecycle of an instance is as shown below:



Starting instance

If there is no running instance when a request arrives, the request will trigger the startup of an instance. Instance startup usually takes some time, which adds extra time to the invocation that triggers the instance startup. Generally, instance startup is triggered only when a function is invoked for the first time, updated, or invoked again after a long period of inactivity.

The instance startup time will be reflected in the `Coldstart` field in the `Init Report` or `Provisioned Report` line of the function execution log.

You can use the provisioned concurrency feature to start instances in advance to avoid triggering the instance startup when the function request arrives.

The instance startup time is limited by the function's initialization timeout period. If the former is longer than the latter, instance startup will fail. You can accelerate instance startup or increase the initialization timeout period accordingly as detailed below.

Method to accelerate the three phases of instance startup:

Code preparation: The platform pulls the function code, layer, or image uploaded by you to prepare for function execution. The code preparation time is positively proportional to the sizes of code package, layer, and image. We recommend you reduce the code package size as much as possible and only keep the code files and dependencies necessary for function execution in order to minimize the code preparation time. This time will be reflected in the `PullCode` field in the `Init Report` or `Provisioned Report` line of the function execution log.

Runtime initialization: The platform prepares the runtime environment that function execution depends on according to your function configuration. This time will be reflected in the `InitRuntime` field in the `Init Report` or `Provisioned Report` line of the function execution log.

Function code initialization: The code initialization time is positively proportional to the complexity of the code logic. We recommend you optimize the code logic as much as possible to minimize the code initialization time. This time will be reflected in the `InitFunction` field in the `Init Report` or `Provisioned Report` line of the function execution log.

Code deployment-based event-triggered function: The code logic before the execution method configured for the function will be executed during instance startup. For example, if the entry function is `main_handler` , all the code logic before `main_handler` will be executed.

Code deployment-based HTTP-triggered function: The bootstrap file `scf_bootstrap` and the code logic before listening port 9000 will be executed during instance startup.

Image deployment-based function: For event-triggered functions, the code logic before the execution method configured for the function will be executed; for HTTP-triggered functions, the code logic before the bootstrap file `scf_bootstrap` and listening port 9000 will be executed.

Reusing instance

In order to minimize the additional time caused by instance startup, the platform will try to reuse the instance for subsequent invocations. After the instance processes the function request, it will be stored for a period of time according to the actual situation of the platform for next invocations and will be used first during this period.

The meaning of instance reuse is as follows:

All declarations outside the execution method part in your code remain initialized and can be reused directly when the function is invoked again. For example, if a database connection is established in your function code, the original

connection can be used directly when the container is reused. You can add logic to your code to check whether a connection already exists before creating a new one.
Each container provides some disk space in the `/tmp` directory. The contents of this directory are retained when the container is retained, providing a temporary cache that can be used for multiple invocations. It is possible to use the contents of the disk directly when the function is invoked again. You can add extra code to check whether such data is in the cache.

Note:

Do not assume that the instance is always reused in the function code, because reuse is related to the single actual invocation, and it cannot be guaranteed whether a new instance will be created or an existing one will be reused.

Repossessing instance

The platform will repossess instances that have not processed requests for a certain period of time.

Temporary Disk Space

Each function has a temporary disk space of 512 MB (`/tmp`) during execution. You can perform certain read and write operations on the space in the execution code or create subdirectories, but this part of data may **not** be retained after function execution is completed. Therefore, if you need to persistently store the data generated during execution, use COS or external persistent storage services such as Redis/Memcached.

Call Types

The SCF platform supports both sync and async calls of functions.

Sync invocation

Sync invocation will wait for the execution result of the function after the invocation request is sent.

Async invocation

Async call will only send the request and get the request ID of the current request, but not wait for the result.
When an async invocation occurs, the async event will be placed in the async queue built in SCF and then consumed by the event execution function in the async queue. Async queues have the following restrictions:
Async queues are at the trigger level, and one function trigger has one queue.
An async event can be retained in a queue for up to 6 hours.
There can be up to 100,000 messages in an async queue.
The retry policy may vary by async queue. For more information, see Retry Policy.

