

# **Serverless Cloud Function**

## **User Guide**

### **Product Documentation**



## Copyright Notice

©2013-2025 Tencent Cloud. All rights reserved.

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

## Trademark Notice



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

## Service Statement

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

# Contents

## User Guide

### Quota Management

- Quota Limits

- Exceeded Quota Management

### Managing Functions

- Function Overview

- Create a Function

- Testing a Function

- Debugging Function

- Querying a Function

- Query a Function Execution Log

- Deleting a Function

- Deploying Functions

### Web Function Management

- Function Overview

- Creating and Testing Function

- Bootstrap File Description

- Trigger Management

- HTTP-Triggered Function Billing

- Deploying Web Function on Command Line

- WebSocket Protocol Support

- HTTP-Triggered Function Request Concurrency Management

### Log Management

- Log Search Guide

- Log Delivery Configuration

- Log Delivery Configuration (Legacy)

### Concurrency Management

- Concurrency Overview

- Concurrency Management System

- Provisioned Concurrency

- Scheduled Provisioned Concurrency

- Dynamic Provisioned Concurrency Metric

- Concurrency Overrun

### Trigger Management

- Creating a Trigger

Deleting Triggers

Enabling/Disabling Triggers

Function URL

Function URL Overview

Creating a Function URL

Function URL Authentication and Authorization Configuration

A Custom Domain Name

Configuring a Custom Domain Name

Version Management

Overview

Viewing a Version

Releasing a Version

Using a Version

Alias Management

Traffic Routing Configuration

Using Alias to Implement SCF Grayscale Release

Permission Management

Permission Management Overview

Role and Authorization

Role and Authorization

SCF Policy Syntax

Sub-users and Authorization

Users and Permissions

Creating a Sub-user and Granting It All Permissions of SCF

Creating a Sub-user and Granting It Permissions to Operate Certain Functions

Plugin Management

Configure Function Plug-Ins

Create Plugin

Bind a plugin

Managing Monitors and Alarms

Descriptions of monitoring metrics

Configuring Alarms

Viewing Execution Logs

Network Configuration

Network Configuration Management

Fixed Public Outbound IP

VPC Communication

Granting a Function in VPC Access to Public Network



## Layer Management

- Overview

- Creating Layer

- Binding function to layer

- Using layer

## Execution Configuration

- Async Execution

- Status Trace

- Async Event Management

## Extended Storage Management

- Mounting CFS File System

## DNS Caching Configuration

## Running Instance Management

- Instance Level Monitoring

- Running Instance Command Line Operation

## Resource Managed Mode Management

## Near-Offline Resource Hosting Model

# User Guide

## Quota Management

### Quota Limits

Last updated : 2025-03-28 10:24:33

SCF has certain quota limits for each user account.

#### User Account Quota Limits

Content	Default Quota Limit
Total function code size per region	100 GB
Total function concurrency quota per region	128,000 MB(Guangzhou, Shanghai, Beijing, Chengdu, and Hong Kong (China))
	64,000 MB(Singapore, Tokyo, Silicon Valley, Frankfurt, Shenzhen Finance, and Shanghai Finance)
Number of namespaces per region	5
Total concurrent function quota per namespace	You can purchase a <a href="#">function package</a> to adjust the quota.
Number of functions per namespace	50

#### Function Quota Limits

Content	Default Quota Limit
Function name length limit	60 characters. The total length of the namespace name + function name cannot exceed 118 characters.
Maximum code size (including bound layers) per function (version) before compression	500 MB
Maximum number of same-type triggers per function	10
Maximum environment variable size per function	4 KB

Number of layer versions bound to one function version	5
--	---

## Layer Quota Limits

Content	Default Quota Limit
Number of layers per region	20
Number of versions per layer	200

### Note:

SCF currently supports one million MB-level concurrency, which can effectively support scenarios with high concurrency demand such as ecommerce promotions and parallel processing of medical and biological data. The concurrency quota per region on the SCF platform is shared by all functions by default. You can customize the [function concurrency](#) to meet your actual needs. If you want to increase the quotas or add concurrency quota management capabilities at the namespace granularity, you can directly purchase a [function package](#).

In SCF, a [COS trigger](#) has limits in two dimensions: SCF and COS, as detailed below:

SCF dimension: One function can be bound to 10 COS triggers at most.

COS dimension: The same event and prefix/suffix rule of one COS bucket can be bound to only one function.

## Function Runtime Environment Limits

Item	Quota Limit
Allocated memory	Minimum: 64 MB, maximum: 3,072 MB, in increments of 128 MB starting from 128 MB
Temporary cache space; i.e., size of the <code>/tmp</code> directory	512 MB
Timeout period	Minimum: 1 second, maximum: 900 seconds
Number of file descriptors	1,024
Total processes and threads	1,024
Sync request event size	6 MB
Sync request response size	6 MB

Async request event size	128 KB
--------------------------	--------

**Note:**

If the size of a Base64-encoded file is below 6 MB, you can pass the encoded file to SCF through [API Gateway](#); otherwise, we recommend you upload the file to [COS](#) and pass the object address to SCF first. Then, SCF will pull the file from COS to complete the upload.

# Exceeded Quota Management

Last updated : 2024-12-02 20:07:18

For the SCF quotas that exceed the limit, the corresponding solutions are as follows:

Content	Solution
The number of namespaces per region reaches the upper limit.	Please <a href="#">submit a ticket</a> to increase the limits.
The number of functions per namespace reaches the upper limit.	There are multiple namespaces in each region. If the functions quota of a namespace reaches the upper limit, the function quotas of other namespaces can be used first. If the number of namespaces and functions in the current region reaches the upper limit, you can <a href="#">submit a ticket</a> to increase the limits.
The number of triggers per function reaches the upper limit.	It is recommended to to split a function into multiple functions by separating the function business logic and bind them to different triggers. If the function cannot be split to multiple functions and bound to different triggers due to business needs, please <a href="#">submit a ticket</a> to increase the limits.
The total concurrency quota of functions per region reaches the upper limit.	Please adjust the Total concurrency quota in the current region in the Concurrency Management > Concurrency Quota page.
The initialization timeout period of the function exceeds the limits.	Please <a href="#">submit a ticket</a> to increase the limits.

# Managing Functions

## Function Overview

Last updated : 2024-12-02 20:07:18

A function is the basic unit of management and operation in SCF, which usually consists of a series of configuration items and executable code/packages. You can trigger a function through APIs. You can also pass different events to a function through different triggers to trigger it for event processing.

## Relevant Concepts of Function

### Regions

A function resource must belong to a certain region. For regions supported by SCF, see [Billing Overview](#).

### Namespace

A function resource must be created under a certain namespace in a certain region. Each region has a `default` namespace. You can also create namespaces, whose names cannot be modified after creation.

### Function name

It is the unique identifier of a function, must be unique under the same namespace, and cannot be modified after creation.

### Function type

SCF supports two function types: event-triggered function and HTTP-triggered function.

Event-triggered functions are triggered by events in a specified format, such as scheduled triggering events and COS triggering events. For more information on the event structure, see [Trigger Overview](#).

HTTP-triggered functions focus on optimizing web services and can directly accept and process HTTP requests. For more information, see [Function Overview](#).

### Time zone

SCF uses the UTC time by default, which you can modify by configuring the `TZ` environment variable. After you select a time zone, the `TZ` environment variable corresponding to the time zone will be added automatically.

### Runtime environment

Execution environment of the function code. Currently, SCF supports [Python](#), [Node.js](#), [PHP](#), [Java](#), [Go](#), [Custom Runtime](#), and [image deployment](#).

## Function execution method

The execution method specifies the starting file and function while invoking the function. There are three ways as follows:

For Go programming, use the "[FileName]" format, such as `main`.

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

`index.main_handler`.

### Note:

Note that FileName does not include the file name extension, and FunctionName is the name of the entry function. Make sure that the file name extension matches the programming language. For example, for Python programming, the file name extension is `.py`, and for Node.js programming, the file name extension is `.js`.

For Java programming, use the "[package].[class]::[method]" format, such as

`example.Hello::mainHandler`.

## Function description

It is used to record information such as the purpose of the function, which is optional.

# Relevant Configurations of Function

In addition to the above configuration items, you can also modify the following configuration items for function execution by editing the function configuration in the console or [updating function configuration](#):

## Resource type

The computing power supported by SCF includes CPU and GPU.

## Resource specification

It sets the specifications of resources, such as different memory sizes for CPU and different card types for GPU.

## Initialization timeout period

Maximum initialization duration of the function between 3 and 300 seconds (90 seconds for image deployment-based functions and 60 seconds for other functions by default).

### Note:

The function initialization phase includes the preparations of function code, image, layer, and other relevant resources and execution of the main process code of the function. If your function has a larger image or complex business logic, increase the initialization timeout period appropriately.

The initialization timeout period only takes effect in the scenario where the triggered instance is cold started for invocation.

The client waiting time is better to be slightly larger than the sum of the initialization timeout period and the execution timeout period.

### Execution timeout period

Maximum execution duration of the function between 1 and 900 seconds (3 seconds by default).

### Environment variable

It can be defined in the configuration and obtained from the environment when the function is executed. For more information, see [Environment Variables](#).

### Execution role

It grants the corresponding permissions of the policy contained in it to the function. For more information, see [Role and Authorization](#). For example, to execute the action of writing an object into COS in the function code, you should configure an execution role with the permission to write COS.

### Log configuration

It delivers function invocation logs to the specified log topic. For more information, see [Log Search Guide](#).

### Network configuration

It configures the function network access permissions. For more information, see [Network Configuration Management](#).  
Public network: It is enabled by default. The function cannot access public network resources after it is disabled.  
Fixed outbound IP: After it is enabled, the platform will assign a fixed public network outbound IP to the function.  
VPC: After it is enabled, the function can access resources in the same VPC.

### File system

After it is enabled, the function can access resources of the mounted file system. For more information, see [Mounting CFS File System](#).

### Execution configuration

The execution configuration includes async execution, status tracking, and async execution event management. For more information, see [Execution Configuration](#).

Async execution: After it is enabled, the function execution timeout period can be up to 24 hours. It cannot be modified after function creation.

Linkage trace: It can be enabled only for async execution. When it's enabled, it will keep the logs of real-time status of response for async function events. You can query and stop the event and check the related statistics. Data of event status will be retained for three days.

### Async invocation configuration



[Async invocation configuration](#): You can use this configuration item to set the retry policy for async invocation. You can also configure the [dead letter queue](#) to collect error event information and analyze the failure cause.

### Application performance monitoring

After it is enabled, SCF will report the basic execution duration of the function to the specified APM system. You can also instrument the function code for custom reporting. This helps you better track and monitor the execution of the function.

### DNS configuration

In SCF use cases, DNS delays may cause function execution timeouts, affecting the normal business logic. In case of frequent function invocations, the resolution of the DNS server may exceed the frequency limit, which also leads to function execution failures. SCF provides the DNS cache configuration to solve these problems, which can improve the DNS efficiency and mitigate the impact of various factors such as network jitter on the DNS success rate. For more information, see [DNS Caching Configuration](#).

## Executable Operations for a Function

[Creating function](#): Creates a function.

[Updating function](#):

Updating function configuration: Updates the configuration items of the function.

Updating function code: Updates the execution code of the function.

[Getting details](#): Gets function configuration, trigger, and code details.

[Testing function](#): Triggers the function in a sync or async manner as needed.

[Getting log](#): Gets the log of function execution and output.

[Deleting function](#): Deletes a function that is no longer needed.

Copying function: Copies a function to the specified region, name, and configuration.

Function trigger-related operations include:

[Creating trigger](#): Creates a trigger.

[Deleting trigger](#): Deletes an existing trigger.

[Enabling/Disabling trigger](#): Disables a trigger to temporarily prevent a function from being triggered by an event occurring at the event source.

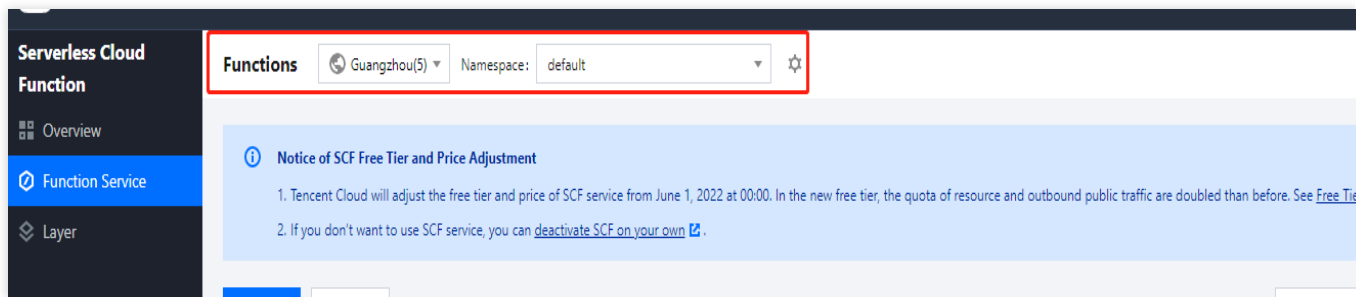
# Create a Function

Last updated : 2024-12-02 20:07:18

SCF offers multiple function creation methods. This document describes how to create a function through the console and command line tool.

## Creating functions via the console

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the **Functions** page and click **Create** to enter the function creation process as shown below:



3. On the **Create function** page, select a function creation method as needed.

**Template:** You need to enter the required function name and use configuration items in the function template to create the function.

**Create from scratch:** You need to enter the required function name and runtime environment to create the function.

**Use TCR image:** You can create a function based on a TCR image. For more information, see [Usage](#).

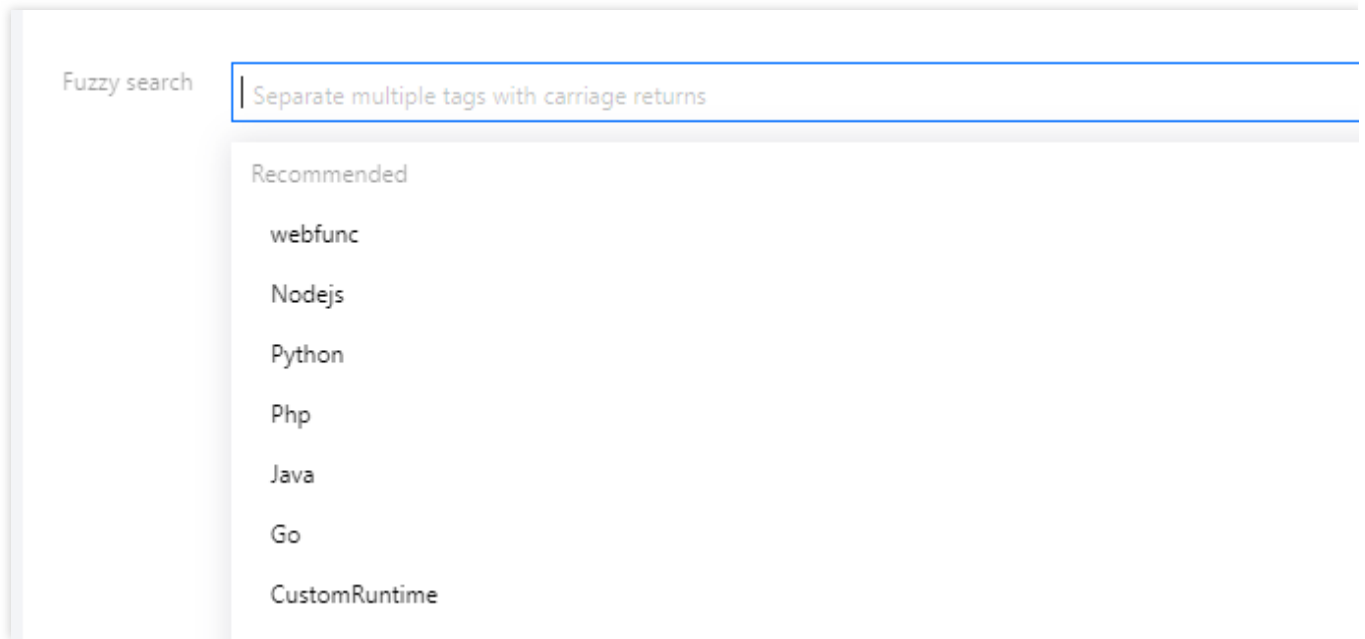
4. Configure the basic information of the function.

Create from template

Create from scratch

Use TCR image

1. Add a tag in **Fuzzy search** to find a template as shown below:



The screenshot shows a 'Fuzzy search' input field with a blue border. Inside the field, the text 'Separate multiple tags with carriage returns' is visible. Below the input field, a dropdown menu is open, displaying a list of recommended runtimes: 'webfunc', 'Nodejs', 'Python', 'Php', 'Java', 'Go', and 'CustomRuntime'. The dropdown menu has a light gray background and a thin border.

2. Select the template and click **Next**.

3. Enter the basic information of the function.

**Function name:** The function name is automatically populated by default and can be modified as needed.

**Region:** The region is automatically populated by default and can be modified as needed.

**Time zone:** SCF uses the UTC time by default, which you can modify by configuring the `TZ` environment variable. After you select a time zone, the `TZ` environment variable corresponding to the time zone will be added automatically.

Enter the basic information of the function.

**Function type:** Select **Event-triggered function** or **HTTP-triggered function**.

Event-triggered function: Receives JSON-formatted events from TencentCloud API or various triggers to trigger the function execution. For more information, see [Basic Concepts](#).

HTTP-triggered function: Directly receives HTTP requests to trigger the function execution in web service scenarios. For more information, see [Function Overview](#).

**Function name:** The function name is automatically populated by default and can be modified as needed.

**Region:** The region is automatically populated by default and can be modified as needed.

**Runtime environment:** The runtime environment is automatically populated by default and can be modified as needed.

**Time zone:** SCF uses the UTC time by default, which you can modify by configuring the `TZ` environment variable. After you select a time zone, the `TZ` environment variable corresponding to the time zone will be added automatically.

Enter the basic information of the function.

**Function type:** Select **Event-triggered function** or **HTTP-triggered function**.

Event-triggered function: Receives JSON-formatted events from TencentCloud API or various triggers to trigger the function execution. For more information, see [Basic Concepts](#).

HTTP-triggered function: Directly receives HTTP requests to trigger the function execution in web service scenarios. For more information, see [Function Overview](#).

**Function name:** The function name is automatically populated by default and can be modified as needed.

**Region:** Select the region where the function is deployed, which must be the same as the region where the image repository is located.

**Time zone:** SCF uses the UTC time by default, which you can modify by configuring the `TZ` environment variable. After you select a time zone, the `TZ` environment variable corresponding to the time zone will be added automatically.

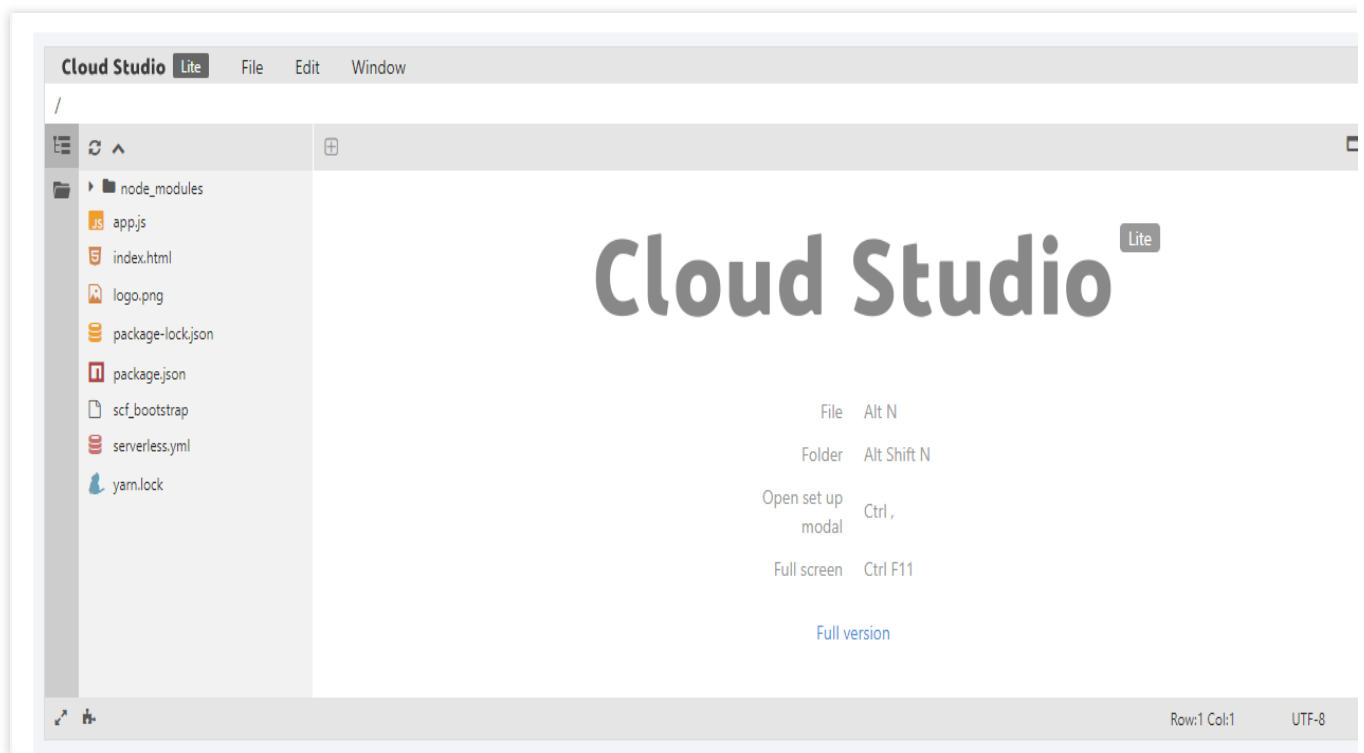
5. Configure the function code.