Defining function invocation type

The call type is independent of the configuration of the function itself and can only be controlled when the function is called.

In the following call scenarios, you can freely define the call type of the function:

The SCF function is invoked by a written application. If you need to make a sync invocation, use the InvokeFunction API; if you need to make an async invocation, use the Invoke API and pass in the `invokeType=Event` parameter.

The SCF function is manually invoked (with API or CLI) for testing. The parameters for the invocation are the same as above.

If you use another Tencent Cloud service as the event source, the invocation type of the cloud service is predefined.

Sync invocation: By API Gateway trigger, CLB trigger, and CKafka trigger, for example.

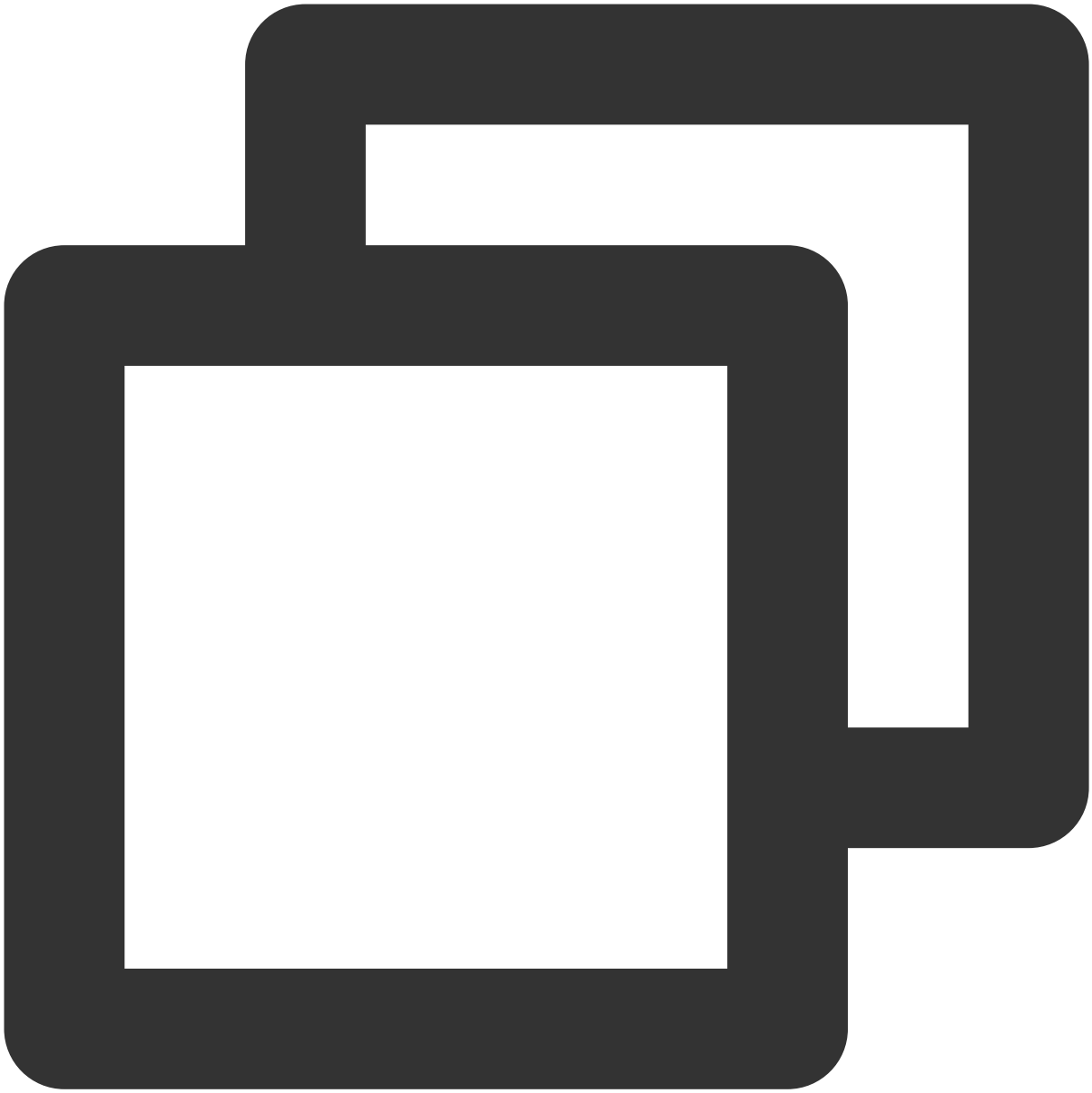
Async invocation: By COS trigger, timer trigger, and CMQ topic trigger, for example. For more information, see Trigger Overview.

Usage Restrictions

For the restrictions on function usage quotas and related environments, see Quota Limits.

Function concurrency

The function concurrency is the number of executions of the function code in any given period of time. For the current SCF function, the request is executed once each time an event is published. Therefore, the number of events (i.e., requests) published by the trigger affects the function concurrency. You can use the formula below to estimate the total number of concurrent function instances.



```
Requests per second * function execution duration (in seconds)
```

For example, for a function that handles COS events, if the function takes an average of 0.2 seconds (i.e., 200 milliseconds) for execution and COS publishes 300 requests per second to the function, then 300 * 0.2 = 60 function instances will be generated simultaneously.

Concurrency limits

Currently, SCF has a limit on the amount of concurrency for each function. You can view the limit for the current function in Concurrency Overview.

If an invocation causes the function concurrency to exceed the default limit, the invocation will be blocked and not executed by SCF. Restricted invocations are handled differently depending on the function invocation type:
Sync invocation: If the function is restricted when invoked synchronously, a 432 error will be returned directly.
Async invocation: If the function is restricted when invoked asynchronously, SCF will retry the restricted event according to a certain policy.

Execution Environment and Available Libraries

The current SCF execution environment is built based on the following:
Standard CentOS 7.2
If you need to include executable binaries, dynamic libraries, or static libraries in your code, make sure that they are compatible with this execution environment.
Based on different language environments, there are base libraries and additional libraries installed for the corresponding language in the SCF execution environment. You can view the additional libraries installed in the environment in the descriptions of each language:
Python
Node.js
Golang
PHP
Java

Deployment Mode

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, see the corresponding documents.

SCF event-triggered function

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

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
memory_limit_in_mb	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:

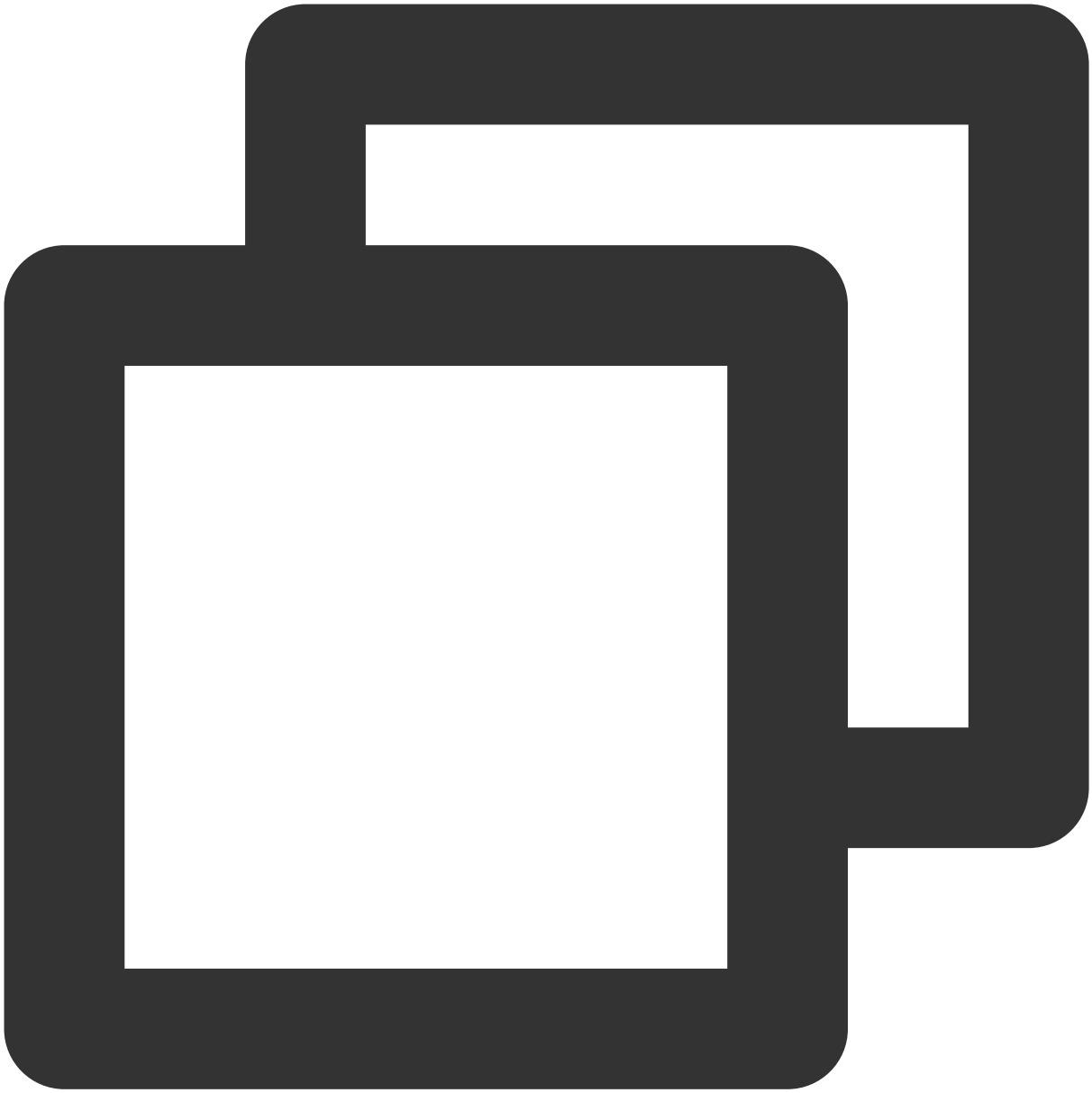
To ensure compatibility, the descriptions of the namespace at different stages of the SCF function are retained in the context.