Create from template

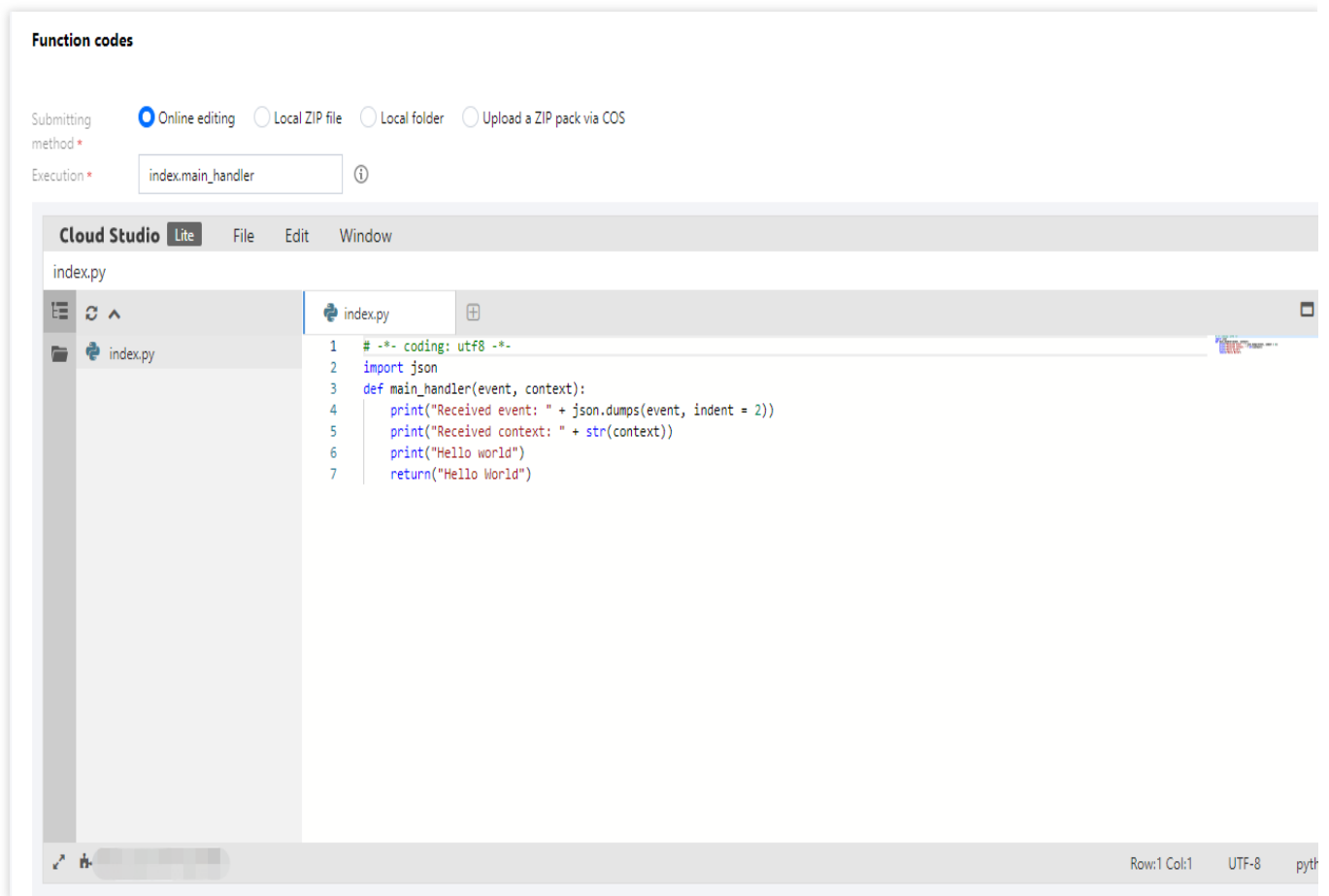
Create from scratch

Use TCR image

The runtime environment and execution method are automatically populated by default as shown below:



Select the function code submitting method and execution method as shown below:



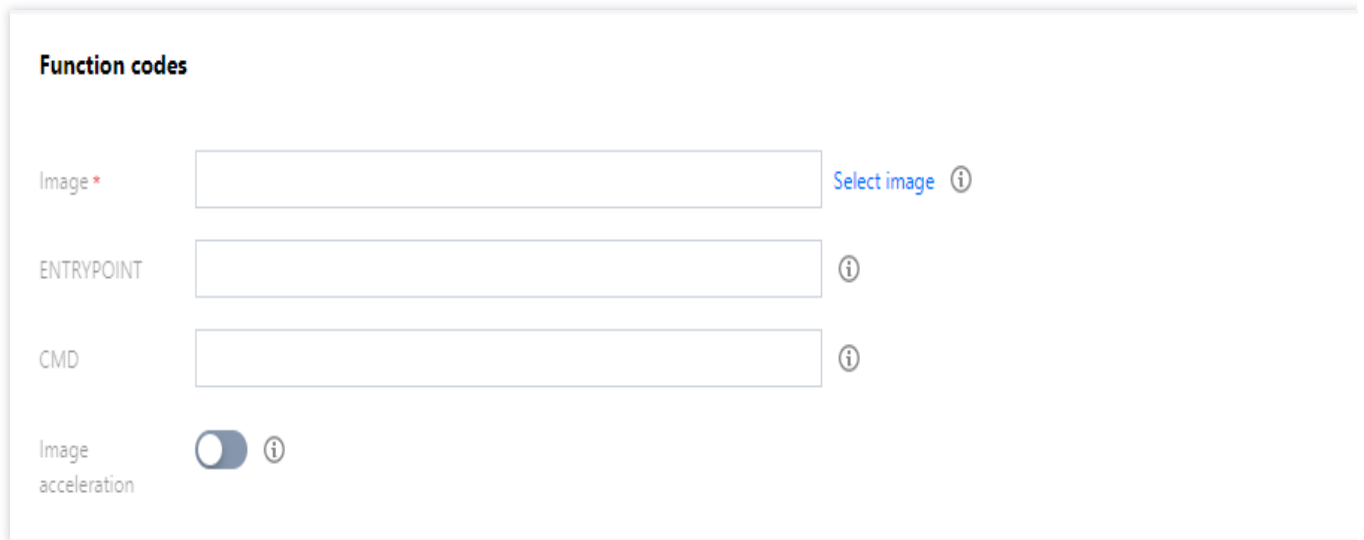
**Submitting method:** Online editing, Local ZIP file, Local folder, and Upload a ZIP pack via COS are supported.

For scripting languages: You can directly use the function code editor.

For non-scripting languages: You can submit the function code by uploading a zip package or through COS and then edit it.

**Execution:** It specifies the starting file and function while invoking the cloud function. For more information, see [Function Overview](#).

Enter the image information as shown below:



**Function codes**

Image \*  [Select image](#) ⓘ

ENTRYPOINT  ⓘ

CMD  ⓘ

Image acceleration ☐ ⓘ

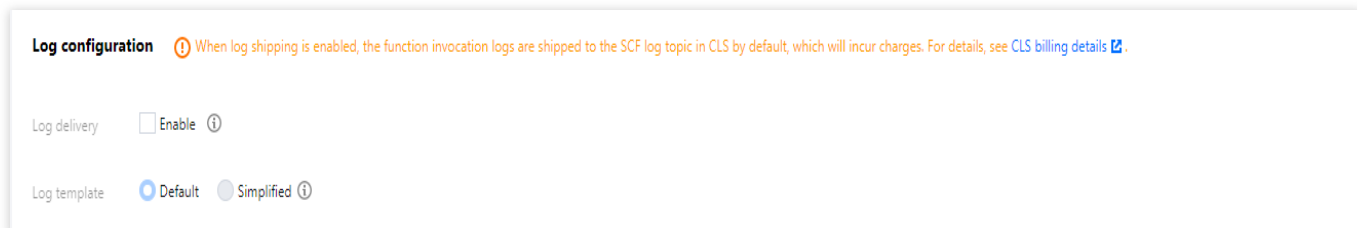
**Image:** Select an image already built in the image repository in the current region.

**ENTRYPOINT:** (Optional) Set the startup command of the container. Enter an executable command (such as python). If it's left blank, the Entrypoint in the Dockerfile is used.

**CMD:** (Optional) Set the startup parameters of the container. Separate each parameter with a space. If it's left blank, the CMD in the Dockerfile is used.

**Image acceleration:** It is disabled by default. After it is enabled, SCF will pulling image much more quickly. It takes over 30 seconds to enable this option; therefore, wait patiently.

6. In **Log configuration**, choose whether to enable log delivery as shown below:



**Log configuration** ⓘ When log shipping is enabled, the function invocation logs are shipped to the SCF log topic in CLS by default, which will incur charges. For details, see [CLS billing details](#).

Log delivery ☐ Enable ⓘ

Log template ☒ Default ☐ Simplified ⓘ

Log delivery is disabled by default. After it is enabled, the execution logs of the function can be delivered to the specified location in real time. For more information, see [Log Delivery Configuration](#).

**Note:**

Currently, you cannot select the log template for image-based functions and HTTP-triggered functions.

7. In **Advanced configuration**, configure the environment, permission, layer, and network of the function as needed. For more information, see [Function Overview](#).

8. In **Trigger configurations**, choose whether to create a trigger. If you select **Custom**, see [Trigger Overview](#).

9. Click **Complete**. You can view the created function on the [Functions](#) page.

## Creating Function on CLI

You can create a function as needed in more ways as detailed below:

Use Serverless Cloud Framework CLI to create a function. For more information, see [Creating Functions on Serverless Cloud Framework](#).

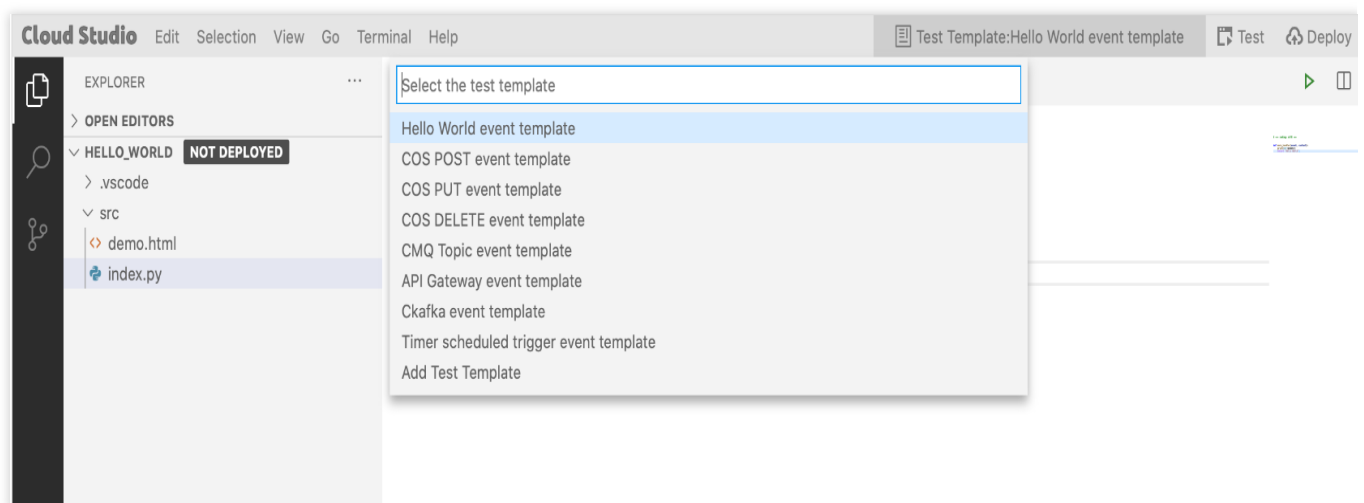
# Testing a Function

Last updated : 2024-12-02 20:07:18

SCF provides a function test feature in the console. With it, you can invoke a function to trigger events, and then check the result, response, and logs.

## Directions

1. Log in to the [SCF console](#) and select **Functions** on the left sidebar.
2. On the **Functions** page, click the target function to enter its details page.
3. Select **Function codes** on the **Function management** page.
4. Select the target test template in the **Editor**.



5. Click **Test** to test the function.

## Preset Testing Event Templates

A testing event template is used to simulate the event and content passed to a function when the corresponding trigger triggers the function, which is represented as the `event` input parameter in the function. The template must be in JSON format. SCF provides the following templates for now.

**Hello World:** This is a simple custom event template that allows you to enter custom event content when you trigger a function through TencentCloud API.

**Files uploaded to/deleted from a COS bucket:** This simulates an event that is generated and passed upon function triggering when a file is uploaded or deleted in a bucket.



**CMQ topic:** This simulates an event that is generated and passed upon function triggering when a message is received in CMQ.

**API Gateway:** This simulates an event that is generated and passed upon function triggering when an API request arrives API Gateway.

## Custom Testing Event Template

You can modify the preset templates based on your own requirements, and save them as your custom templates.

### Use limits

Custom testing event templates are configured at the account level, which means the templates are shared by functions under the same account.

Up to 5 custom testing templates can be configured for a single account.

A custom testing template can be up to 64 KB in size.

### Creating custom testing event templates

Select a preset template and click **Create Template**. Make changes as you want, specify a new template name and save it as a custom template. This template is used by default next time when you enter the test page.

### Deleting custom testing event templates

To delete a custom template that is no longer used, select the template and click **Delete**.

# Debugging Function

Last updated : 2024-12-02 20:07:18

Online debugging is now supported in the SCF Console, so you can debug and locate problems in the console easily.

## Note:

Currently, online debugging can only be performed in Chrome and only supports Node.js 10.15 and Node.js 12.16.

## Enabling Debugging Mode

### Note

:

Before using the online debugging feature, you need to manually enable the debugging mode for the function. **Doing so will change part of the function's original configuration**, which will be restored after the debugging mode is disabled. This may affect your business; therefore, please be sure to understand the following:

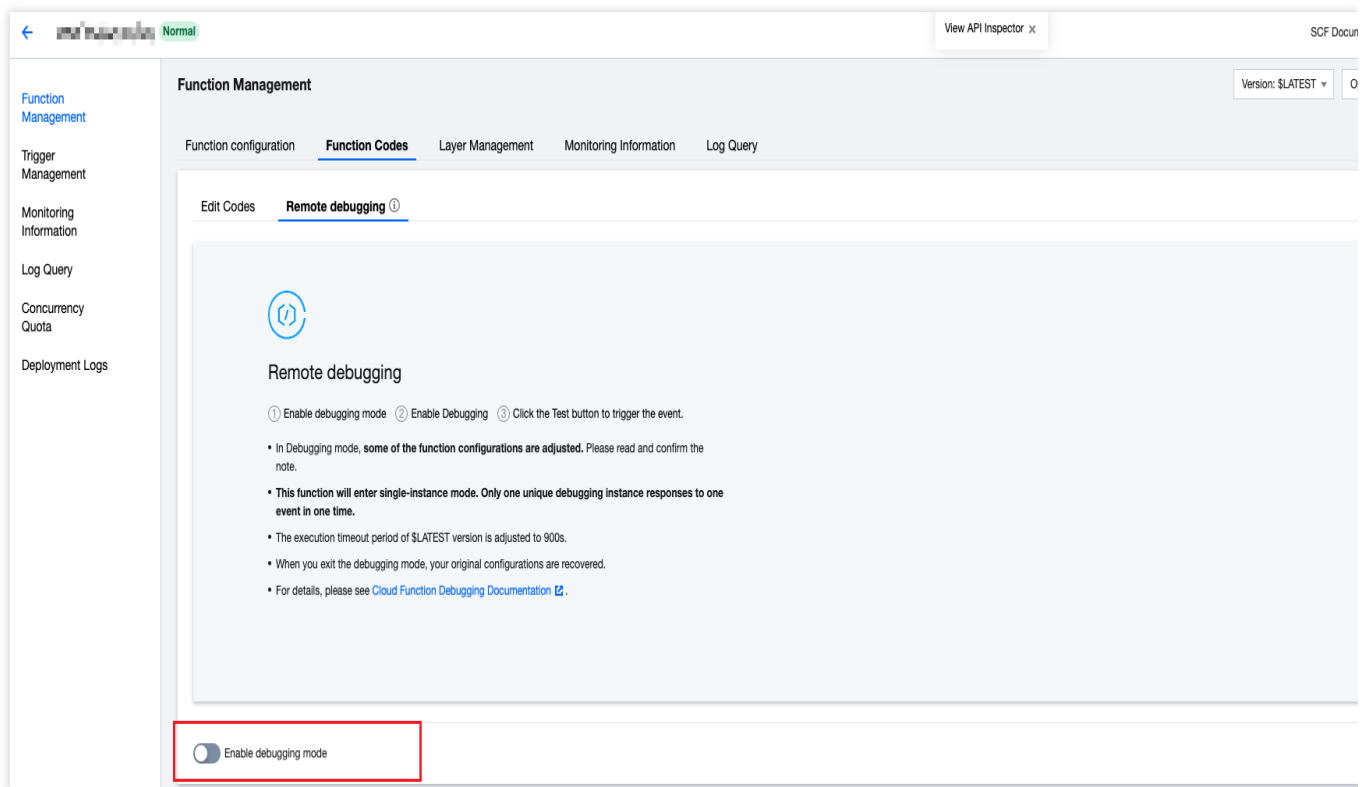
The function will enter the single-instance mode. All its versions can respond to only one event at a time, and if the concurrency limit is exceeded, events will fail.

The execution timeout period is adjusted to 900s and cannot be set during debugging.

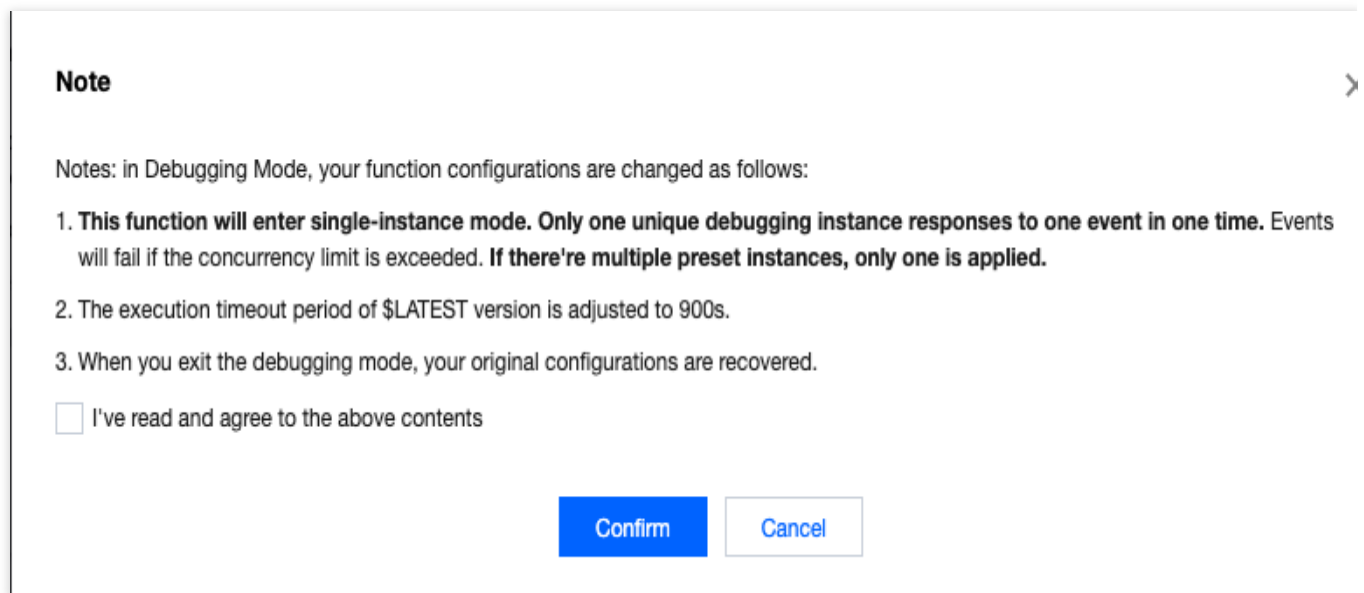
Multiple preset instances will be reduced to one single instance.

The execution performance of the function will be reduced after the debugging mode is enabled.

1. Log in to the [SCF Console](#) and select "Function Service" on the left sidebar.
2. At the top of the "Function Service" page, select the region of the function for which you want to enable the debugging mode and click the function name to enter the function details page.
3. Select **Function Code > Remote debugging** on the "Function Management" page and click **Enable debugging mode** as shown below:



4. Click **Confirm** in the pop-up window to enable the debugging mode as shown below:



## Debugging

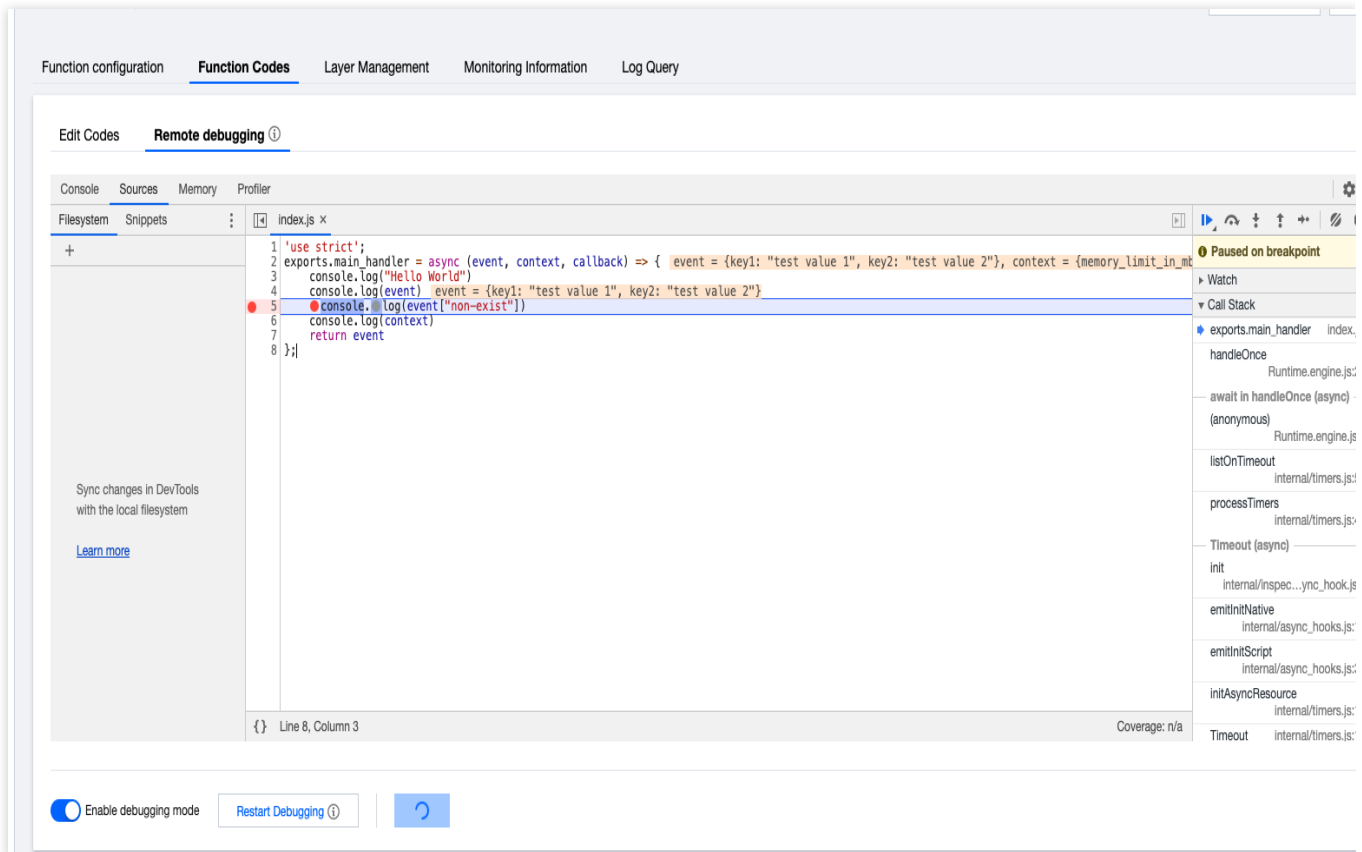
1. After [the debugging mode is enabled](#), debugging will start automatically once the function is updated.

### Note:

After the debugging mode is enabled, when you enter the debugging page again, you need to manually select **Start debugging**.

2. After the loading is completed, the page will automatically display the entry file. To open any file you want, use the keyboard shortcut Cmd + P (macOS) or Ctrl + P (Windows).

3. You can set breakpoints as needed and click **Test** to trigger the test based on the test template as shown below:



### Note:

For more information on debugging tools, please see Chrome DevTools.

## Disabling Debugging Mode

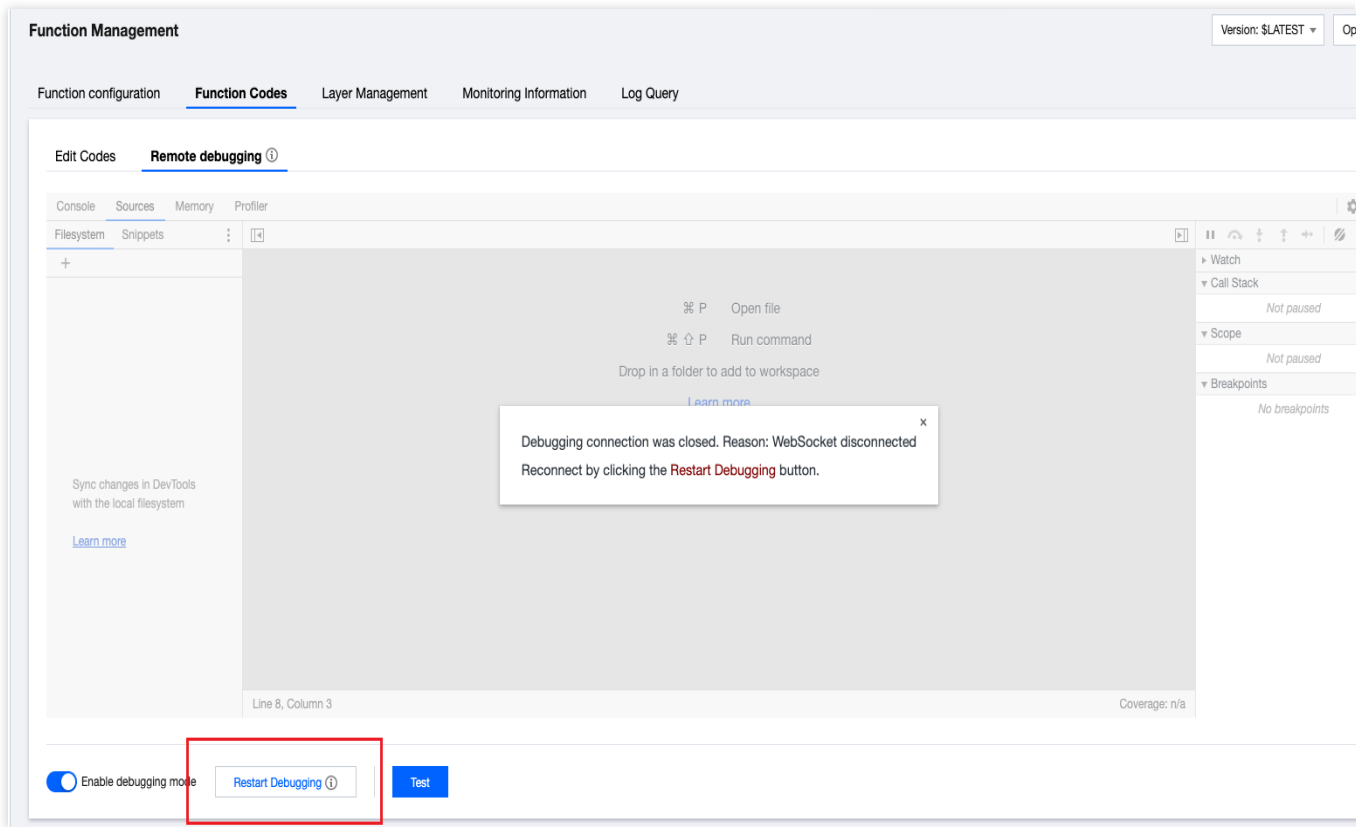
1. Select **Function Code > Remote debugging** on the "Function Management" page.
2. Toggle off **Enable debugging mode** to disable the debugging mode, and the original function configuration will be restored.

### Note:

Code changes made on the debugging page will not be synced to the cloud. If you want to save the changed code, please save the changes and use the online code editing feature.

## FAQs

Due to network and code exceptions, the inspector may be disconnected. When an error like the one in the following figure occurs, you need to click **Restart debugging** to reconnect.



If your function runs normally, but you encounter an OOM error in the debugging mode, you need to increase the memory configured for the function. This way, you can solve the problem of insufficient memory caused by the fact that the function requires more memory in the debugging mode.

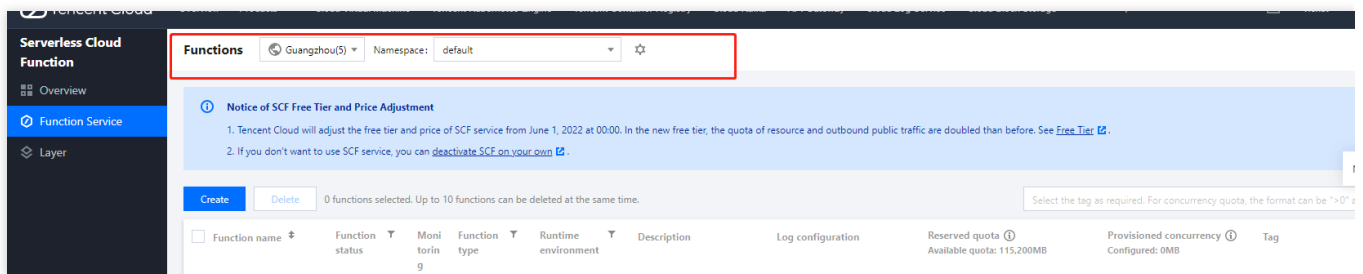
# Querying a Function

Last updated : 2024-12-02 20:07:18

A function can be queried in the console or on Serverless Cloud Framework CLI.

## Viewing a Function in the Console

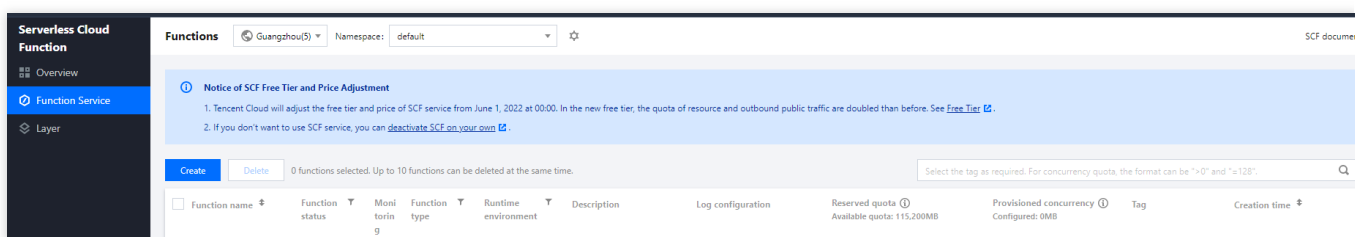
1. Log in to the [SCF console](#) and select **Functions** on the left sidebar.
2. Select the region and namespace for which to view functions at the top of the **Functions** page. In the function list, you can view all the functions in the specified region and namespace as shown below:



3. The function list displays the function name, monitoring information, function type, runtime environment, log configuration, and creation time. You can customize the displayed fields as needed by clicking



on the right as shown below:



In the pop-up window, select the fields you want to display and click **OK** as shown below:

### Display settings

Select the columns you want to display. With your screen resolution, up to 18 columns can be selected (13 selected now).

☒ Function name

☐ Public access configuration

☒ Function status

☐ Private access configuration

☒ Monitoring

☐ Permission configuration

☒ Function type

☒ Log configuration

☒ Runtime environment

☐ File system

☒ Description

☐ Execution configurations

☒ Reserved quota

☒ Provisioned concurrency

☒ Tag

☒ Creation time

☒ Last modified

☐ Operation

OK

4. Click the function name to enter the function details page as shown below:

Function management

Version management

Alias Management

Trigger management

Monitoring information

Log Query

Function management

Version: SLATEST

Function configuration

Function codes

Layer management

Monitoring information

Log Query

Bind

Sort

Priority	Layer name	Version	Description	Runtime environment	Operation
No data yet					

The function details page contains the following information:

**Function management:** View and manage the configurations, [codes](#), and [layers](#) of the function.

**Version management:** Fixate the code and configuration of the function by publishing a version. For more information, see [Overview](#).

**Alias management:** Use an alias to call the bound function.

**Trigger management:** View the triggers configured for the function and create triggers. For more information, see [Creating Triggers](#).

**Monitoring information:** View the monitoring information of function execution. For more information, see [Descriptions of monitoring metrics](#).

**Log query:** View the execution logs of the function and filter logs by certain criteria. For more information, see [Viewing Execution Logs](#).

**Concurrency quota:** View the concurrency quota of the function and set the reserved quota and provisioned concurrency of the function. For more information, see [Concurrency Overview](#).

**Deployment logs:** View the deployment logs of the function.

## Getting Deployment Information on Serverless Cloud Framework CLI

### Note:

Before starting, install the Serverless Cloud Framework CLI tool first as instructed in [Installation](#).

You can run the `scf info` command on Serverless Cloud Framework CLI to view deployment information.



# Query a Function Execution Log

Last updated : 2024-12-02 20:07:18

Function execution logs can be queried in the console, and many filtering methods can be used to get the desired logs.

1. Log in to the [SCF console](#) and select **Function Service** on the left sidebar.
2. Select the region for which to view functions at the top, and click the name of a function in the function list to enter the function details page.
3. Open the function query tab to query the function execution logs. You can view all logs, success logs, and failure logs by choosing the function execution result at the top left. Logs with the specified request ID can be queried in the query window at the top right.

# Deleting a Function

Last updated : 2024-12-02 20:07:18

A function can be deleted in the console or on Serverless Framework CLI.

## Deleting Function in Console

1. Log in to the [SCF Console](#) and select "Function Service" on the left sidebar.
2. Select the region for which to view functions at the top. In the function list, you can view all the functions in the specified region.
3. On the list page, click **Delete** in the "Operation" column on the right of the function you want to delete and then click **OK** to delete it.

## Deleting Function on Serverless Framework CLI

### Note:

Before starting, please install the Serverless Framework CLI tool first as instructed in [Installing Serverless Framework](#).

You can run the `sls remove` command on Serverless Framework CLI to delete a deployed project.

# Deploying Functions

Last updated : 2024-12-02 20:07:18

## Deployment Through Console

A deployment package is a zip collection of all the code and dependencies that SCF runs, which should be specified during function creation. You can create a deployment package in your local environment and upload it to SCF or write the code directly in the SCF Console which will create and upload the deployment package for you. Please use the following criteria to determine whether you can use the console to create a deployment package:

**Simple scenario:** if your custom code only needs to use standard libraries or SDK libraries such as COS and SCF libraries provided by Tencent Cloud, and there is only one code file, you can use the online editor in the SCF Console which automatically compresses the code and associated configuration information into an operable deployment package.

**Advanced scenario:** if your code requires additional resources (such as a graphics library for image processing, web framework for web programming, or database connection library for executing database commands), you need to first create a function deployment package in your local environment and then upload it to the console.

### Packaging requirements

#### Zip format

The code package submitted directly to the SCF platform or submitted by uploading to COS and then imported into SCF must be in [zip format](#). 7-Zip can be used on Windows and zip command line tools can be used on Linux for compression or decompression.

#### Packaging method

When packaging, you need to package the code files but not the entire directory of the code; after packaging is completed, the entry function file must be in the root directory of the package.

When packaging on Windows, you can enter the function code directory, select all files, right-click and select "Send to > Compressed (zipped) folder" to create the deployment package. If a tool such as 7-Zip is used to unzip and browse the package, the entry function and other libraries should be included in the package.

When packaging on Linux, you can enter the function code directory, run the `zip` command, and specify the source files as all files in the code directory to generate the deployment package, such as `zip /home/scf_code.zip *`  
`-r` .

### Sample deployment package

The following sample illustrates the steps to create a deployment package in a local environment.

#### Note:

Normally, locally installed dependent libraries work well on the SCF platform. In rare cases, the binary files installed may have compatibility problems. If this happens, please [contact us](#).

For the Python programming language in the sample, libraries and dependencies will be installed locally using the pip tool. Please make sure that you have already installed Python and pip locally.

## Python deployment package

### Creating Python deployment package on Linux

1. Create a directory:

```
mkdir /data/my-first-scf
```

2. Save all the Python source files (.py files) of the function into this directory.

3. Install all dependencies to this directory using pip:

```
pip install <module-name> -t /data/my-first-scf
```

For example, the following command installs the Pillow library in the `my-first-scf` directory:

```
pip install Pillow -t /data/my-first-scf
```

4. Compress everything in the `my-first-scf` directory. Please note that you need to compress the content of the directory but not the directory itself:

```
cd /data/my-first-scf && zip my_first_scf.zip * -r
```

#### Note:

For libraries with compilation process, we recommend you perform packaging on CentOS 7 to maintain consistency with the SCF runtime environment.

If there are requirements for other software, compilation environment, or development libraries during installation or compilation, please solve the compilation and installation problems as instructed.

### Creating Python deployment package on Windows

We recommend you package the dependent packages and codes that have already been successfully executed on Linux into a zip package as the execution code of the function. For more information, please see [Code Practice - Getting Images in COS and Creating Thumbnails](#).

In Windows, you can use the `pip install <module-name> -t <code-store-path>` command to install Python libraries. But for packages that need to be compiled or have static or dynamic libraries, libraries compiled and generated on Windows cannot be invoked in SCF runtime environment (CentOS 7), so only libraries completely implemented in Python can be installed on Windows.

# Deployment Through Serverless Cloud Framework

**Note:**

Before starting, please install the Serverless Cloud Framework tool first as instructed in [Installing Serverless Cloud Framework](#).

You can run `scf deploy` on Serverless Cloud Framework to deploy the function.

# Web Function Management

## Function Overview

Last updated : 2024-12-02 20:07:18

HTTP-triggered function is a function type in SCF. Compared with event-triggered function that has limits on the event format, HTTP-triggered function focuses on optimization of web service scenarios and can directly send HTTP requests to URLs to trigger function execution.

## Features and Advantages

In terms of the support for web service scenarios, HTTP-triggered function excels event function in the following aspects:

Functions can directly receive and process HTTP or WebSocket requests, so API Gateway doesn't need to convert the requests to JSON format, which reduces the request processing steps and improves the web service performance.

The writing experience of HTTP-triggered function is closer to that of native web services, and native Node.js APIs can be used to deliver a service experience consistent with that of local development.

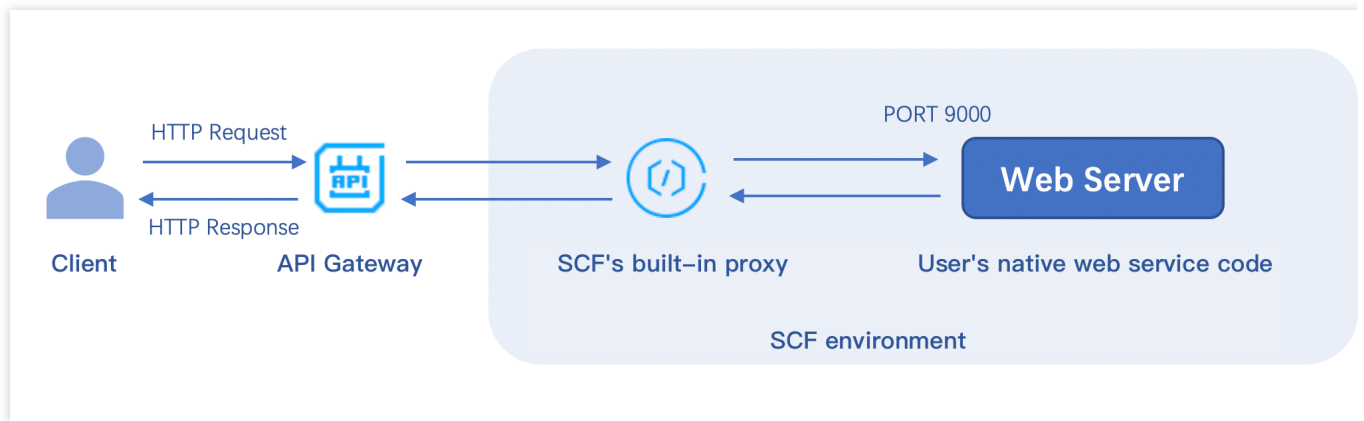
A rich set of frameworks are supported. You can use common web frameworks, such as Node.js web frameworks `Express` and `Koa` , to write HTTP-triggered functions. You can also quickly migrate your local web framework services to the cloud with minimal modification.

An HTTP-triggered function can automatically create an API Gateway service for you. After the deployment is completed, API Gateway will automatically generate a default URL for user access and invocation, which reduces the learning costs and simplifies debugging.

The SCF console provides testing capabilities for you to quickly test your services.

## How It Works

How an HTTP-triggered function works is as shown below:



After your HTTP request passes API Gateway, when directly passing through the native request, API Gateway will add the content required by the gateway to trigger the function, such as function name and function region, to the request header and pass the modified request to the function environment to trigger the backend function.

In the function environment, the built-in proxy is used to implement Nginx-based forwarding, remove the request information not required by the service specification from the header, and send the native HTTP request to your web server service through the specified port.

After being configured with the specified listening port `9000` and service bootstrap file, your web server will be deployed in the cloud and use this port to get HTTP requests for processing.

## Usage Limits

### Feature limits

Currently, HTTP-triggered functions can be bound to only API Gateway triggers.

A function can be bound to multiple API Gateway triggers, but all APIs must be under the same API service.

Async invocations and retries are not supported.

In the Tencent Cloud standard environment, only the `/tmp` directory is readable and writable. When outputting files, please select the `/tmp` path; otherwise, the service will exit exceptionally due to the lack of write permission. For projects that requires compression for deployment (JAVA, Go), please make sure that `scf_bootstrap` is included in the ZIP package.

### Request limits

HTTP-triggered functions can be invoked only through API Gateway but not function APIs.

`Response headers` has the following limits:

The total size of all `key` and `value` values cannot exceed 4 KB.

The size of `body` cannot exceed 6 MB.

When deploying your web service, you must listen on the specified `9000` port and cannot listen on the internal loopback address `127.0.0.1`.

Currently, the `Connection` field in the HTTP request header cannot be customized.

## Common Function Request Headers

The common request headers received by your web server from the function environment are as detailed below, none of which can be customized:

Header Field	Description
X-Scf-Request-Id	Current request ID
X-Scf-Memory	Maximum memory that can be used during function instance execution
X-Scf-Timeout	Timeout period for function execution
X-Scf-Version	Function version
X-Scf-Name	Function name
X-Scf-Namespace	Function namespace
X-Scf-Region	Function region
X-Scf-Appid	<code>Appid</code> of function owner
X-Scf-Uin	<code>Uin</code> of function owner



# Creating and Testing Function

Last updated : 2024-12-02 20:07:18

## Overview

This document describes how to quickly create and use a web function in the SCF console.

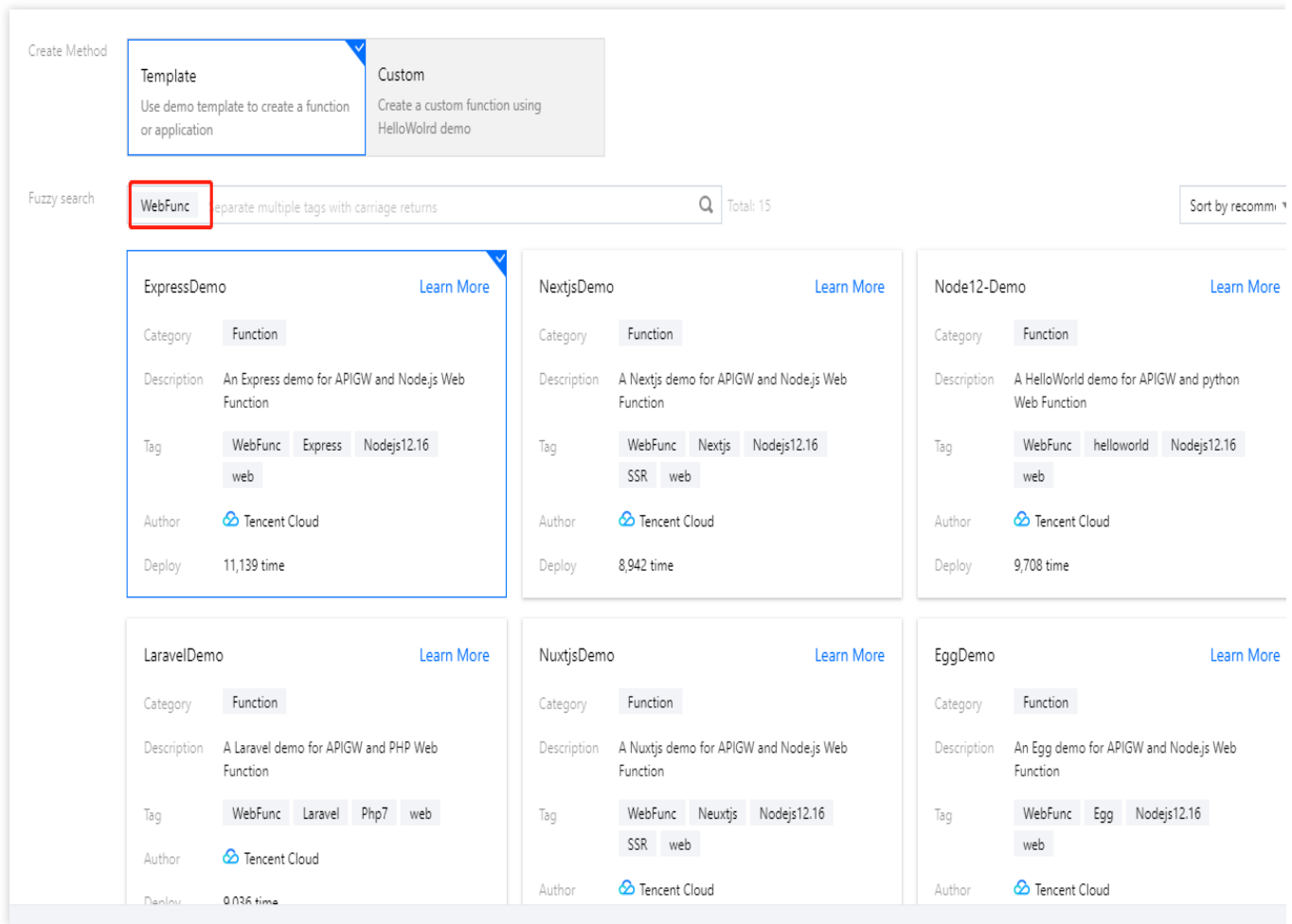
## Prerequisites

Before using SCF, you need to sign up for a [Tencent Cloud account](#) and complete [identity verification](#) first.