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-
```

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, see Trigger Overview.

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

Function response

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	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:

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).

Solutions

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

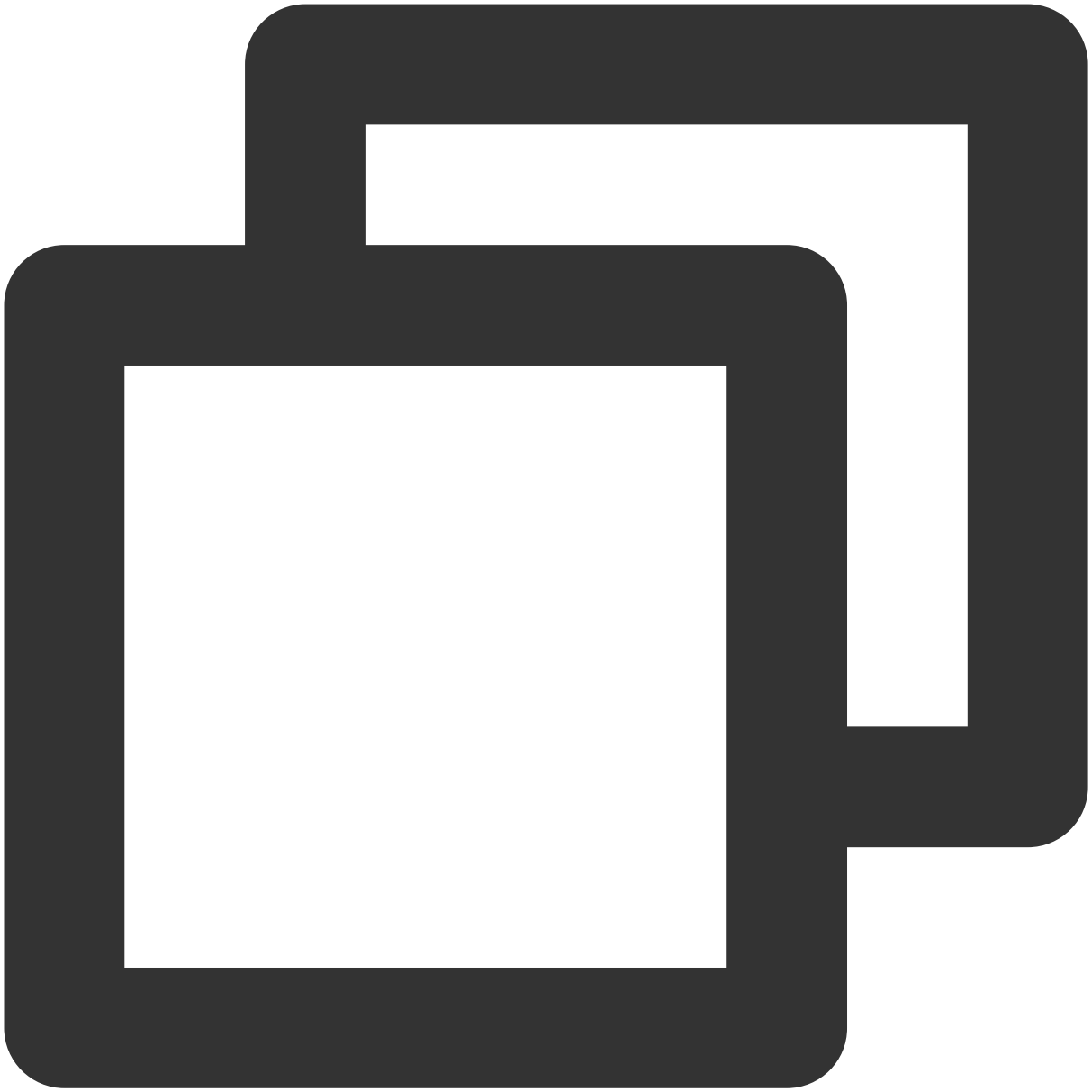