## Directions

### Creating function from template

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template** for **Creation Method**, enter `WebFunc` in the search box to filter all web function templates, select the one you want to use, and click **Next** as shown below:



4. On the **Configuration** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**.

After creating the web function, you can view its basic information on the **Function Management** page and access it at the access path URL generated by API Gateway.

## Creating custom function

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Custom Creation** for **Creation Method** and enter the basic function information as shown below:

The screenshot shows the 'Create Method' section with two options: 'Template' (Use demo template to create a function or application) and 'Custom' (Create a custom function using HelloWolrd demo). The 'Custom' option is selected and highlighted with a blue border and a checkmark. Below this is the 'Basic Configurations' section. It includes a 'Function Type' section with radio buttons for 'Event Function' and 'Web Function'. The 'Web Function' option is selected and highlighted with a red box. Below this is a 'Function name' input field with the text 'helloworld-'. Below that is a 'Region' dropdown menu with 'Guangzhou' selected. At the bottom is a 'Deployment Mode' section with radio buttons for 'Code' and 'Image'. The 'Code' option is selected.

Create Method

Template  
Use demo template to create a function or application

Custom  
Create a custom function using HelloWolrd demo

Basic Configurations

Function Type \*  
☐ Event Function ☒ Web Function ⓘ

Function name \*  
helloworld-

It supports 2 to 60 characters, including letters, numbers, underscores and hyphens. It must start with a letter and end with a number or letter.

Region \*  
Guangzhou

Deployment Mode \*  
☒ Code ☐ Image

**Function Type:** select **Web function**.

**Function Name:** enter the name of your function.

**Region:** enter your function deployment region.

**Deployment Method:** select **Code deployment**.

**Runtime Environment:** select **Nodejs 12.16** since Node.js is used as an example here.

4. In **Advanced Configuration**, view other required configuration items.

**Namespace:** the default namespace is used by default. You can select another namespace for deployment.

**Bootstrap Command:** for web functions, you must configure the `scf_bootstrap` bootstrap file for your project to ensure that the web server can be started properly in the function environment. You can select the default framework template provided by SCF or use a custom template to write your own start command. For more information, please see [Bootstrap File Description](#).

5. In **Trigger Configuration**, only API Gateway trigger is supported currently, and a trigger will be created automatically based on the default configuration.

### Trigger Configurations

Triggered Version	<div>Default Traffic</div>
Trigger Method	<div>API Gateway Trigger</div>
API Service Type ⓘ	<div><input checked="" type="radio"/> Create API Service <input type="radio"/> Use Existing API Service</div>
API Service	<div>SCF_API_SERVICE</div>
Request method ⓘ	<div>ANY</div>
Publishing Environment ⓘ	<div>Publish</div>
Authentication Method ⓘ	<div>No authentication</div>

1. Web functions only support API gateway triggers, which can directly receive HTTP request.

2. You can configure a custom domain name after creating the function. [Learn More](#)

6. Click **Complete**.

After creating the web function, you can view its basic information on the **Function Management** page and access it at the access path URL generated by API Gateway.

## Cloud test

Method 1

Method 2

Method 3

You can open the access path URL in a browser, and if it can be accessed normally, the function is successfully created.

You can concatenate the specified HTTP request for testing on the function code page through the test capability and check whether the function is successfully deployed through the HTTP response.

### Note:

The console invokes and tests the function by using the gateway API. If the test fails, the API will automatically execute the retry logic for up to 4 retries. Therefore, you will see multiple execution logs for one failed request.

You can use other HTTP testing tools such as CURL and Postman to test the web function you have successfully created.

## Viewing log

For web functions, the returned body information of each request will not be automatically reported to the log. You can customize the reporting in the code through statements such as `console.log()` or `print()` in your programming language.

For PHP, as all inputs are automatically used as the body, you need to run the following command to output the log to `stdout` and complete log reporting:

```
<?php
    $stdout = fopen("php://stderr", "w");
    fwrite($stdout, "123\\n");
?>
```

On the details page of a created function, select **Log Query** to view its detailed logs. For more information, please see [Viewing Execution Logs](#).

## View monitoring data

On the details page of a created function, select **Monitoring Information** to view metrics such as function invocations and execution duration. For more information, please see [Monitoring Metric Descriptions](#).

### Note:

The minimal granularity of monitoring statistics collection is 1 minute. You need to wait for 1 minute before you can view the current monitoring record.

## Common Error Codes and Solutions

Common errors are divided into two types: user errors and platform errors.

**User errors:** the execution failure is caused by improper user operations; for example, the sent request does not meet the standard, the commands in the bootstrap file are incorrect, the correct port is not listened on, or the internal business code is incorrectly written. The returned error code is 4xx.

**Platform errors:** the execution failure is caused by internal errors of the SCF platform. The returned error code is 500.

The table below describes the possible scenarios of request errors and function errors, so that you can quickly troubleshoot problems. For more information on error codes, please see [Function Status Code](#).

### 2xx status codes

Status Code	Return Message	Description
200	Success	The function is executed successfully. If you see this return code, but the returned information does not meet your expectations, check whether your code logic is correct.

## 4xx status codes

Status Code	Return Message	Description
404	InvalidSubnetID	The subnet ID is invalid. Check whether the network configuration of the function is correct and whether the subnet ID is valid.
405	ContainerStateExitedByUser	The container process exits normally. Check whether your bootstrap file is correctly written.
406	RequestTooLarge	The request body size is too large. The upper limit for sync request events is 6 MB.
410	The HTTP response body exceeds the size limit.	The size of function response body exceeds the upper limit of 6 MB. Adjust it and try again.
430	User code exception caught	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.
433	TimeLimitReached	The function execution times out. Check whether a large number of time-consuming operations exists in the service code or adjust the timeout period on the <b>Function Configuration</b> page.
439	User process exit when running	The user process exits accidentally. Based on the error message, find out the cause and fix the function code.
446	PortBindingFailed	No listening port is specified. Check whether your business code listens on port <code>9000</code> .
499	kRequestCanceled	The user manually interrupts the request.

## 5xx status codes

Status Code	Return Message	Description
500	InternalServerError	An internal error occurs. Try again later. If the problem persists, <a href="#">submit a ticket</a> .

## Notes on local debugging

When debugging in a local container, in order to ensure consistency with the standard container environment in the cloud, you need to pay attention to the limits of readable and writable files in the local environment. The local container startup command is as follows:

```
docker run -ti --read-only -w /var/user \\  
  -v /usr/local/cloudfunction/runtime:/var/runtime:ro \\  
  -v ${PWD}:/var/user:ro \\  
  -v /tmp:/tmp \\  
  -v /usr/local/cloudfunction/runtime:/var/runtime:ro \\  
  -v /usr/local/cloudfunction/lang:/var/lang:ro \\  
  ccr.ccs.tencentyun.com/cloudfunc/qcloud-func bash
```

# Bootstrap File Description

Last updated : 2024-12-02 20:07:18

An HTTP-triggered function runs in the image environment built in it based on the standard programming language. You must create an executable file `scf_bootstrap` to start your web server, and then package the file with your code files for deployment to create an HTTP-triggered function. During actual request processing, your web server will receive HTTP requests by listening on the specified `9000` port, forward them to the backend service for logic processing, and return the responses to end users.

## Bootstrap File Usage

`scf_bootstrap` is the bootstrap file of your web server and ensures that your web service can normally start and listen on requests. In addition, you can customize `scf_bootstrap` to implement more personalized operations as needed:

Set the paths and environment variables of the runtime's dependency libraries.

Load the dependency library files and extensions of the custom programming language and version. If there are dependent files that need to be pulled in real time, you can download them to the `/tmp` directory.

Parse the function file and execute the global operations or initialization processes (such as initializing SDK client (HTTP client) and creating database connection pool) required before function invocation, so they can be reused during invocation.

Start plugins such as security and monitoring.

### Note:

SCF only supports reading **scf\_bootstrap** as the bootstrap file name, and other names cannot start the service normally.

In the Tencent Cloud standard environment, only the `/tmp` directory is readable and writable. When outputting files, select the `/tmp` path; otherwise, the service will exit abnormally due to the lack of write permission.

## Prerequisites

The permission to execute is required. Make sure that your `scf_bootstrap` file has the 777 or 755 permission; otherwise, it cannot be executed due to insufficient permissions.

It can run on SCF's operating system (CentOS 7.6).

If the bootstrap file is a shell script, it must have `#!/bin/bash` in the first line.

The bootstrap command must be the absolute path

`/var/lang/${specific_lang}${version}/bin/${specific_lang}`; otherwise, it cannot be invoked normally. For more information, see [Absolute Paths in Standard Language Environments](#).

The recommended listening address is `0.0.0.0`, and the internal loopback address `127.0.0.1` cannot be used.

The file must end with an LF carriage return.



## Creation Method

Local package upload

Quick creation in console

You can write your `scf_bootstrap` file locally, make sure that the file permission meets the requirements, package it with the project code, and deploy them together on the HTTP-triggered function.

You can create an HTTP-triggered function in the [SCF console](#).

You can edit your bootstrap file in **Advanced configuration** > **Startup command** during [function creation](#). SCF provides a general enablement template for commonly used web frameworks. You can also modify it as needed as shown below:

### Advanced configuration

Namespace \*

default ▼

Description

Helloworld empty template function

Up to 1000 letters, digits, spaces, commas, and periods.

Startup command \*

Demo template ▼ ⓘ

```
#!/usr/bin/env bash
/var/lang/python37/bin/python3 -u app.py
```

After the creation is completed, the console will automatically package your code and `scf_bootstrap` for deployment.

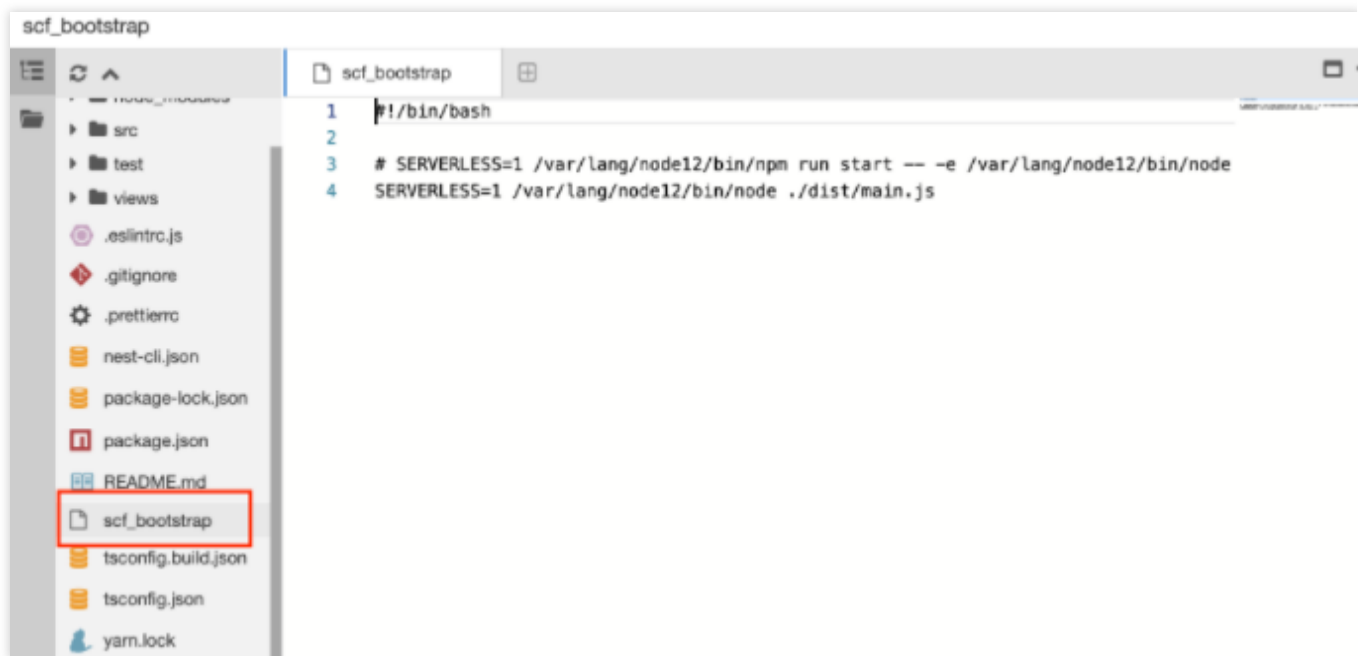
**Note:**

©2013-2025 Tencent Cloud International Pte. Ltd.

Page 45 of 251

The configuration in the console takes effect only if no `scf_bootstrap` is detected in the uploaded code. If there is an `scf_bootstrap` file in your project, the system will deploy the function based on it.

After the deployment is completed, you can view and edit the `scf_bootstrap` file in the code editor as shown below:



### Troubleshooting common errors

Run the `scf_bootstrap` file as the container bootstrap command. You must ensure that the container can normally start and run the code logic. Therefore, make sure that your bootstrap command is written correctly. If you encounter a `405` error code, it is usually caused by the failure to run the file; therefore, also make sure that your bootstrap file is written correctly.

### Absolute Paths in Standard Languages Environments

Language Version	Absolute Path
Node.js 16.13	<code>/var/lang/node16/bin/node</code>
Node.js 14.18	<code>/var/lang/node14/bin/node</code>
Node.js 12.16	<code>/var/lang/node12/bin/node</code>
Node.js 10.15	<code>/var/lang/node10/bin/node</code>
Python 3.7	<code>/var/lang/python37/bin/python3</code>
Python 3.6	<code>/var/lang/python3/bin/python3</code>

Python 2.7	<code>/var/lang/python2/bin/python</code>
PHP 8.0	<code>/var/lang/php80/bin/php</code>
PHP 7.4	<code>/var/lang/php74/bin/php</code>
PHP 7.2	<code>/var/lang/php7/bin/php</code>
PHP 5.6	<code>/var/lang/php5/bin/php</code>
Java 11	<code>/var/lang/java11/bin/java</code>
Java 8	<code>/var/lang/java8/bin/java</code>

## Common Web Server Bootstrap Command Templates

Node.js

Python

PHP

```
#!/bin/bash
export PORT=9000
/var/lang/node12/bin/node app.js # Change to your own bootstrap function name
```

```
#!/bin/bash
export PORT=9000
/var/lang/python3/bin/python3 app.py # Change to your own bootstrap file name
```

```
#!/bin/bash
/var/lang/php7/bin/php -c /var/runtime/php7 -S 0.0.0.0:9000 hello.php # Change
to your own entry function name
```

# Trigger Management

Last updated : 2024-12-02 20:07:18

Currently, HTTP-triggered functions only support **API Gateway triggers**. You can bind API Gateway triggers in the [SCF console](#) or bind backend functions in the [API Gateway console](#).

## Trigger Type Description

For HTTP-triggered functions, triggers support two creation methods: **default creation** and **custom creation**. You can choose an appropriate method according to the actual situation:

Function	Default Creation (Basic API Gateway)	Custom Creation (Standard API Gateway)
Default domain name	Supported	Supported
Binding to custom domain name	Manual binding	Management in API Gateway console
Request method configuration	Supported	Supported
Release environment configuration	Supported	Supported
Authentication method configuration	Supported	Supported
Visibility in API Gateway console	Invisible	Visible
Advanced API Gateway capabilities (such as plugin and dedicated instance)	Not supported	Supported
Billing method	Gateway calls are not billed.	It is billed by standard API Gateway billable items.
Type conversion	The gateway can be upgraded to a standard API Gateway instance. After upgrade, you	The gateway edition cannot be changed. A standard API Gateway

	can use all gateway capabilities and billed by standard API Gateway billable items.	instance cannot be rolled back to a basic API Gateway instance in default creation.
--	---	---

## Trigger Overview

Characteristics of HTTP API Gateway triggers:

### Passthrough HTTP request

After API Gateway receives an HTTP request, if the API Gateway backend is connected with an SCF function, the function will be triggered. At this time, API Gateway will directly pass through the HTTP request, without performing `event` type format conversion. HTTP request information includes the specific service that receives the request, API rule, actual request path, method, and `path` , `header` , and `query` of the request.

### Sync invocation

API Gateway invokes the function synchronously, and it will wait for the function to return before the timeout period configured in it elapses. For more information on invocation types, see [Invocation Types](#).

## Trigger Configuration

Basic gateways can only be bound in the default creation method in the SCF console.

API Gateway triggers can be configured in the [SCF console](#) or the [API Gateway console](#). For more information on trigger configuration, see [Overview](#).

## Trigger Binding Limits

In API Gateway, one API rule can be bound to only one function, but one function can be bound to multiple API rules as the backend. You can create an API with different paths in the [API Gateway console](#) and point the backend to the same function. APIs with the same path, same request method, and different release environments are regarded as the same API and cannot be bound repeatedly.

## Request and Response

Request method is the method to process request sent from API Gateway to SCF, and response method is the method to process the returned value sent from SCF to API Gateway. For HTTP-triggered functions, API Gateway will add the information required for function triggering in the `header` and directly pass through the original request to trigger the backend function.

### Note:

The following parameters don't support custom configuration:

`connection` field

Custom fields starting with `X-SCF-`

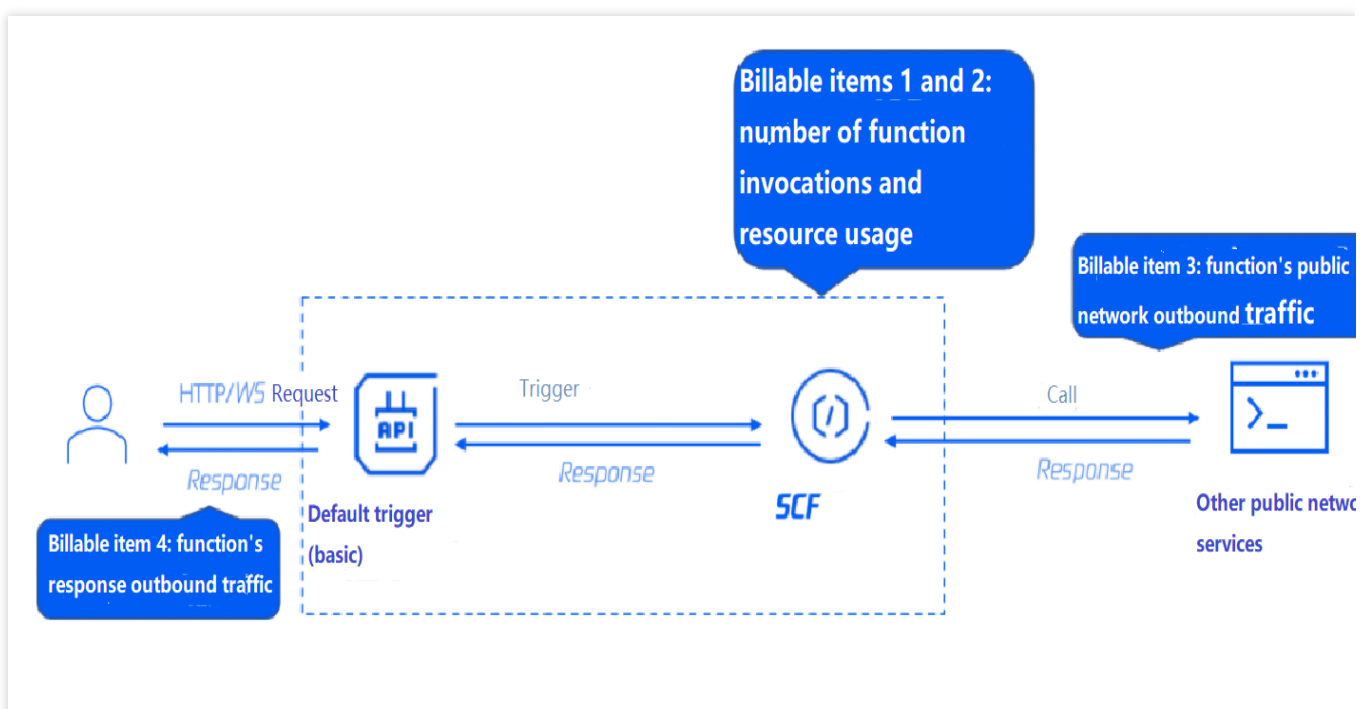
# HTTP-Triggered Function Billing

Last updated : 2024-12-02 20:07:18

Two trigger creation methods are provided for HTTP-triggered functions: default creation and custom creation, which have different billing logic.

## Default Creation

If you select "default creation", SCF will automatically create a basic API Gateway trigger, which provides only an access URL invisible in the console. In this case, an HTTP-triggered function is billed as follows:



**Trigger side:** Invocation is not billed, and outbound traffic is billed on the function side.

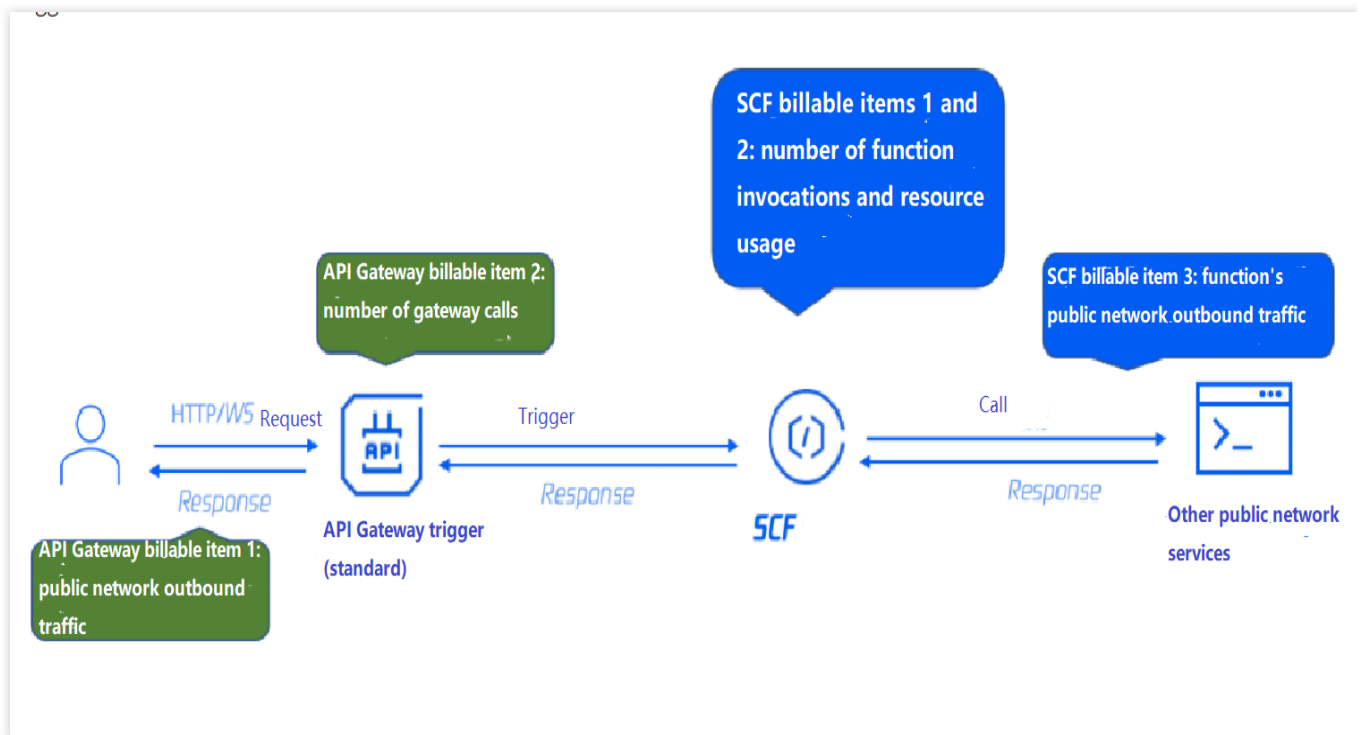
**Function side:** in addition to standard billable items, the outbound traffic is also billed.