Use the `Try` statement to capture errors

Sample

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

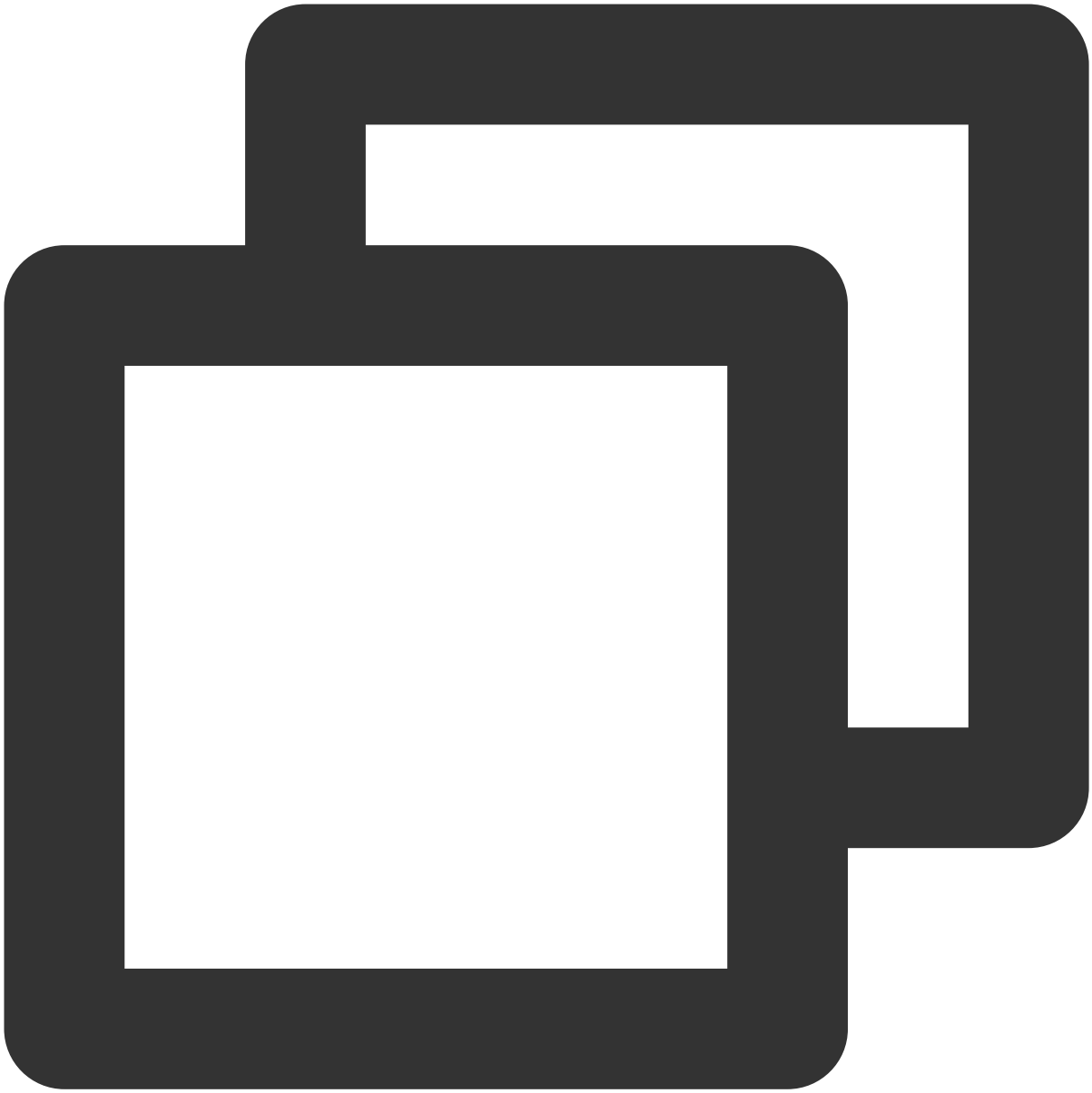
Description

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

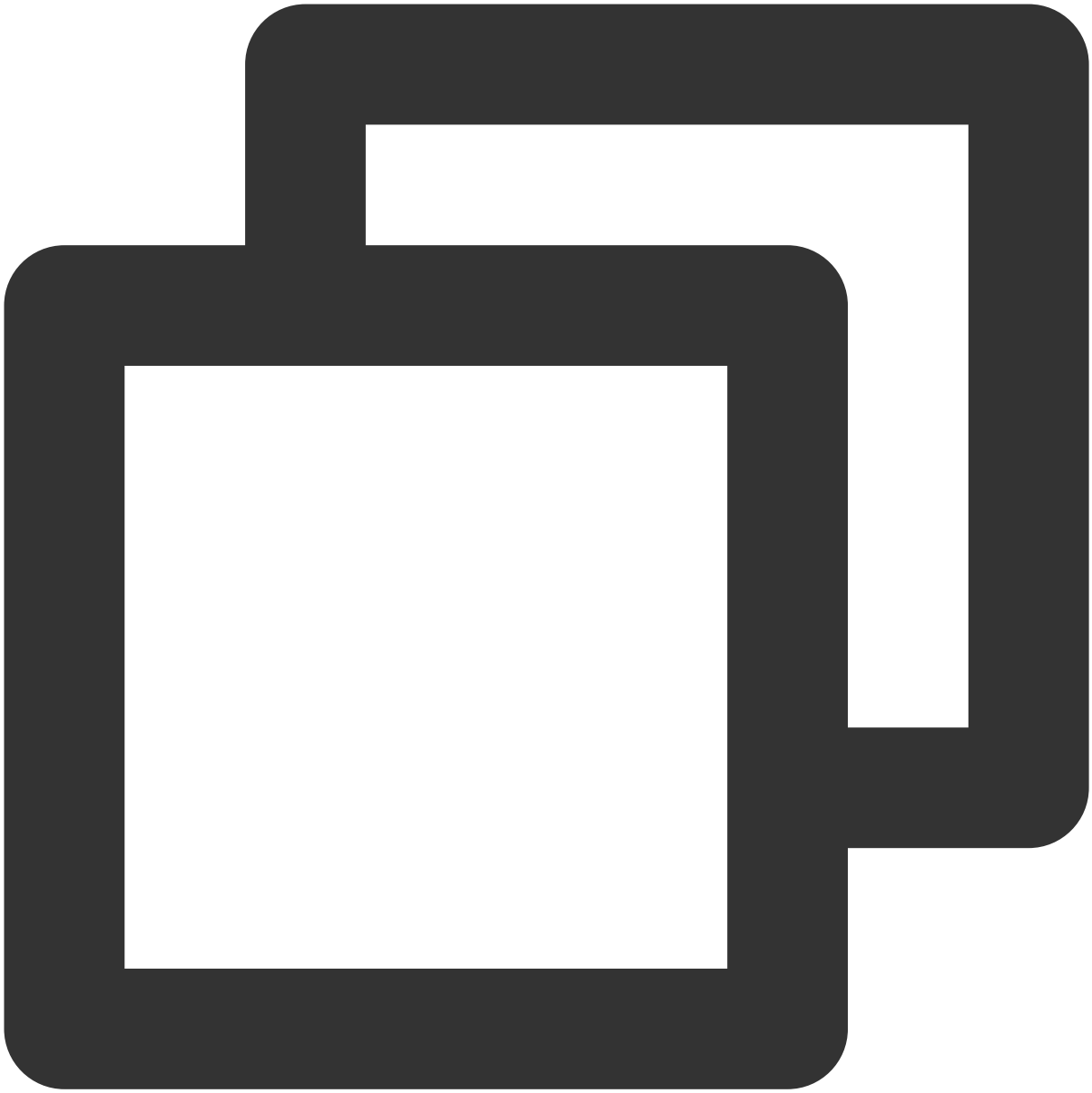