### Note:

A basic API Gateway instance will be created in default creation mode. You can upgrade it to the standard edition in the SCF console. After upgrade, you can use all gateway capabilities, which will be billed in the standard API Gateway billing method. The upgrade is irreversible.

## Custom creation

If you select "custom creation", you need to select a trigger type in the SCF console and bind a created service. In this case, the current billing method applies, where the function and trigger are billed according to their own billing rules respectively. Taking a standard API Gateway trigger as an example, an HTTP-triggered function is billed as follows:



**Trigger side:** the trigger is billed according to the billing rules of the product itself.

**Function side:** the function is billed by standard billable items (invocations, resource usage, and outbound traffic), and the response traffic isn't billed on the function side.

## Trigger Capability Comparison

Function	Default Creation (Basic API Gateway)	Custom Creation (Standard API Gateway)
Default domain name	Supported	Supported
Binding to custom domain name	Manual binding	Management in API Gateway console
Request method configuration	Supported	Supported
Release environment configuration	Supported	Supported
Authentication method configuration	Not supported	Supported
Visibility in API	Invisible	Visible

Gateway console		
Advanced API Gateway capabilities (such as plugin and dedicated instance)	Not supported	Supported
Billing method	Gateway calls are not billed.	It is billed by standard API Gateway billable items.
Type conversion	The gateway can be upgraded to a standard API Gateway instance. After upgrade, you can use all gateway capabilities and billed by standard API Gateway billable items.	The gateway edition cannot be changed. A standard API Gateway instance cannot be rolled back to a basic API Gateway instance in default creation.
Backend timeout period	Fixed at 15s	Configurable



# Deploying Web Function on Command Line

Last updated : 2024-12-02 20:07:18

## Overview

Web function is a new function capability in SCF. Compared with event function that has limits on the event format, web function focuses on optimization of web service scenarios and can directly send HTTP requests to URLs to trigger function execution. For more information, please see [Function Overview](#).

The Serverless Framework SCF component now supports deploying web functions; therefore, you can use it to quickly create and deploy web functions.

## Directions

1. Run the following command to initialize the serverless web function template.

```
sls init http-demo
```

2. Enter the demo project and view the directory structure as shown below:

```
. http-demo
├─ serverless.yml # Configuration file
├─ package.json # Dependency file
├─ scf_bootstrap # Project bootstrap file
└─ index.js # Service function
```

Here, `scf_bootstrap` is the project bootstrap file. For the specific writing rules, please see [Bootstrap File Description](#).

3. Open `serverless.yml` to view the configuration information.

You only need to add the `type` parameter in `yml` to specify the function type and deploy the web function.

### Note:

For web functions, there is no need to specify the entry function.

If the `type` parameter is not entered, the function will be an event function by default.

If there is no `scf_bootstrap` bootstrap file in the local code, you can specify the `entryFile` parameter in `yml` to specify the entry function, and the component will generate a default `scf_bootstrap` file for you to complete the deployment based on the runtime language. After the deployment is completed, you need to modify the content of the `scf_bootstrap` file in the [SCF console](#) according to the actual needs of your project.

Below is a sample `yml` file:

```
component: scf
name: http
inputs:
  src:
    src: ./
    exclude:
      - .env
  # Specify web type as the function type
  type: web
  name: web-function
  region: ap-guangzhou
  runtime: Nodejs12.16
  # For Node.js, you can enable automatic dependency installation
  installDependency: true
  events:
    - apigw:
        parameters:
          protocols:
            - http
            - https
          environment: release
          endpoints:
            - path: /
              method: ANY
```

4. In the root directory, run `sls deploy` to complete the service deployment. Below is a sample:

```
$ sls deploy
serverless ⚡components
Action: "deploy" - Stage: "dev" - App: "http" - Name: "http"
type:          web
functionName:  web-function
description:   This is a function in http application
namespace:    default
runtime:       Nodejs12.16
handler:
memorySize:   128
lastVersion:  $LATEST
traffic:      1
triggers:
-
NeedCreate:   true
created:      true
serviceId:    service-xxxxxxx
serviceName:  serverless
```

```
subDomain:    service-xxxxxxx.cd.apigw.tencentcs.com
protocols:    http&https
environment:  release
apiList:
  -
    path:      /
    method:    ANY
    apiName:    index
    created:    true
    authType:   NONE
    businessType: NORMAL
    isBase64Encoded: false
    apiId:      api-xxxxxxx
    internalDomain:
    url:        https://service-xxxx.cd.apigw.tencentcs.com/release/
18s > http > executed successfully
```

## Relevant Commands

### Viewing access log

Similar to event function, you can directly run the `sls log` command to view the latest 10 logs of the deployed function.

Below is a sample:

```
$ sls log
serverless ⚡components
Action: "log" - Stage: "dev" - App: "http" - Name: "http"
-
  requestId:    xxxxxx
  retryNum:     0
  startTime:    1624262955432
  memoryUsage: 0.00
  duration:     0
  message:
    ""
    ""
-
  requestId: xxxxxx
  retryNum:   0
  startTime:  1624262955432
  memoryUsage: 0.00
  duration:   0
  message:
    ""
```

```
"""
```

## Testing service

Scheme 1: directly open the output path URL in a browser, and if it can be accessed normally, the function is successfully created, as shown below:



Scheme 2: use other HTTP testing tools such as CURL and Postman to test the web function you have successfully created. Below is a sample test with CURL:

```
curl https://service-xxx.cd.apigw.tencentcs.com/release/
```

## Deleting service

Run the following command to remove your deployed cloud resources.

```
sls remove
```

# WebSocket Protocol Support

Last updated : 2024-12-02 20:07:18

Currently, an HTTP-triggered function allows you to connect the client to the server where it runs over the native WebSocket protocol.

## How it Works

### Starting service

You can use a bootstrap file to start the WebSocket server in the runtime environment of the HTTP-triggered function configured with support for the WebSocket protocol and listen on the **specified port (9000)** to wait for client connections.

In addition, for the API Gateway trigger, you need to configure WebSocket (WS) or WebSocket Secure (WSS) as the frontend protocol and configure the currently specified WebSocket-enabled HTTP-triggered function as the backend. The WebSocket path provided by API Gateway can be used for client connection.

### Establishing WebSocket connection

The WebSocket client uses the WebSocket link provided by the API Gateway trigger to initiate a WebSocket connection. API Gateway and SCF will pass through the connection to the service process in the runtime environment. The negotiation and communication for connection establishment are both processed on the server through code. After the connection is established, the client and server normally communicate over the WebSocket protocol.

### WebSocket connection lifecycle

If an HTTP-triggered function supports WebSocket, the lifecycle of a WebSocket connection is the same as an invocation request of the function, the WebSocket connection establishment process equals request initiation, and disconnection equals request end.

Plus, function instances and connections are in one-to-one correspondence; that is, one instance only processes one WebSocket connection at a time. When more client connection requests are initiated, the same number of instances will be started for processing.

When a WebSocket connection request is initiated, a function instance will be started and receive the connection request.

After the WebSocket connection is established, the instance will run continuously and receive and process the upstream data from the client based on the actual business conditions. Or, the server actively pushes the downstream data.

After the WebSocket connection is closed, the instance will stop running.

## Disconnection

A WebSocket connection will be closed in the following cases, and the current request execution will also end, as the request lifecycle is the same as the connection lifecycle:

Disconnection Conditions	Function Status	Function Status Code
The client or server initiates an operation of ending or closing the connection, and the end status code is 1000, 1010 (sent by client), or 1011 (sent by server).	The function ends normally after execution, and the execution status is "success".	200
The client or server initiates an operation of ending or closing the connection, and the end status code is not 1000, 1010, or 1011.	The function ends exceptionally, and the execution status is "failure".	439 (closed by server) or 456 (closed by client)
The connection is closed by SCF when there are no upstream or downstream messages sent over the WebSocket connection within the configured idle timeout period.	The function ends exceptionally, and the execution status is "failure".	455
The connection is closed by SCF as it is used continuously after being established and the function execution duration reaches the maximum value.	The function ends exceptionally, and the execution status is "failure".	433

For more information on WebSocket status codes for connection end, see [WebSocket Status Codes](#).

For more information on function status codes, see [Function Status Code](#).

## Usage Limits

Use of WebSocket has the following limits:

Idle timeout period: 10–7200 seconds. The execution timeout period of a function must be greater than or equal to the idle timeout period.

Maximum request or response packet size: 256 KB. You can [submit a ticket](#) to increase the quota limit.

Maximum request size per connection: 128 KB/s. You can [submit a ticket](#) to increase the quota limit.

Maximum request QPS per connection: 10. You can [submit a ticket](#) to increase the quota limit.

## Directions

### Creating a function

1. Log in to the [Serverless console](#).
2. Click **Create** to create a function. You can select **Custom > HTTP-Triggered Function > Advanced Configuration** to view the supported protocols as shown below:

The screenshot shows the 'Create Function' interface in the Tencent Cloud Serverless console. The 'Create Method' section has 'Custom' selected. Under 'Basic Configurations', 'Function Type' is set to 'HTTP-triggered Function'. The 'Advanced Configuration' section is highlighted with a red box, displaying a note: 'Function invocation logs are published to the SCF-specific topic of CLS, which will use the free tier of CLS. See [CLS Billing Details](#)'.

3. Select **WebSocket Support** and configure **WebSocket Idle Timeout** to support the WebSocket as shown below:

### Concurrency Configurations

Reserved Quota  =  Concurrent Executions X  Configured memory ⓘ

### Async Invocation

Dead Letter Queue ☐ Enable ⓘ

#### Supported Protocols

WebSocket support ☐ Enable

WebSocket Idle Timeout  s  
Range: 1 - 600 seconds

### Tag

Tag ☐ Enable

4. After **WebSocket Support** is selected, the supported protocols of API Gateway are also automatically switched to **WS&WSS**, and the link provided by the created API gateway will also be a WebSocket address as shown below:



### Trigger Configurations

Triggered Version	Default Traffic
Trigger Method	API Gateway Trigger
API Service Type	<input checked="" type="radio"/> Create API Service <input type="radio"/> Use Existing API Service
API Service	SCF_API_SERVICE
Supported Protocols	WS&WSS
Request method	GET
Publishing Environment	Publish
Authentication Method	No authentication

1. HTTP-triggered functions only support API gateway triggers, which can directly receive HTTP request.  
2. You can configure a custom domain name after creating the function. [Learn More](#)

#### Note:

After creation, the support for WebSocket protocol cannot be canceled, but the idle timeout period can be changed as needed.

#### Sample code

You can use the following demos to create a function and try out WebSocket:

[Demo for Python](#): Use the `websockets` library to implement the WebSocket server.

[Demo for Node.js](#): Use the `ws` library to implement the WebSocket server.

# HTTP-Triggered Function Request Concurrency Management

Last updated : 2024-12-02 20:07:18

## Request Concurrency Overview

### Single concurrent request

By default, when a function is invoked, SCF will assign a concurrent instance to process the request or event. After the function code is executed and its response is returned, the instance will process other requests. If all instances are running when a request arrives, SCF will assign a new concurrent instance. One concurrent instance processes only one event at any time so as to ensure the processing efficiency and stability of each event.

### Multiple concurrent requests

In most cases, one single concurrent request is the recommended mode, and you don't need to consider how to solve the typical concurrency problems for processing multiple concurrent requests, such as thread security, blocked invocation, and exception handling, when writing code.

However, in web applications, typical business scenarios are I/O-intensive, and access to downstream services such as database or other system APIs in the function takes a long time to wait for the downstream services to respond. Generally, such waits are `await` and don't consume the CPU resources. In this case, if the multiple concurrent requests feature is enabled, one instance can process multiple requests to better utilize the CPU resources of the single instance.

Currently, HTTP-triggered functions allow you to [enable multiple concurrent requests](#). You can enable and configure this feature as needed, which supports two modes: **custom concurrency** and **dynamic concurrency**.

#### Custom concurrency

After it is enabled, if there are multiple concurrent requests, requests within the specified maximum number of concurrent requests will be scheduled to the same function instance for execution. If there are more concurrent requests, the function instance's CPU and memory usage will increase. We recommend you use stress tests to evaluate appropriate configurations, so as to avoid function execution exceptions. Currently, 2–100 concurrent requests are supported.

#### Dynamic concurrency

After it's enabled, requests are scheduled to the same function instance to the maximum extent. This feature will be released in the future.

## Directions

## Enabling multiple concurrent requests

1. Log in to the SCF console and select **Function Service** on the left sidebar.
2. On the **Function Service** list page, select the name of the target HTTP-triggered function.
3. On the **Function Management** page, select **Function Configuration**.
4. On the **Function Configuration** page, click **Edit**.
5. In **Multiple Concurrent Requests**, select **Enable** to enable the multiple concurrent requests mode. In the input box displayed below **Custom concurrency**, enter the desired maximum number of concurrent requests.
6. Click **Save**.

## Strengths of Request Concurrency

In I/O-intensive scenarios such as WebSocket persistent connection, the billable execution duration can be shortened to reduce the costs.

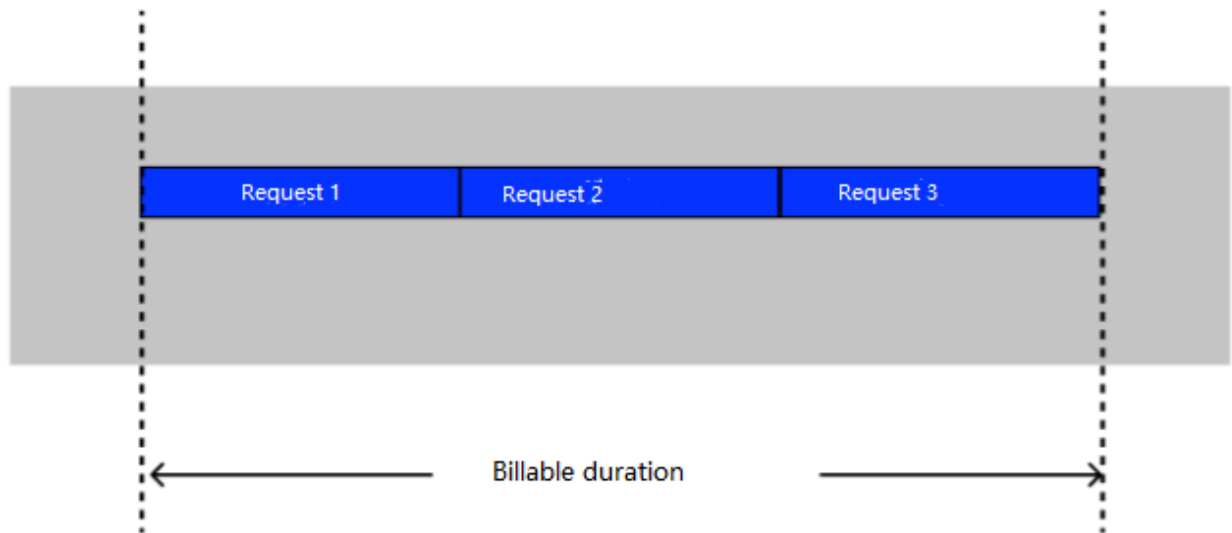
Multiple concurrent requests in the same instance can reuse the database connection pool to relieve the pressure on the downstream server.

When there are intensive concurrent requests, they only require one instance for processing, with no need to start multiple instances. This reduces the cold instance starts and response latency.

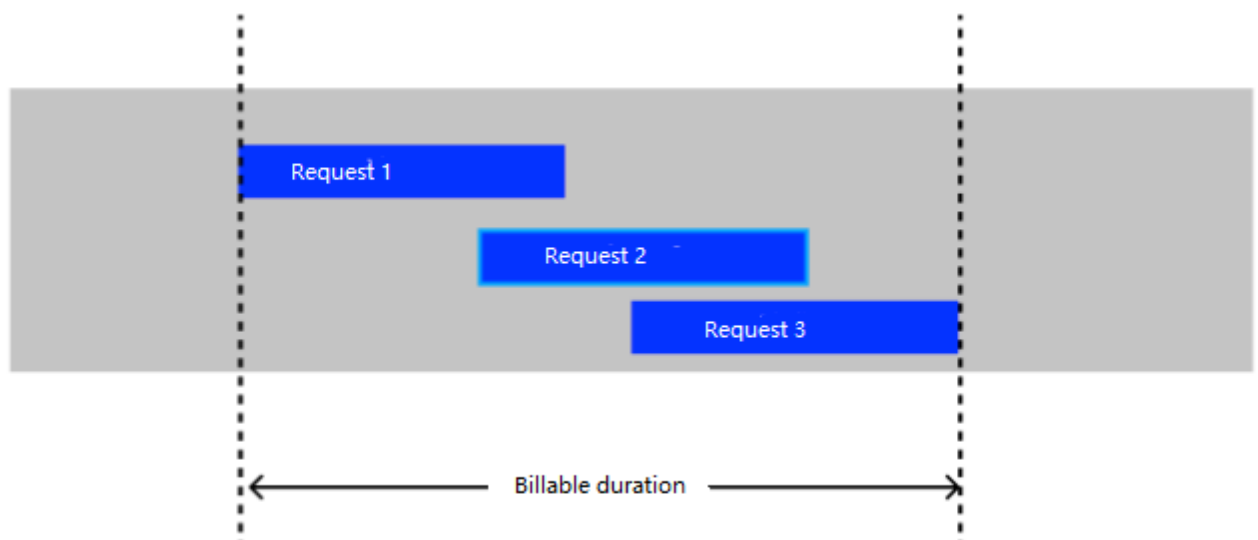
## Notes

### Cost

If the multiple concurrent requests feature is not enabled, a single function instance will process only one request at a time and will process the next request only after processing the current request. The memory billing duration is the sum of the execution duration of all requests as shown below:



After the multiple concurrent requests feature is enabled, a single function instance will process multiple concurrent requests. If another request is received while a request is being processed, there will be a period during which the two requests are processed at the same time. In this case, the period of concurrent execution will be billed only once as shown below:



Other billable items remain unchanged. For more information, see [Billing Overview](#).

## Log

After the multiple concurrent requests feature is enabled, as multiple requests are processed concurrently, when the logs generated by each request are reported, the logs and `RequestID` values may not be in one-to-one correspondence. In this case, you need to correctly set a logger in the code to print `RequestID` values in logs.

`RequestID` can be obtained from the `X-Scf-Request-Id` field in the common request header received by the HTTP-triggered function.

## Limit exceeding error

### OOM

Multiple concurrent requests increase the probability of OOM. If OOM occurs, the instance will restart. In this case, the `abort` error (error code: 434 MemoryLimitReached) will be reported for multiple ongoing requests in the instance. Before setting the maximum number of concurrent requests, you need to perform stress tests on the function to determine a safe number, so as to avoid the impact of OOM.

### Timeout

If a request fails to be executed within the configured execution timeout period, it will be stopped, and error code `433 TimeLimitReached` will be returned to the client. Other ongoing requests in the instance won't be affected.

## Monitoring

After the multiple concurrent requests feature is enabled, the **Number of Concurrent Requests** panel will be displayed on the monitoring page, where you can directly view the concurrent request statistics for the specified time period.

# Log Management

## Log Search Guide

Last updated : 2024-12-02 20:07:18

### Overview

SCF upgraded its log service on January 29, 2021 and was fully connected to Tencent Cloud CLS. The functions created before are gradually migrated by regions. For more information, please see [SCF Log Service Change Notification](#).

SCF provides the advanced search feature for functions created after January 29, 2021 and the migrated functions. The CLS search and analysis page is embedded in the [SCF console](#). You can search for logs by using keyword or using query syntax to combine keyword.

#### Note:

If your function was created before January 29, 2021 and has not been migrated, but you need to use more log analysis features, please see [Log Delivery Configuration \(Legacy\)](#) to deliver function invocation logs to Cloud Log Service (CLS).

### Directions

You can use the advanced search feature in the following steps:

1. Log in to the [SCF console](#) and select **Function Service** on the left sidebar.
2. On the **Function Service** list page, select the name of the function for which to search for logs to enter the **Function Management** page.
3. On the **Function Management** page, select **Log Query > Advanced Retrieval**.
4. On the **Advanced Retrieval** page, you can search for logs by using keyword or using query syntax to combine keyword. **For more information on syntax rules, please see [Syntax and Rules](#).**
5. After configuring the search criteria, you can click **Search and Analysis** on the right, and the results will be returned as shown below:

Function Management

Version: \$LATESTOperat

Function configurationFunction CodesLayer ManagementMonitoring InformationLog Query

Invocation LogsAdvanced Retrieval

Quick SearchIndex ConfigurationPreferences

1SCF\_FunctionName:helloworld-1616063925ANDSCF\_Qualifier:\$LATESTSave asLast 15 MinutesSearch and Analysis

Log Quantity 9Mar 26, 2021 @ 11:17:16.026 - Mar 26, 2021 @ 11:32:16.026

105

11:17:0011:18:3011:20:0011:21:3011:23:0011:24:3011:26:0011:27:3011:29:0011:30:3011:32:00

Raw DataChart Analysis

Search

Log Time ↓Full Log

Showed FieldFull Log

03-26 11:32:00.457

\_\_SOURCE\_\_:\_FILENAME\_\_:\_PKG\_LOGID\_\_:\_SCF\_FunctionName:helloworld-1616063925SCF\_Namespace:defaultSCF\_StartTime: 1616729518514SCF\_RequestId: 0ba76dce-ddfa-4f25-9aa7-7cb91806e0dbSCF\_Duration:4SCF\_Alias:\_SCF\_Qualifier:\$LATESTSCF\_LogTime: 1616729520457881475SCF\_RetryNum: 0SCF\_MemUsage: 5

©2013-2025 Tencent Cloud International Pte. Ltd.

Page 67 of 251

# Log Delivery Configuration

Last updated : 2024-12-02 20:07:18

## Note:

If your function was created before January 29, 2021 and has not been migrated, but you want to use more log analysis features, deliver function invocation logs to CLS as instructed in [Log Delivery Configuration \(Legacy\)](#). SCF was fully connected to [Tencent Cloud CLS](#) starting from January 29, 2021. After then, the invocation logs of newly created functions will be delivered to CLS, and logs can be output in real time. The existing functions are gradually migrated by regions. For more information, see [SCF Log Service Change Notification](#).