```
class UserNameAlreadyExistsException (Exception): pass
def create_user (event):
    raise UserNameAlreadyExistsException ('
        The username already exists,
        please change a name!')
```

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
`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"}

Logs

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, 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, we recommend you store persistent states in TDSQL, COS, TencentDB for Memcached, or other cloud storage services.

Development Process

For more information on the function development process, see User Guide.

Strengths

Last updated：2024-04-19 15:12:30

Ease of Use

Reduced component overheads

When using SCF, you don't have to take care of peripheral components such as load balancing, automatic scaling, and gateways; instead, you only need to write core codes, which greatly reduces service architecture complexity.

Automatic scaling

SCF can scale based on the amount of requests with no manual configuration required. Regardless of whether your application receives only couples of requests daily (e.g., scheduled transactions such as log collection) or handles thousands of requests per second (e.g., backend of a mobile app), SCF can automatically arrange reasonable computing resources to meet the business needs.

Efficient Development

Accelerated development

SCF does not require specific frameworks or dependencies, so developers can focus on the development of core code. In addition, they can form multiple small teams, and the development of individual modules does not require knowledge of the details of code written by other teams, which significantly accelerates independent development and iteration and helps optimize the timing of product launches.

Reuse of third-party services

You can use SCF to write some single-purpose, logically independent business modules, so that you can fully reuse mature third-party code implementations, such as using OAuth to implement a login module.

Simplified OPS

Each function is executed, deployed, and scaled separately and can be automatically deployed after you upload the corresponding code, which eliminates the trouble with deploying and upgrading standalone applications.

Stability and Reliability

High-availability deployment

SCF can automatically select a random availability zone in each region for function execution. If an availability zone is down due to a natural disaster or power failure, SCF can automatically select the infrastructure of other availability zones for code execution, eliminating the risk of service interruption inherent in single-availability zone operations.

Supplement to other computing services

Resident workloads can be sustained by CVM or TKE, while event-triggered workloads can be sustained by SCF. Different Tencent Cloud services can be leveraged to meet the needs of different business scenarios and make your service architecture even more robust.

Simplified Management

Simple security configuration

With SCF, complex configuration and management of OS intrusions, login risks, file system security, network security and port monitoring become a thing of the past. Everything is handled by the platform which ensures user isolation through customized containers.

Visual Management

You can directly manage the function code and execution time (i.e., function trigger) in the console, and deploy and test functions in one click with no complicated configuration files needed.

Significantly Reduced Overheads

Usage-based billing

SCF does not incur any fees when it is not in use, so the costs will be significantly reduced for non-resident business processes. When the SCF executes code, you will be billed for the number of requests and the execution duration of the computing resources. This billing mode has obvious advantages and is very friendly to developers in the start-up stage.

Use Cases

Last updated：2024-04-19 15:12:30

Serverless Cloud Function (SCF) is constantly being developed and upgraded. With increasing capabilities and compatibility with other products, SCF can be used in more use cases.

File Processing and Notifications

By using COS as a function trigger, event notifications can be sent when a file in a COS bucket changes, which enables the prompt processing of the changed file and business notifications.
For example, once an image is uploaded to a COS bucket, the cloud function is notified immediately, and the image can be immediately obtained for automatic processing, such as cropping, thumbnailing, and watermarking. In addition, the processed image can be written to a database for future use.

Data ETL Processing

Some data processing systems need to process huge amounts of data frequently, either periodically or planned.
For example, a securities company needs to collect statistics on its transactions every 12 hours and determine the top 5 transaction volumes. Another example is that a flash sale website needs to process its transaction flow logs once a day to obtain errors caused by the resources being sold out and thereby analyze the website buzz and trends.
Possessing great scalability, SCF facilitates the computation of large-volume data. SCF also enables the concurrent execution of multiple mapper and reducer functions on the source data, and finishes the tasks in a short period of time. Compared with traditional functions, SCF reduces costs by preventing resources from being idleness and wasting resources.

Mobile and Web Applications

Cloud functions let you run mobile and web application backend code to implement the server-side application logic and provide services through APIs. By using SCF with Cloud Cache, TencentDB, COS, and other products, developers can build mobile and web applications with elastic scalability and create various serverless backends. These applications can run in multiple IDCs with high availability. You don’t need to manage the scalability and backup redundancy.

AI Inference

After an AI model is trained and ready to provide inference services, you can use SCF to encapsulate the model into a function. The code is executed upon request reception. In this way, you only need to pay on demand without investing in servers and GPUs to get the automatic scaling capability featuring high request concurrency.

Message Relay

CMQ and CKafka can be used as function triggers. A received message triggers the execution of the cloud function, and the message is passed to the cloud function as the event content.
For example, when CKafka receives a log of a business system, the cloud function can write the log content as a file to COS for log archive and storage.

Business Flow

As the intermediate channel of business event flow, CMQ connects multiple functions for business status flow and assignment. According to the messages, business logic judgment and processing in the functions can be performed to implement channel assignment, status flow, and event distribution, which connect the complex business processes.

Related Products

Last updated：2024-04-19 15:37:08

Serverless Cloud Function (SCF) may be associated with or use the following products:

Product name	Relationship to SCF
Virtual Private Cloud (VPC)	You can access resources in VPC by configuring SCF to VPC.
Cloud Object Storage (COS)	You can configure a COS trigger to trigger a function when an event is generated by the corresponding bucket.
Cloud Log Service (CLS)	You can configure a connection with CLS to write the execution logs of functions to CLS.
Cloud Message Queue (CMQ)	You can configure a CMQ trigger to trigger a function when a message is received by the corresponding queue.
Cloud Kafka (CKafka)	You can configure a Ckafka trigger to trigger a cloud function when a message is received by the corresponding Kafka topic.
API Gateway	You can configure an API Gateway trigger to trigger a function when an HTTP request is received at the API URL.
Cloud Access Management (CAM)	You can configure a CAM role to grant SCF the access to authorized resources when function code is executed.