This document describes the two log delivery methods of [default delivery](#) and [custom delivery](#) provided by SCF and how to configure them.

## Permission Description

To view the logs normally, ensure that the sub-account at least has the read-only permission of CLS

`QcloudCLSReadOnlyAccess` . For how the root account grant permissions for the sub-account, see [Authorization Management](#).

## Restrictions

Delivering function invocation logs to CLS has the following limits:

The maximum amount of logs printed within 5 seconds for each request is 1 MB.

The maximum number of logs printed within 5 seconds for each request is 5000.

The maximum length of each log is 8 KB, and excessive parts will be discarded.

## Directions

### Default delivery

When creating a function, if you don't specify the destination topic for log delivery, the default log delivery capability will be used. For default log delivery, SCF will activate the CLS service for you and deliver the function invocation logs to the log topic under the SCF-specific logset. The SCF-specific logset and log topic are prefixed with `SCF_logset` and `SCF_logtopic` respectively, and will be created automatically if they do not exist. Function invocation logs will be retained for 7 days by default, and you can view and manage them on the [CLS console](#).

## Note:

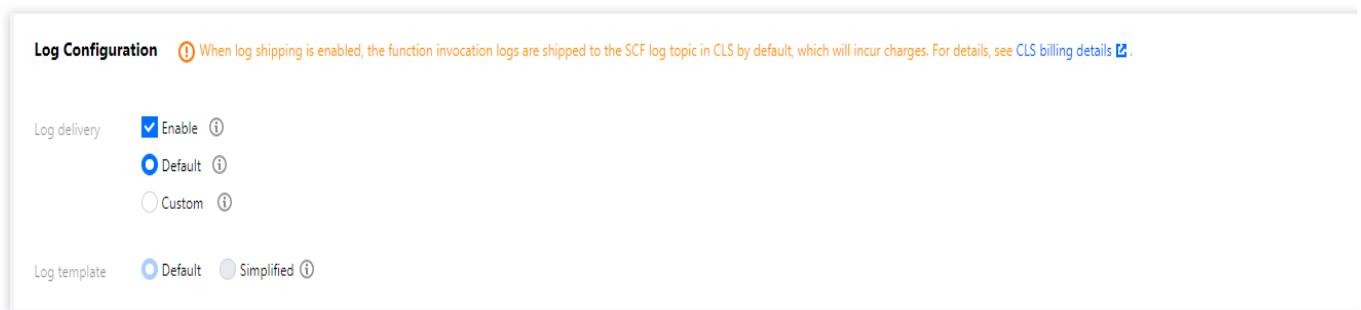


CLS is billed separately, and the SCF-specific log topic will consume the free tier of CLS. For more information, see [Billing Overview](#).

To ensure the proper display of logs in the SCF console, SCF-specific log topics do not support modifying the log index configuration. To customize the index configuration, configure the function log topic as instructed in **Custom delivery** below.

### Configuring CLS

1. Log in to the SCF console and select **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. In **Log configuration**, select **Default** as shown below:



**Log Configuration** ⓘ When log shipping is enabled, the function invocation logs are shipped to the SCF log topic in CLS by default, which will incur charges. For details, see [CLS billing details](#).

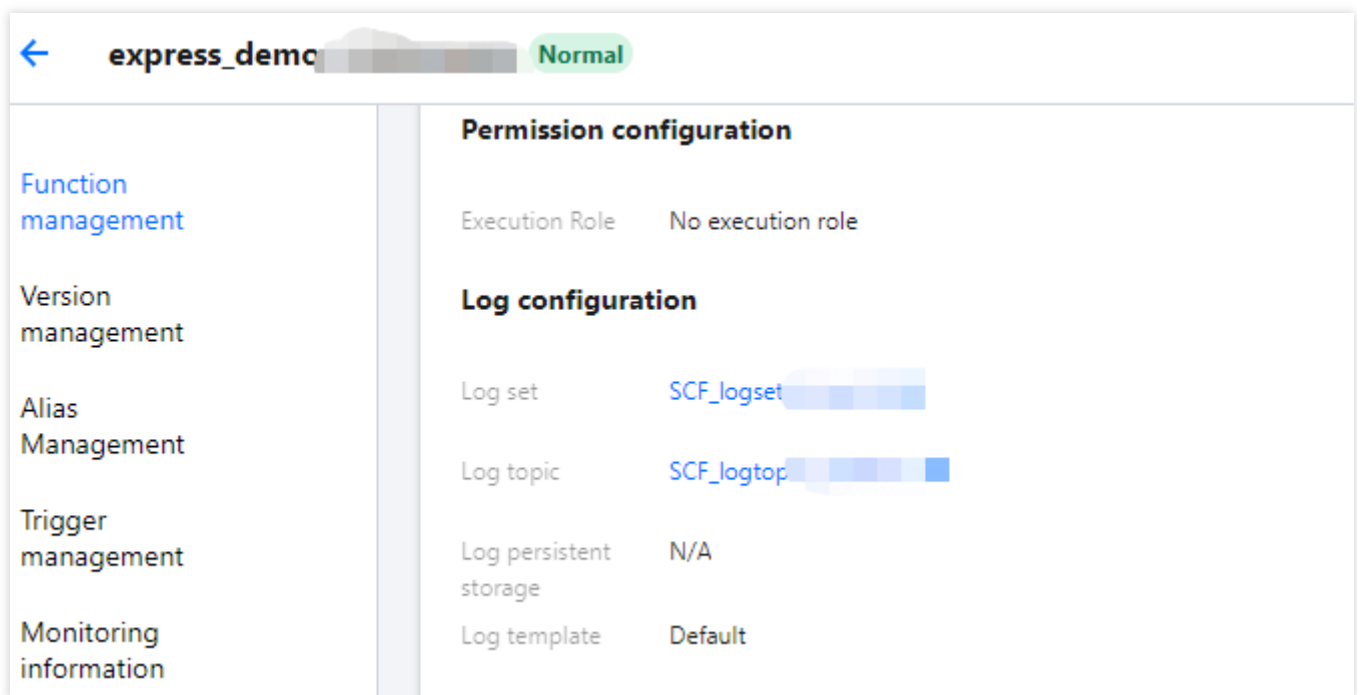
Log delivery ☒ Enable ⓘ

☒ Default ⓘ

☐ Custom ⓘ

Log template ☒ Default ☐ Simplified ⓘ

4. Click **Complete**. You can view the log configuration in **Function management** > **Function configuration** as shown below:



← express\_demo Normal

<b>Function management</b>	<b>Permission configuration</b>
Version management	Execution Role No execution role
Alias Management	<b>Log configuration</b>
Trigger management	Log set SCF_logset
Monitoring information	Log topic SCF_logtop
	Log persistent storage N/A
	Log template Default

### Viewing and managing logs

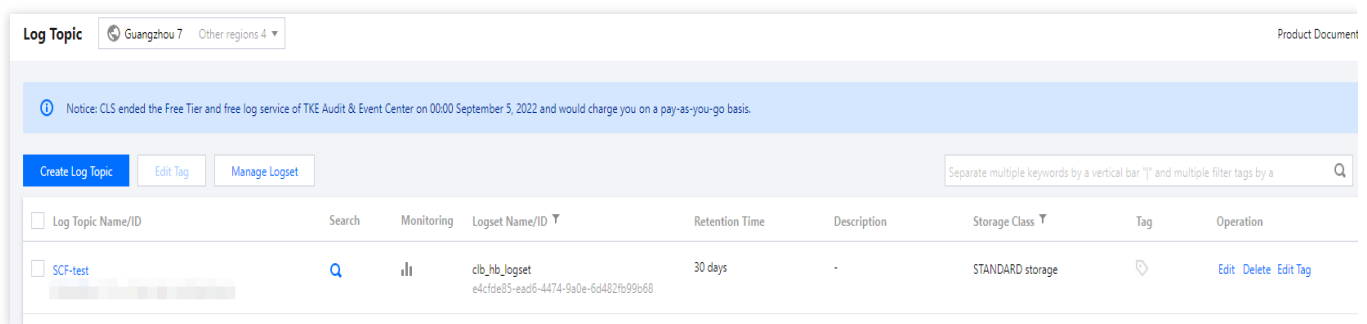
You can click the logset ID in **Log Configuration** in **Function Configuration** to enter the [CLS console](#) to view and manage logs. The SCF-specific logset is marked with the word `SCF` in the CLS console. If you need to persistently store, deliver, or consume logs or monitor and set alarms on log content, you can complete the configuration in the CLS console.

## Custom delivery

When creating a function, if you need to specify the destination log topic to deliver function invocation logs, you can use the custom log delivery capability. Before using this capability, you should make sure that the [CLS](#) service has been activated.

## Creating logset and log topic

Log in to the [CLS console](#) and [create a log topic](#). This document uses the creation of the `SCF-test` log topic in Guangzhou as an example as shown below:

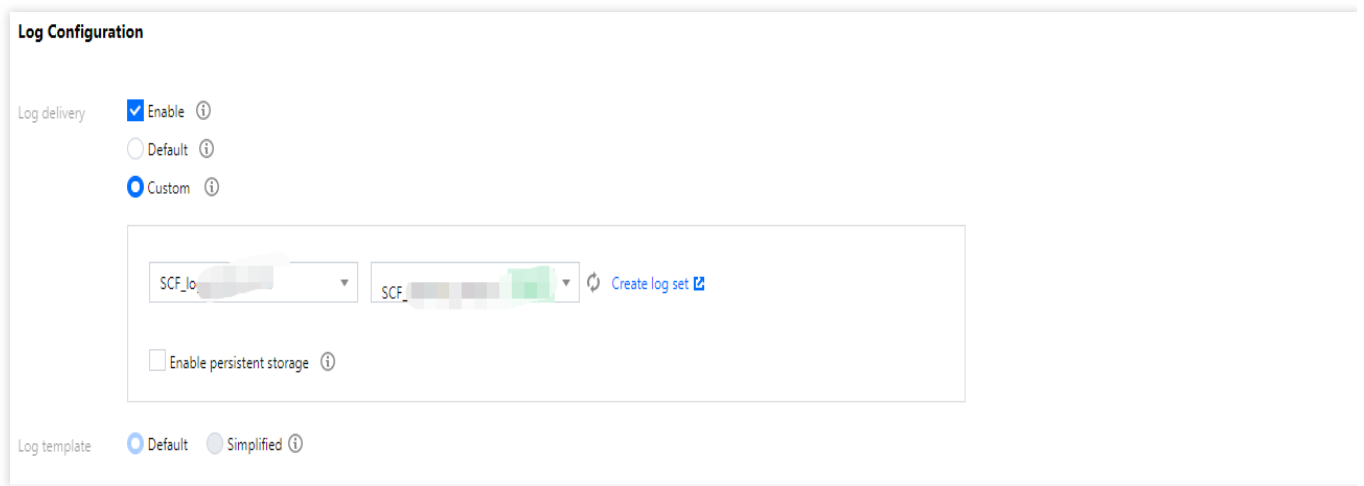


### Note:

For the logset region, select the region where the SCF service is located. Cross-region log push is not supported currently.

## Configuring CLS

1. Log in to the SCF console and select **Functions** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. In **Log configuration**, select **Custom** and select the log topic already created for this function. This document uses `SCF-test` as an example as shown below:



**Log Configuration**

Log delivery ☒ Enable ⓘ  
☐ Default ⓘ  
☒ Custom ⓘ

SCF\_...

SCF\_...

↻ Create log set ⓘ

☐ Enable persistent storage ⓘ

Log template ☒ Default ☐ Simplified ⓘ

4. Click **Confirm**.

## Configuring indexes

Log search depends on the index configuration of the log topic. SCF will automatically complete the index configuration when you create a function. If the index is abnormal and logs cannot be viewed properly, configure the index in the following steps:

1. Log in to the SCF console and select **Functions** on the left sidebar.
2. On the **Function Service** list page, select the name of the function whose index is abnormal to enter the **Function Management** page.
3. On the **Log Query** tab, select **Index Configuration** in **Advanced Retrieval** as shown below:



Invocation logs **Advanced retrieval**

**Index Configuration** Preferences Create Data Processing Task

1 SCF\_FunctionName:"express\_d..." AND SCF\_Qualifier:"\$LATEST" AND SCF\_Namespace:"default"

☆ Last 15 Minutes ▾

Search and

+ Add Filter Condition

4. On the **Index configuration** page, enable **Index status** and **Key-value index** as shown below:

### Index Configuration

Import Configuration Rules

Index Status ☒

Full-Text Index ☒ ☐ Case sensitive

Full-Text Delimiter

Allow Chinese Characters ☐

Key-Value Index ☒ ☐ Case sensitive

[Auto Configure](#)

Field Name	Field Type	Delimiter	Allow Chi...	Enable St...	
SCF_Alias	text	Enter delimit	<input type="checkbox"/>	<input checked="" type="checkbox"/>	✕
SCF_Duration	long	None	<input type="checkbox"/>	<input checked="" type="checkbox"/>	✕
SCF_FunctionName	text	Enter delimit	<input type="checkbox"/>	<input checked="" type="checkbox"/>	✕

This configuration method is only valid for scenarios where there are function invocation logs in the log topic; otherwise, manually configure the **key-value index** by referring to the table below.

Field Name	Field Type	Description
SCF_FunctionName	text	Function name
SCF_Namespace	text	Function namespace
SCF_StartTime	long	Invocation start time
SCF_LogTime	long	Log generation time
SCF_RequestId	text	Request ID
SCF_Duration	long	Function execution duration

SCF_Alias	text	Alias
SCF_Qualifier	text	Version
SCF_MemUsage	double	Function runtime memory
SCF_Level	text	Log4J log level. Default value: INFO
SCF_Message	text	Log content
SCF_Type	text	Log type. Platform: Platform log; Custom: User log
SCF_StatusCode	long	<a href="#">Status code</a> of function execution
SCF_RetryNum	long	Number of retries

To ensure the display effect of the logs in the SCF console, toggle on **Enable Statistics** for the field in the key-value index configuration.

5. After configuring the index, click **OK**.

# Log Delivery Configuration (Legacy)

Last updated : 2024-12-02 20:07:18

## Note:

SCF was fully connected to [CLS](#) starting from January 29, 2021. After then, the invocation logs of newly created functions will be delivered to CLS by default, and logs can be output in real time. If your function was created before January 29, 2021, but you need to search for and deliver logs, please refer to this document to use this feature.

## Overview

When SCF is used for function computation, a large number of function execution logs will be generated. If you need to persistently store, deliver, or consume logs and monitor and set alarms on log content, you can deliver logs to the Tencent Cloud CLS, as shown below:



## Prerequisites

Before using the SCF real-time log service, you need to activate [CLS](#) first.

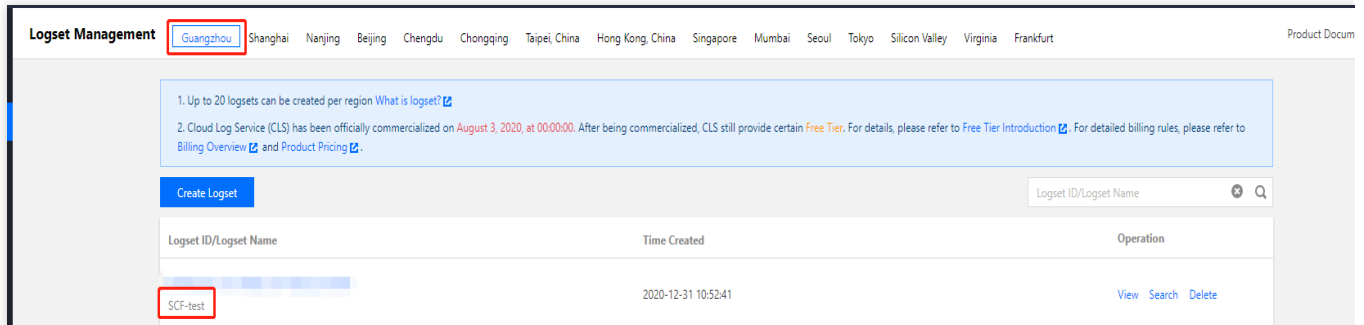
## Directions

### Creating logset and log topic

Log in to the [CLS console](#) and [create a logset and log topic](#). This document uses the creation of the `SCF-test` logset and log topic in Guangzhou as an example, as shown below:

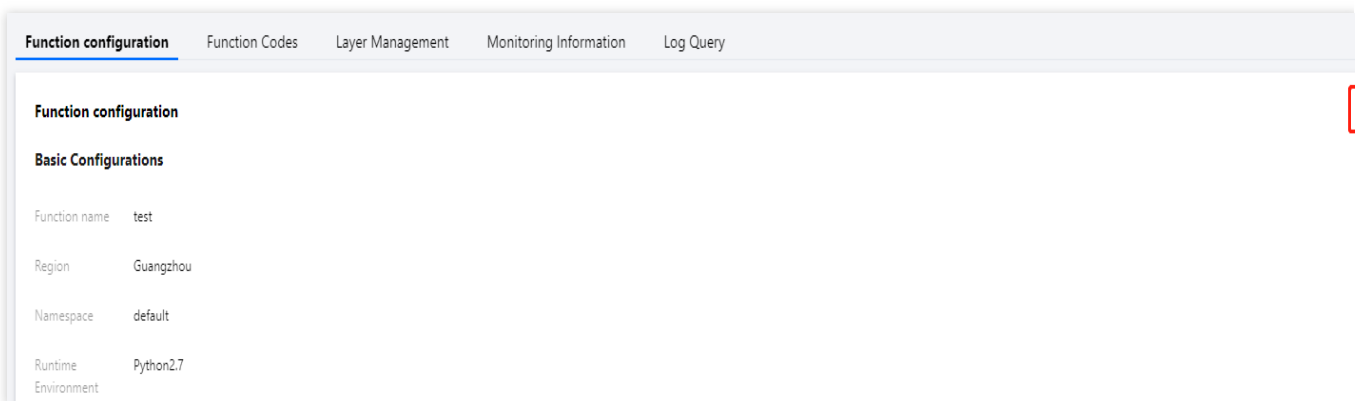
## Note:

For the logset region, please select the region where the SCF service is located. Cross-region log push is not supported currently.

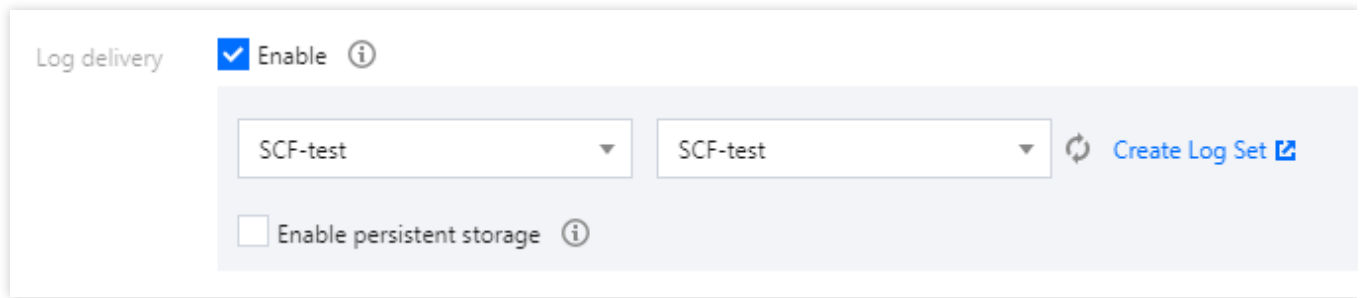


## Configuring CLS

1. Log in to the SCF console and select **Function Service** on the left sidebar.
2. Select the SCF region and namespace at the top of the page and click the function name in the list for which to collect logs in real time.
3. On the **Function Configuration** page, click **Edit** in the top-right corner as shown below:



4. In **Log delivery**, click **Enable** and select the logset and log topic already created for this function. This document uses `SCF-test` as an example, as shown below:



Log delivery ☒ Enable ⓘ

SCF-test SCF-test ↻ Create Log Set ↗

☐ Enable persistent storage ⓘ

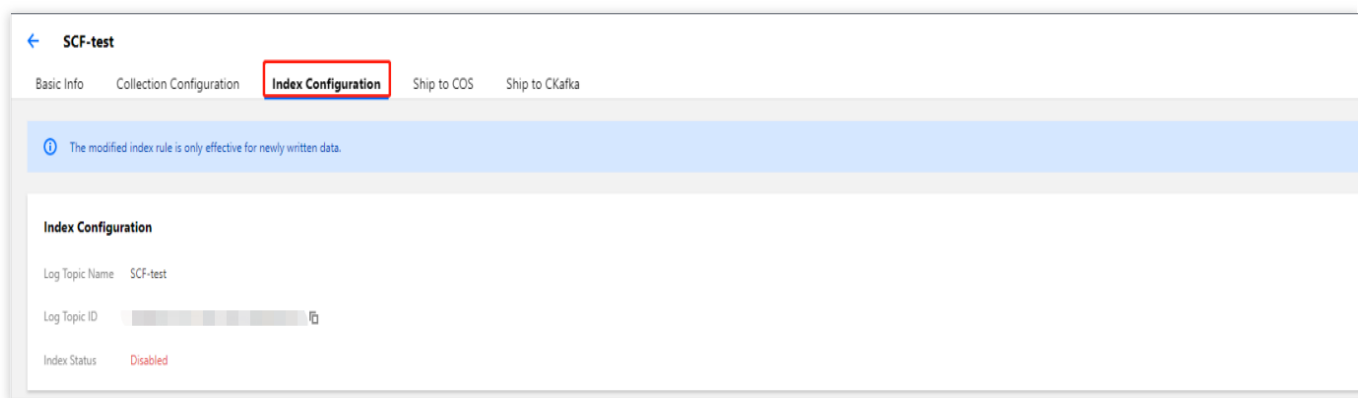
5. Click **Save** to connect to the CLS platform.

## Enabling index

### Note:

Log search depends on the index configuration of the log topic. After the function is associated with the log topic, SCF will automatically complete the index configuration for the log topic. If the index is exceptional and logs cannot be pulled properly, please adjust the index configuration in the following steps:

1. Log in to the CLS console, and select **Logset** on the left sidebar.
2. Click the ID of a created logset to enter the **Basic Info** page.
3. Select **Manage** on the right of the log topic row to enter the **Basic Info** page of the log topic.
4. On the **Basic Info** page of the log topic, click **Index Configuration** as shown below:



← SCF-test

Basic Info Collection Configuration **Index Configuration** Ship to COS Ship to CKafka

ⓘ The modified index rule is only effective for newly written data.

**Index Configuration**

Log Topic Name SCF-test

Log Topic ID [redacted] ⓘ

Index Status **Disabled**

5. Click **Edit** in the top-right corner, enable **Key-Value Index**, and add **Field Name** and **Field Type** according to the following table.

### Note:

For functions configured with CLS, to ensure the display effect of the logs in the SCF console, please toggle on **Enable Statistics** for the field in key-value index configuration, as shown below:



SCF-test

Basic Info Collection Configuration **Index Configuration** Ship to COS Ship to CKafka

The modified index rule is only effective for newly written data.

**Index Configuration**

Index Status ☒

Full-Text Index ☒ ☐ Case sensitive

Full-Text Delimiter

Key-Value Index ☒ ☐ Case sensitive

[Auto Configure](#)

Field Name	Field Type	Delimiter	Enable Statistics	Operation
SCF_FunctionName	text	None	<input checked="" type="checkbox"/>	Delete
SCF_Namespace	text	None	<input checked="" type="checkbox"/>	Delete

[Add](#)

Metadata Index (TAG) ☐

[OK](#) [Cancel](#)

Field Name	Field Type	Description
SCF_FunctionName	text	Function name
SCF_Namespace	text	Function namespace
SCF_StartTime	long	Invocation start time
SCF_LogTime	long	Log generation time
SCF_RequestId	text	Request ID
SCF_Duration	long	Function execution duration
SCF_Alias	text	Alias
SCF_Qualifier	text	Version
SCF_MemUsage	double	Function runtime memory
SCF_Level	text	Log4J log level. Default value: INFO
SCF_Message	text	Log content
SCF_Type	text	Log type. Platform: platform log, Custom: user log
SCF_StatusCode	long	<a href="#">Status code</a> of function execution
SCF_RetryNum	long	Number of retries

For more features, such as real-time log search, log shipping and consumption, see the [CLS documents](#), and log in to the [CLS console](#) to use them.

## Real-time search sample

### Note:

Before using the real-time search feature, please make sure that your SCF log service has been connected to the CLS platform and the index has been enabled for the log topic to be searched.

1. Log in to the CLS console, and select [Log Search](#) on the left sidebar.
2. On the **Search and Analysis** page, select the desired log topic and time and enter the search syntax in the input box. This document uses `START` as an example.

The search syntax supports keyword search, fuzzy search, range search, and other search methods. For more information, please see [Legacy CLS Search Syntax](#).

3. Click **Search and Analysis** to view real-time log information.

# Concurrency Management

## Concurrency Overview

Last updated : 2025-03-28 10:24:33

Concurrency refers to the number of requests that can be processed by a function concurrently at a moment. If it can be sustained by other services of your business, you can increase the function concurrency from several to tens of thousands with simple configuration.

## How Concurrency Works

When a function is invoked, SCF will assign a concurrent instance to process the request or event. After the function code is executed and its response is returned, the instance will process other requests. If all instances are running when a request arrives, SCF will assign a new concurrent instance.

SCF follows the execution logic that one concurrent instance processes only one event at any time so as to ensure the processing efficiency and stability of each event.

### Concurrent processing for async invocations

Async events enter a queue on SCF, where they will be processed in a FIFO manner. The system will select an appropriate concurrency processing method based on the conditions such as queue length and current number of concurrent instances of the function to pull sufficient concurrent instances and process the events in sequence.

If an async invocation fails, SCF will retry according to certain rules. For more information, see [Error Types and Retry Policies](#).

### Concurrent processing for sync invocations

When sync events arrive, SCF checks for idle concurrent instances. If yes, the events are immediately sent to idle instances; otherwise, new concurrent instances are started to process them.

When a sync invocation fails, you need to retry on your own.

### Concurrency calculation

SCF concurrency refers to the number of requests or invocations processed by the function code at a time, which can be estimated according to the following formula:

Concurrency = request rate \* function execution duration = QPS \* average time per request

You can view the average time per request in **execution duration** in the monitoring data.

For example, if the QPS of a business is 2,000, and the average time per request is 0.02 seconds, then the concurrency during function execution will be  $2000 * 0.02 = 40$ .

## Concurrent Instance Reuse and Repossession

After a concurrent instance processes a request event, it will not be repossessed immediately; instead, it will be retained for a certain period of time for reuse. During the retention duration, if there are new request events that need to be processed, the retained concurrent instance will be used first, so the events can be processed quickly with no need to start new concurrent instances.

After the retention duration elapses, if there are no requests that need to be processed by the instance, the SCF platform will repossess it. For low concurrency scenarios, no retention duration is set, and the platform will enable the smart repossession mechanism for resource repossession.

The concurrent instance retention duration is dynamically adjusted by SCF as needed; therefore, you cannot assume a certain retention duration when writing the function business code.

## Concurrency Scale-out

When a request arrives, but no concurrent instance for that version is available, a new concurrent instance is automatically launched and initialized for it. This is called elastic concurrency scale-out, and its speed limit is called **function burst**.

The **default upper limit of scale-out speed (function burst) per region under each account is 500 instances/minute**, that is, up to 500 new concurrent instances can be started in one minute for all functions in this region. If the limit is hit in one minute, no more new instances will be started until the minute elapses, during which an over-limit error (429 ResourceLimit) occurs if new scale-out requests are initiated. For more information, see [Function Status Code](#).

For example, the concurrency quota of an account in the Guangzhou region is 1,000 concurrent instances by default for a 128 MB function, and if many requests arrive, 500 concurrent instances can be started from 0 in the first minute. If there are still other requests to be processed, 500 more concurrent instances can be started to reach 1,000 instances in total in the second minute, until the number of concurrent instances is sufficient for the requests or reaches the upper limit.

Currently, the function burst of 500 instances/minute can meet the requirements in most business scenarios. If your business is limited by this scale-out speed, or you need to add namespace-level function burst management capabilities, you can select provisioned concurrency for prefetch or purchase a [function package](#) to increase the limit.

### Provisioned concurrency

Concurrent instances launched for elastic concurrency scale-out need to be initialized, which includes initialization of the runtime environment and the business code.

You can configure the **provisioned concurrency** to get concurrent instances ready in advance. **When the provisioned concurrency is configured, SCF will start the concurrent instances, and will not actively repossess the provisioned instances, so as to guarantee the number of concurrent instances.** If errors

such as code memory leak occur on a concurrent instance, it will be replaced by a new one. For more information, see [Provisioned Concurrency](#).

## Concurrency service level

### Concurrency scale-out limit

Limit on Concurrency Scale-out	Default Limit	Additional Quota Available for Application
Elastic concurrency scale-out speed limit (function burst)	500 concurrent instances/minute	Concurrency scale-out speed at the 10,000 concurrent instances/minute level is supported, which you can get by purchasing a function package.
Provisioned concurrency scale-out speed limit (provisioned burst)	100 concurrent instances/minute	The speed of starting provisioned concurrency can be automatically adjusted according to business conditions.

At the region level, the function burst is limited to 500 concurrent instances/minute by default. For example, if you need 50,000 concurrent instances, it will take  $50000/500 = 100$  minutes to complete the scale-out at the maximum function burst speed. If you need to increase function burst, you can directly purchase a [function package](#). SCF will adjust the provisioned burst based on your business, which is 100 concurrent instances/minute by default.

### Concurrent function quota

SCF provides concurrency management capabilities at the function granularity by default for you to flexibly control the concurrency of different functions. Each account has a limit on the quota of total concurrent functions in different regions as detailed below. If you want to increase the quotas or add concurrency quota management capabilities at the namespace granularity, you can directly purchase a [function package](#).

Function	Region	Default Quota	Additional Quota Available for Application
Concurrent function quota	Guangzhou, Shanghai, Beijing, Chengdu, and Hong Kong (China)	128,000 MB	At the one million MB level, which can be increased by purchasing a function package
	Singapore, Tokyo, Silicon Valley, Frankfurt, Shenzhen Finance, and Shanghai Finance	64,000 MB	

## Concurrency Management

SCF provides concurrency management capabilities at the function granularity. For more information, see [Concurrency Management System](#).

## Concurrent memory and concurrency

To help you manage concurrency more precisely, the SCF concurrency quota is calculated by memory; for example, a 256 MB concurrency quota represents one concurrent instance with 256 MB memory or two instances with 128 MB memory each.

## Reserved quota

What a reserved quota does:

The reserved quota is the upper limit of the concurrency quota for all versions of this function. The sum of the concurrency quotas of all versions cannot exceed this limit.

The concurrency quota is allocated **exclusively** to this function and will not be shared by other functions.

## Concurrency Monitoring

A concurrent instance is marked as running when it's processing requests. In SCF monitoring information, you can query the running concurrent instances of a function, a specific function version, or an alias. Because of the time difference between function execution and data collection, if a function has a large number of concurrent instances running in a very short time, the current monitoring data may not be completely accurate.

## Use Cases

By using reserved quota and provisioned concurrency together, you can flexibly allocate resources among multiple functions and warm up functions as needed.

### Shared quota

If nothing is configured, all functions share the account quota by default. When the requests to a function increase, it can make full use of the unused quota to prevent overrun errors.

### Guaranteed concurrency

For functions used for key business, a high request success rate is required. We can set up the reserved quota to allocate dedicated resources for the function, so as to guarantee the concurrency reliability and avoid overruns caused by concurrency preemption by multiple functions.

### Provisioned concurrency

If a function is sensitive to cold start, the code initialization process takes a long time, or many libraries need to be loaded, then you can set the provisioned concurrency for a specific function version to start function instances in advance and ensure smooth execution.

# Concurrency Management System

Last updated : 2024-12-02 20:07:18

You can configure the concurrency capability of each function.

## Concurrency Management

SCF supports account-level concurrency quota and function-level reserved concurrency quota.

```
Account-level concurrency quota
|- Function-level reserved quota
```

### Note:

[Provisioned concurrency](#) has nothing to do with the concurrency capability. It only serves as the ability to start instances in advance. Versions under the same function share the concurrency of the function.

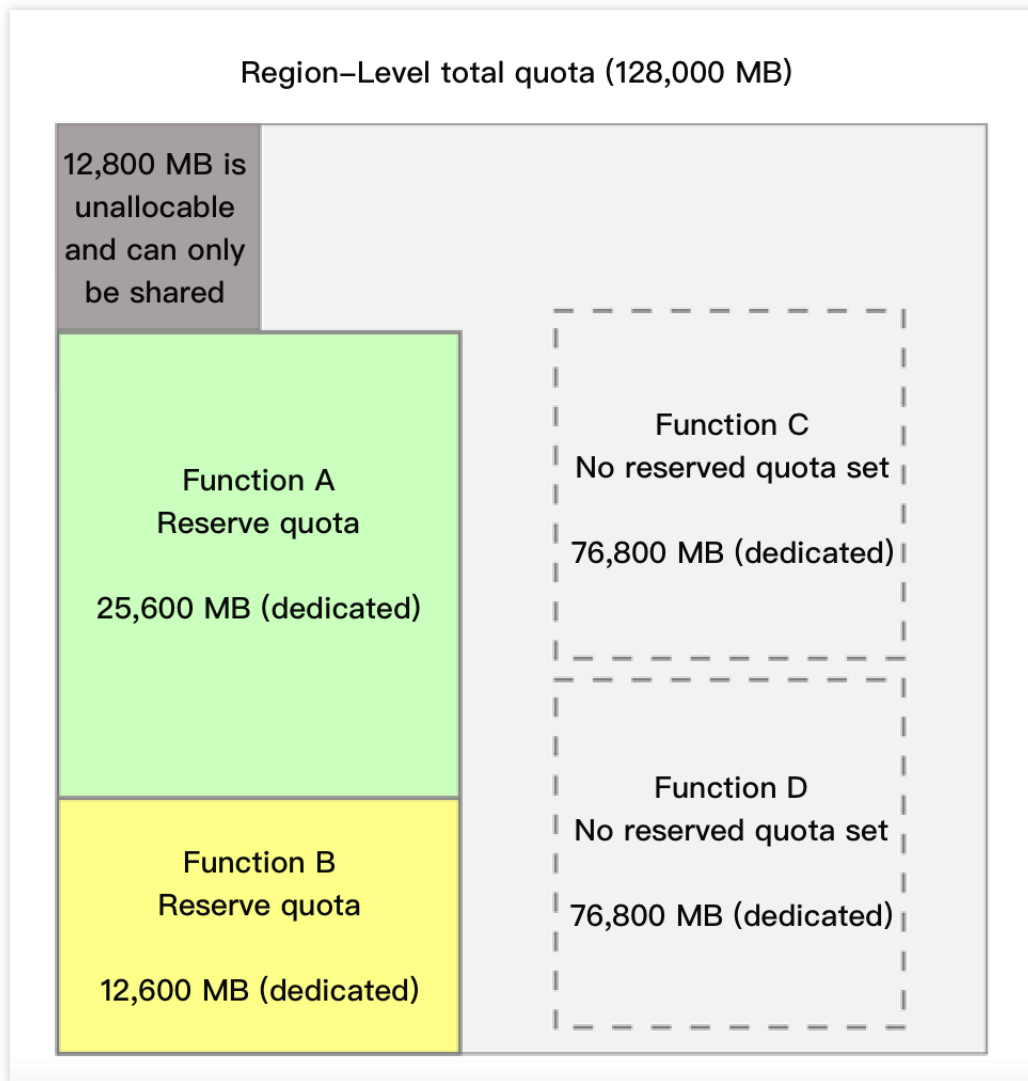
## Account-Level Concurrency Quota

Each account has a total concurrency quota limit at the region level. The default value is 128,000 MB or 64,000 MB. For more information, see [Quota Limits](#). The concurrency quotas between regions are independent of each other and don't affect each other.

By default, the account-level concurrency quota is shared by all functions in the current region. This means that at any specific time point, the sum of actual concurrently running instances of all functions can reach up to the concurrency quota of the account. Requests exceeding the concurrency quota will encounter the overrun error (432 ResourceLimitReached). You can [purchase extra packages](#) to increase the account-level quota.

By setting the function's [reserved quota](#), you can allocate the region-level concurrency to a certain function. 12,800 MB of the account-level concurrency quota can only be shared by functions without the reserved quota configured. This is to avoid the situation where functions with no reserved quota set cannot be invoked after the account-level quota is fully allocated.





## Reserved quota

Reserved quota is the concurrency management capability at the function level. When you set a reserved quota for a function, it will have the following two effects:

**The reserved quota is the upper limit of the concurrency quota of this function.** The sum of the concurrency quotas of all versions is less than or equal to the reserved quota.

The concurrency quota is allocated **exclusively** to this function and will not be shared by other functions.

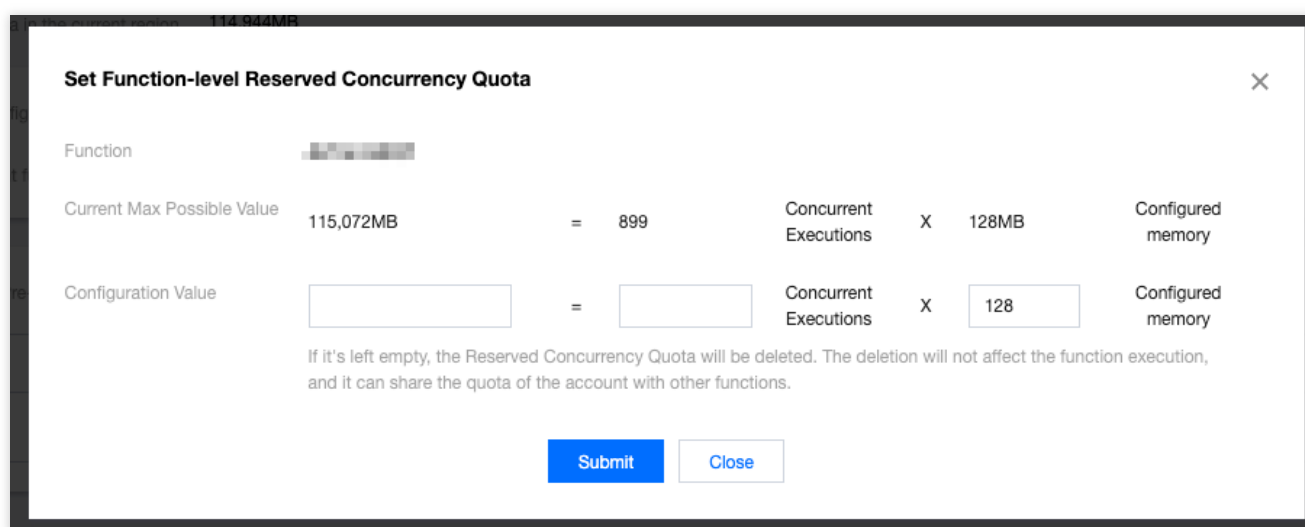
The reserved quota is the upper limit of the function concurrency quota. You can use this capability to manage the function concurrency and control the costs so as to avoid out-of-control costs. At the same time, you can also **disable a function by setting its reserved quota to 0**. Then, all requests for this function will encounter the concurrency overrun error.

The function reserved quota counts toward the regional concurrency quota. It can not be set if the regional unoccupied quota (= regional quota - reserved quotas allocated to other functions - 12,800 MB) is insufficient.

## Setting the reserved quota

You can set the desired reserved quota for a function in the following steps:

1. Log in to the [SCF console](#) and select **Functions** on the left sidebar.
2. On the **Functions** list page, select the name of the target function to enter the **Function management** page.
3. Select **Concurrency quota** from the left, in the **Reserved quota** section, click **Settings**.
4. In the **Set function-level reserved concurrency quota** window that pops up, set the desired maximum dedicated quota and click **Submit**.



**Set Function-level Reserved Concurrency Quota**

Function: [Function Name]

Current Max Possible Value: 115,072MB = 899 Concurrent Executions X 128MB Configured memory

Configuration Value: [Input Field] = [Input Field] Concurrent Executions X [Input Field] Configured memory

If it's left empty, the Reserved Concurrency Quota will be deleted. The deletion will not affect the function execution, and it can share the quota of the account with other functions.

**Submit** **Close**

Then, you can view the configuration status in the **Reserved Quota** section on the **Concurrency Management** page.

## Deleting reserved quota

If you no longer use the reserved quota, you can delete it. After the deletion, the function will share the concurrency quota at the account level with other functions.

### Note:

Deleting the reserved quota and setting the it to 0 are different configurations.

**Deleting reserved quota:** The function does not have a dedicated quota and uses the shared quota in the region. The upper limit is subject to the usage of the shared quota.

**Setting reserved quota to 0:** Both the function's dedicated quota and concurrency upper limit are 0, so the function cannot run and stops responding to triggering events.

1. Log in to the [SCF console](#) and select **Functions** on the left sidebar.
2. On the **Functions** list page, select the name of the target function to enter the **Function management** page.

3. Select **Concurrency quota** from the left, in the **Reserved quota** section, click **Delete**.
4. In the pop-up window, click **Confirm**.

# Provisioned Concurrency

Last updated : 2024-12-02 20:07:18

Provisioned concurrency can start concurrent instances in advance according to the configuration. SCF will not repossess these instances proactively; instead, it will ensure as much as possible that a corresponding number of concurrent instances are available to process requests.

The provisioned concurrency is version-specific. When it's set, the requested computing resources are prepared in advance to shorten the time required for cold start, and initialization of runtime environment and business code.

## Overview

Provisioned concurrency is a version-level configuration. When it's configured for a function version, the following happen:

1. SCF **immediately launches instances** until the configured value is reached.
2. SCF **does not repossess provisioned concurrent instances proactively**. Instead, it guarantees the number of provisioned concurrent instances as much as possible.

The speed of launching instances for provisioned concurrency defaults to 100 instances per minute. This has nothing to do with the speed for launching instances for elastic invocation. It does not count toward the quota of 500 concurrent instances/minute at the region level.

SCF does not repossess provisioned concurrent instances proactively. However, concurrent instances can become unavailable due to exiting the process or exceeding the memory limit. Once an instance becomes unavailable, it is released and a new instance is launched to meet the requirement of the provisioned concurrency configuration. During this period, the number of actual concurrent instances may be temporarily smaller than the number of provisioned concurrent instances. Concurrent instances do not incur charges when they are not launched. You can check the status of provisioned concurrency in **number of concurrent executions and provisioned concurrency** in the function monitoring data.

For the service stability and version consistency, provisioned concurrency **can be configured only on published versions, but not the `$LATEST` version**. For more information, see [Version Management](#).

## Provisioned Concurrency and Concurrency Management

Setting up the provisioned concurrency can speed up function initialization. However, the provisioned concurrency has nothing to do with the concurrency capability. It does not affect the maximum number of concurrent requests a function can process, which depends entirely on the function's reserved quota or region-level concurrency quota.

A function version with 128 MB memory is taken as an example below:

--	--	--	--	--

Scenario	Average concurrency	Provisioned concurrency	Function reserved quota	Result
Default	100 concurrent instances	Not configured	Not configured	All concurrent instances need to be initialized when they process requests for the first time. The concurrency quota of the function is affected by other functions under the same account and may be exceeded.
Function disabled	100 concurrent instances	Not configured	0 MB (0 concurrent instances)	The reserved quota is 0, the function is disabled, and all requests will get an overrun error.
No provisioned concurrency required	100 concurrent instances	Not configured	19,200 MB (150 concurrent instances)	All concurrent instances need to be initialized when they process requests for the first time. 150 concurrent instances can be guaranteed, and an overrun error will occur if this limit is exceeded.
80% provisioning	100 concurrent instances	10,240 MB (80 concurrent instances)	19,200 MB (150 concurrent instances)	80 concurrent instances don't need to be initialized, and 20 concurrent instances need to be initialized when they are invoked for the first time. 150 concurrent instances can be guaranteed, and an overrun error will occur if this limit is exceeded.
100% provisioning	100 concurrent instances	12,800 MB (100 concurrent instances)	19,200 MB (150 concurrent instances)	100 concurrent instances don't need to be initialized, and excessive concurrent instances need to be initialized when they are invoked for the first time. 150 concurrent instances can be guaranteed, and an overrun error will occur if this limit is exceeded.
Full provisioning	100 concurrent instances	19,200 MB (150 concurrent instances)	19,200 MB (150 concurrent instances)	All concurrent instances don't need to be initialized. 150 concurrent instances can be guaranteed, and an overrun error will occur if this limit is exceeded.

Overprovisioning	100 concurrent instances	25,600 MB (200 concurrent instances)	19,200 MB (150 concurrent instances)	It is the same as full provisioning except that it incurs additional fees for 50 more provisioned concurrent instances.
------------------	--------------------------	--------------------------------------	--------------------------------------	---

The concurrency management system (region-level concurrency quota and function-level reserved quota) is responsible for processing requests concurrently, while provisioned concurrency is responsible for ensuring that there are concurrent instances available to process requests. The decoupling of the two can implement capabilities such as [traffic switch with no initialization process](#).

## Provisioned Concurrency Limits

The configured provisioned concurrency quota is subject to the account-level quota; in other words, **the total provisioned concurrency quotas of all versions of all functions in a region is less than or equal to the concurrency quota at the account level**.

## Directions

### Adding provisioned concurrency

For a published function version, you can set a desired number of provisioned concurrent instances.

#### Note:

See [Publishing a version](#).

1. Log in to the SCF console and select **Functions** on the left sidebar.
2. On the **Functions** list page, select the name of the target function to enter the **Function management** page.
3. Select **Concurrency quota** from the left, go to the **Provisioned concurrency** page and click **Add provisioned concurrency**.
4. In the **Add provisioned function concurrency** window that pops up, select the target version, set the number of provisioned concurrent instances and click **Submit**.

After completing the settings, you can view the configuration status in **Provisioned Concurrency**.

### Updating provisioned concurrency

You can modify the number of concurrent instances as needed.

1. Log in to the SCF console and select **Functions** on the left sidebar.
2. On the **Functions** list page, select the name of the target function to enter the **Function management** page.
3. On the **Concurrency quota** page, select **Set** on the right of the target version.

4. In the **Configure provisioned function concurrency** window that pops up, update the value and click **Submit**. After the settings are completed, SCF will adjust the number of concurrent instances according to your modification in a certain period of time.

## Deleting provisioned concurrency

You can delete a provisioned concurrency configuration if you don't need to use it.

1. Log in to the SCF console and select **Functions** on the left sidebar.
2. On the **Functions** list page, select the name of the target function to enter the **Function management** page.
3. Select **Concurrency quota** from the left, go to the **Provisioned concurrency** page, locate the version you want to delete and click **Delete**.
4. In the **Delete provisioned function concurrency** window that pops up, click **OK**.

After the configuration is deleted, concurrent instances are released gradually.

## More

### Using provisioned concurrency for traffic switch

You can set the reserved quota for a function based on the volume of concurrent business requests and configure the provisioned concurrency based on the traffic switch needs as instructed below:

1. Publish a new version.
2. Set the desired provisioned value for the new version.
3. Wait for the provisioned concurrent instances of the new version to be started completely.
4. Gradually switch the traffic from the previous version to the new version through [traffic routing configuration](#). If a problem occurs, switch the traffic back to the previous version.
5. Switch the traffic completely to the new version and delete the provisioned concurrency of the old version if nothing goes wrong after a period of time.

In the example below, the reserved quota of the function is 150 concurrent instances, which means the function can concurrently process up to 150 requests. You can set 100 provisioned concurrent instances for multiple versions (100 instances are started for each version), so as to switch the traffic with no initialization needed.

Scenario	Average Business Concurrency	Provisioned Concurrency	Function Reserved Quota	Effect
100% provisioning	100 concurrent instances	12,800 MB (100 concurrent instances)	19,200 MB (150 concurrent instances)	100 concurrent instances don't need to be initialized, and excessive concurrent instances need to be initialized when they are invoked for the first time. 150 concurrent instances can be guaranteed, and an

				overrun error will occur if this limit is exceeded.
--	--	--	--	---

As shown in the figure below, 100 provisioned concurrent instances is configured for both version 4 and version 5, and you can use the traffic grayscale capability to switch the 100 concurrent instances of the business from version 4 to version 5. No matter how the 100 concurrent instances are allocated to versions 4 and 5 according to any proportion, no instances will need to be initialized, thus making it easier for you to publish a version and switch traffic more quickly.

4	12,800MB (100个) / 12,800MB (100个)
5	12,800MB (100个) / 12,800MB (100个)



# Scheduled Provisioned Concurrency

Last updated : 2024-12-02 20:07:18

## Overview

Scheduled provisioned concurrency is an elastic policy for [provisioned concurrency](#). You can reasonably configure provisioned concurrency based on the business conditions and upgrade/downgrade it at specified times to improve the utilization of provisioned concurrent instances and reduce the fees incurred by idle resources. If the number of concurrent instances actually required by a function exceeds that configured in scheduled provisioned concurrency, auto scaling will be performed as needed. This feature supports the following task types: one time, daily, Monday to Friday, Saturday and Sunday, and custom.

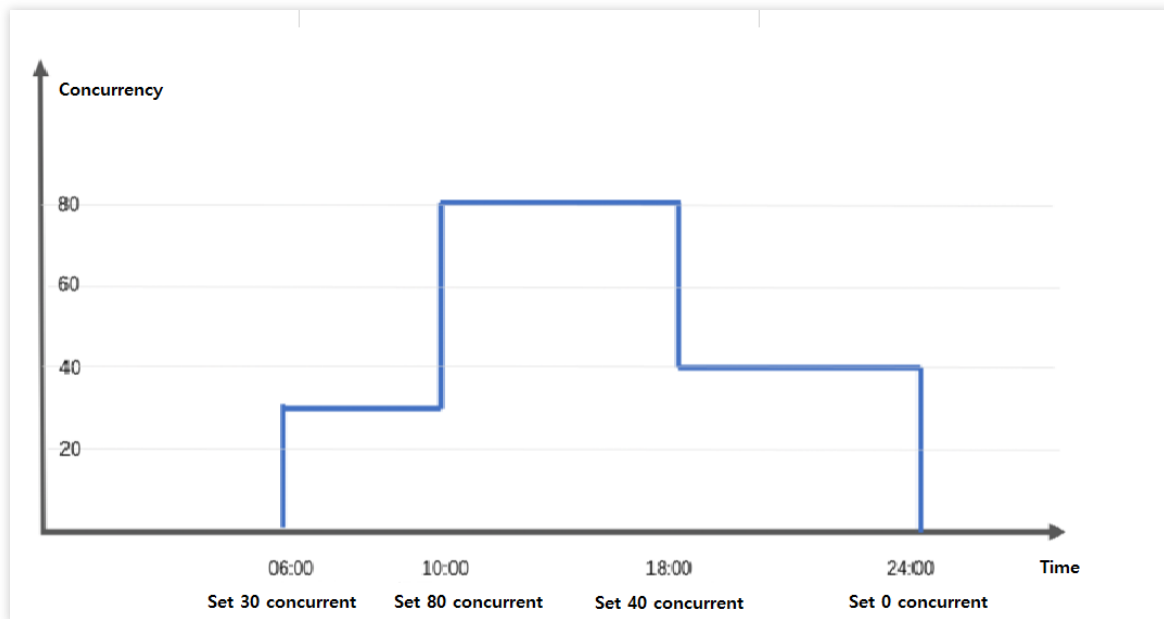
## Use Cases

Functions with regularly fluctuating traffic, such as data processing functions.

Functions whose business traffic peaks can be predicted, such as functions for events or other scheduled businesses.

## Features and Limits

The configured value in scheduled provisioned concurrency is the target number of concurrent instances during task execution; for example, if you need to configure four scheduled tasks for your business, i.e., starting 30 concurrent instances at 6:00, 80 at 10:00, 40 at 18:00, and 0 at 24:00, the final provisioned concurrency fluctuation will be as follows:



## Notes

A cron expression for scheduled provisioned concurrency has seven required fields separated with spaces. For more information, see [Timer Trigger Description](#).

There is a limit to the number of scheduled provisioned concurrency tasks under one user account on one function version. For more information, see [Quota Limits](#). To increase the maximum number of scheduled tasks (i.e., quota limit), you can [submit a ticket](#) for application.

SCF will adjust the speed of starting provisioned concurrency based on your business, which is 100 concurrent instances per minute by default. You should reasonably configure the start time of scheduled provisioned concurrency. For more information, see [Concurrency Overview](#).

If two scheduled provisioned concurrency tasks are scheduled for the same time, the new one will overwrite the old one.

## Example

To start a scheduled provisioned concurrency task at 12:00 on November 13, 2021 to sustain your business traffic peak and end the task at 16:00 on the same day, perform the following operations:

### Starting scheduled task

To start a provisioned concurrency task as scheduled, you need to add a scheduled task. Select the start time and configure the target number of concurrent instances as shown below:

▼ timer-1

28 37 16 21 01 \* 2022

Edit Delete


Action Name

timer-1

Repeat

Once only ▼

Start time

2021-11-03 12:00:00 

Configuration Value

512

=

1

Concurrent Executions

X

512MB

Configured memory

Save Cancel

Add Scheduled Action

Submit

Close

## Ending scheduled task

To end a provisioned concurrency task as scheduled, you need to **add another scheduled provisioned concurrency task**. Select the desired end time of the previous task as the start time of the new task and configure the target number of concurrent instances to 0 as shown below:

▼ timer-1

28 37 16 21 01 \* 2022

EditDelete

Action Name

timer-2

Repeat

Once only ▼

Start time

2021-11-03 16:00:00

Configuration Value

0

=

0

Concurrent Executions

X

512MB

Configured memory

Save

Cancel

Add Scheduled Action

Submit

Close

# Dynamic Provisioned Concurrency Metric

Last updated : 2024-12-02 20:07:18

## Overview

Dynamic provisioned concurrency metric is an elastic policy for [provisioned concurrency](#). SCF will periodically collect information about actual concurrent function executions and control the dynamic scaling of the provisioned concurrency feature based on the configured metrics of maximum concurrency, minimum concurrency, and target concurrency usage. This makes the number of provisioned concurrent function instances closer to the actual resource usage, improves the usage of provisioned concurrent instances, and reduces the fees incurred by idle resources. If the number of concurrent instances actually required by a function exceeds that configured dynamic provisioned concurrency metric, auto scaling will be performed as needed.

## Application Scenarios

Businesses that are sensitive to idle provisioned concurrency fees.

Functions that are sensitive to cold start with unpredictable business traffic peaks.

## How It Works

When the dynamic provisioned concurrency metric is configured, scaling will be performed according to the configured dynamic policy. If the metrics of minimum concurrency, maximum concurrency, and concurrency usage are set, the system will guarantee the minimum concurrency of provisioned resources, and the provisioned concurrency will be dynamically scaled between the minimum and maximum values.

### Scaling policy

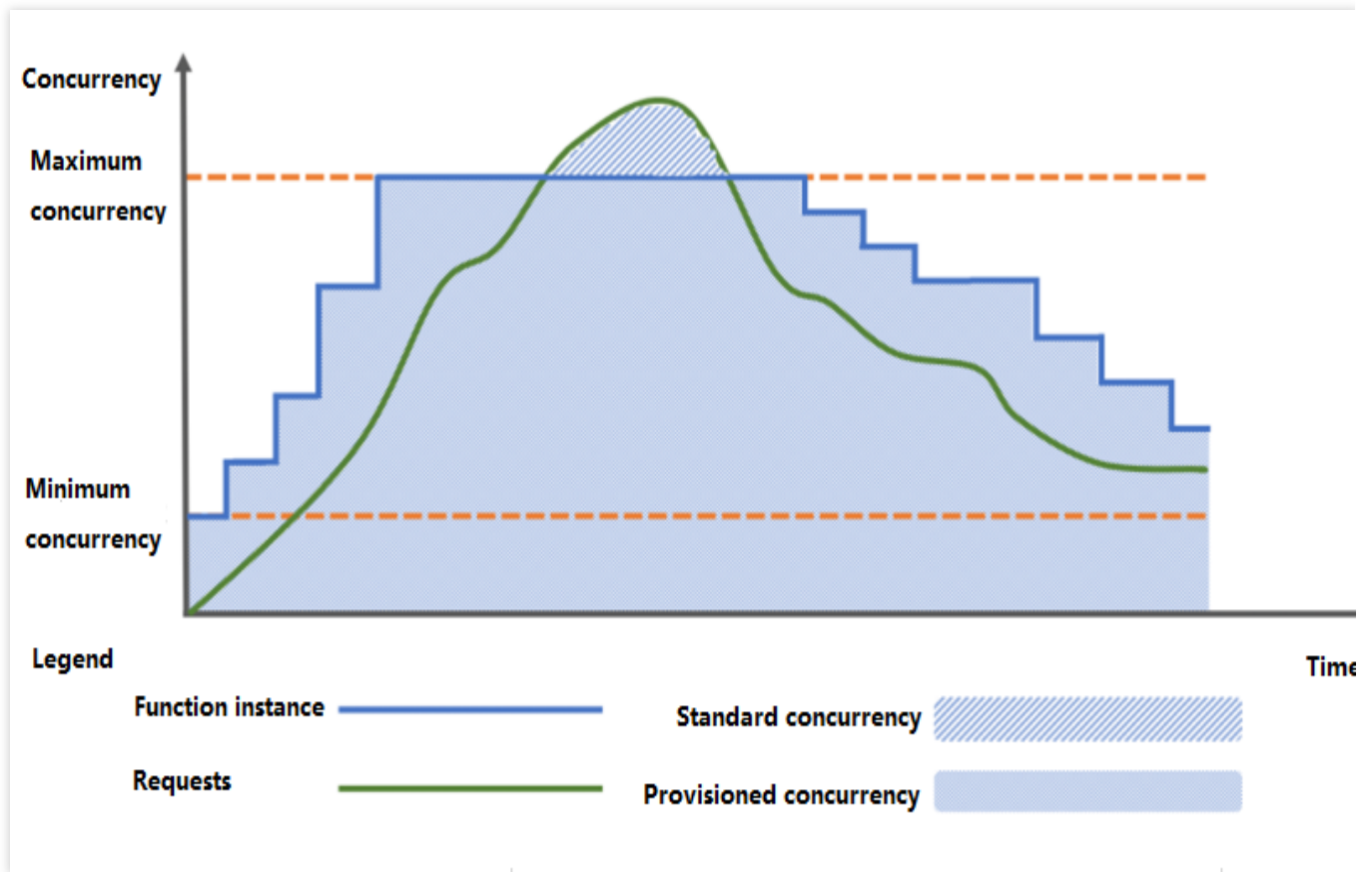
**Concurrency expansion:** The system will expand the concurrency when the actual number of business requests increases and triggers the threshold for concurrency expansion until the maximum concurrency is reached. For excessive requests, the concurrency will be expanded as needed.

Concurrency expansion frequency: Concurrency expansion will be performed once every ten seconds, without a time window.

**Concurrency reduction:** The system will reduce the concurrency when the actual number of business requests drops and triggers the threshold for concurrency reduction until the minimum concurrency is reached.

Concurrency reduction frequency: A time window of ten minutes is provided to implement a relatively conservative concurrency reduction process; that is, concurrency reduction operations will not be performed repeatedly within the

time window, which can be understood as the cooling time for releasing a skill in a game. If not performed previously, a concurrency reduction operation can be performed in ten seconds.



### Target provisioned concurrency value

The target provisioned concurrency value is jointly determined by the metrics of current concurrency and the target concurrency usage.

**Target provisioned concurrency value** = Current total number of function instances \* current concurrency usage / target concurrency usage = Current total number of function instances \* (current concurrency / current total number of function instances) / target concurrency usage = **Current concurrency/target concurrency usage**.

Calculation example of the target provisioned concurrency value: If the current concurrency is 100 and the target concurrency usage is 80%, then the target provisioned concurrency value will be  $100 / 80\% = 125$ .

### Concurrency usage

The concurrency usage of a function refers to the ratio of the number of concurrent requests being responded to by the current function instances to the current total number of function instances. Its value range is  $[0,1)$ .

### Minimum concurrency

The minimum concurrency refers to the minimum required number of provisioned concurrent instances of a function, i.e., the lower limit for concurrency reduction.

### Maximum concurrency

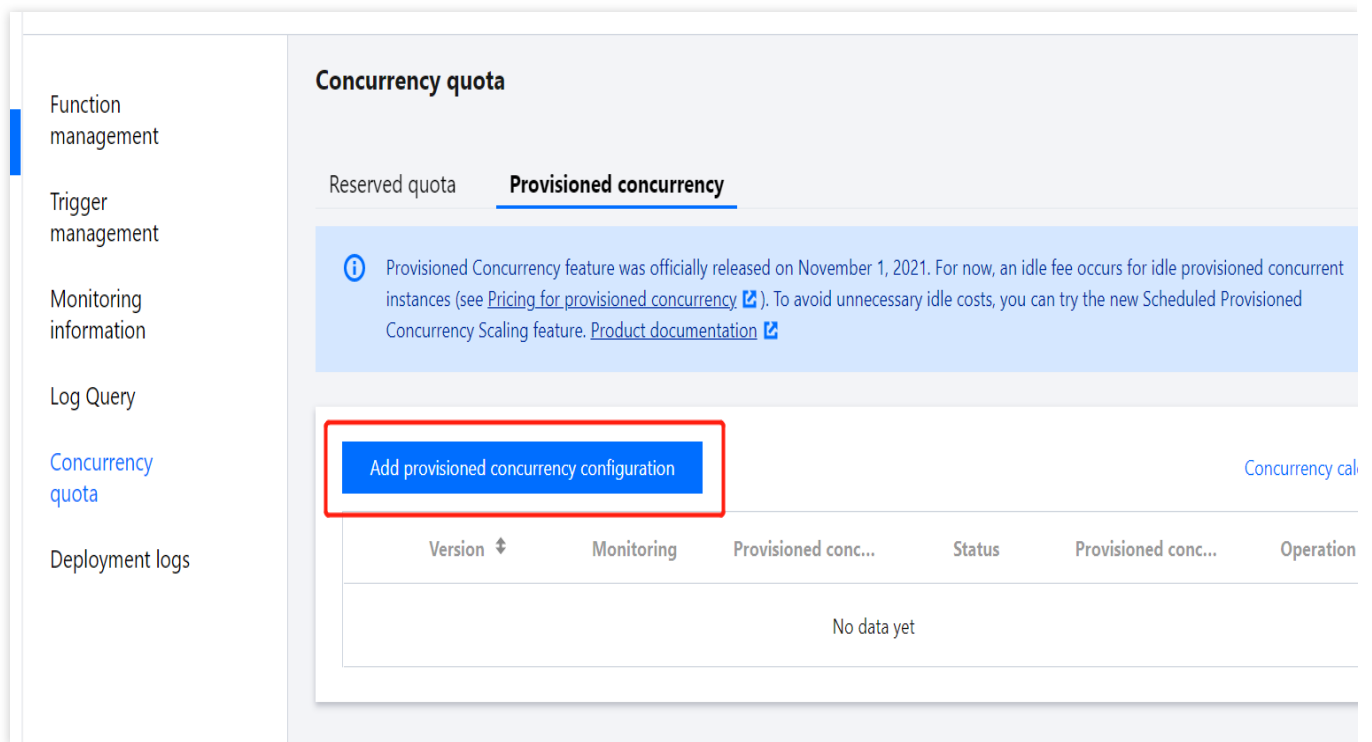
The maximum concurrency refers to the maximum number of provisioned concurrent instances of a function, i.e., the

upper limit for concurrency expansion.

## Directions



### Adding dynamic provisioned concurrency metric

1. Log in to the SCF console and select [Function Service](#) on the left sidebar.
2. On the **Function Service** list page, select the name of the target function to enter the **Function Management** page.
3. On the left, select **Concurrency quota** > **Provisioned concurrency** to enter the **Provisioned concurrency** tab.
4. On the **Provisioned concurrency** tab, click **Add provisioned concurrency configuration** as shown below:




5. In the **Add provisioned function concurrency** pop-up window, select **Dynamic provisioned concurrency metric** for **Provisioned concurrency type**, select the function version, set **Min concurrency**, **Max concurrency**, and **Concurrency usage metric** based on the business scenario, and click **Submit** as shown below:


## Add provisioned function concurrency

 To configure the provisioned concurrency, you need to [release a version](#)  first.

Provisioned concurrency type ☐ Basic provisioned concurrency

☒ Dynamic provisioned concurrency metric 

Function nextjs\_4zmvm4

Version \*  

Version description -

Current max possible value 128,000MB = 1,000 Concurrent executions X 128MB Configured memory

Min concurrency \*  =  Concurrent executions X 128MB Configured memory

Max concurrency \*  =  Concurrent executions X 128MB Configured memory

Concurrency Usage Metric \*

Submit

Close

After completing the settings, you can view the configuration status in **Provisioned Concurrency**. It will take some time for the SCF backend to add the instances and display the number of ready-to-start concurrent instances and completion status in the list.

## Updating dynamic provisioned concurrency metric


When updating the dynamic provisioned concurrency metric, you can modify **Provisioned concurrency type**, **Min concurrency**, **Max concurrency**, and **Concurrency usage metric**.

1. Log in to the SCF console and select [Function Service](#) on the left sidebar.
2. On the **Function Service** list page, select the function for which to update the provisioned concurrency to enter the **Function Management** page.
3. On the left, select **Concurrency quota** > **Provisioned concurrency** to enter the **Provisioned concurrency** tab.
4. On the **Provisioned concurrency** tab, select **Settings** on the right of the target version.
5. In the **Set provisioned function concurrency** pop-up window, update the set value and click **Submit** as shown below:



## Add provisioned function concurrency

 An idle fee occurs for idle provisioned concurrent instances. Instances in use are billed elastically. See [Billing Mode](#) 

Provisioned concurrency type ☐ Basic provisioned concurrency  
☒ Dynamic provisioned concurrency metric 

Function helloworld-1638349988

Version \*  

Version description lii

Current max possible value	128MB	=	1	Concurrent executions	X	128MB	Configured memory
----------------------------	-------	---	---	-----------------------	---	-------	-------------------

Min concurrency *	<input type="text" value="128"/>	=	<input type="text" value="1"/>	Concurrent executions	X	128MB	Configured memory
-------------------	----------------------------------	---	--------------------------------	-----------------------	---	-------	-------------------

Max concurrency *	<input type="text" value="128"/>	=	<input type="text" value="1"/>	Concurrent executions	X	128MB	Configured memory
-------------------	----------------------------------	---	--------------------------------	-----------------------	---	-------	-------------------

Concurrency Usage Metric \*

Submit

Close

**Note:**


Basic provisioned concurrency and dynamic provisioned concurrency metric are supported for the provisioned concurrency type. After the provisioned concurrency type is updated, the previously set type will become invalid.

**Deleting dynamic provisioned concurrency metric**

1. Log in to the SCF console and select [Function Service](#) on the left sidebar.
2. On the **Function Service** list page, select the function for which to delete the provisioned concurrency to enter the **Function Management** page.
3. On the left, select **Concurrency quota > Provisioned concurrency** to enter the **Provisioned concurrency** tab.
4. On the **Provisioned concurrency** tab, select **Delete** on the right of the target version as shown below:

Add provisioned concurrency configuration

Concurrency calcu

Version ↕	Monitoring	Provisioned conc...	Status	Provisioned conc...	Operation
▶ 1		128MB (1) / 128MB (1)	Completed	Basic provisioned concurrency	<a href="#">Settings</a> <a href="#">Delete</a>

5. In the **Delete provisioned function concurrency quota** pop-up window, click **OK**.

# Concurrency Overrun

Last updated : 2024-12-02 20:07:18

## Concurrency Overrun

Concurrency overrun (ResourceLimitReached) refers to a situation where the number of concurrent executions of an SCF function at the same time exceeds the [quota limit](#), leading to a function error. Concurrency overrun is divided into two types: sync invocation and async invocation.

### Async invocation

Types of async invocation include async invocation by [TencentCloud API trigger](#), [COS trigger](#), [scheduled trigger](#), [CMQ topic trigger](#), [CLS trigger](#), [MPS trigger](#), etc. For specific trigger invocation types, please see the relevant trigger documentation.

When concurrency overrun occurs in an async invocation, the function will automatically retry. For more information, please see [Error Types and Retry Policies](#).

### Sync invocation

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

In sync invocation, the error message will be directly returned; therefore, when an error occurs in sync invocation, the platform will not automatically retry, and the retry policy (i.e., whether to retry and the number of retries) will be determined by the invoker. In this case, the function will return a [432 status code](#) and will not retry the request.

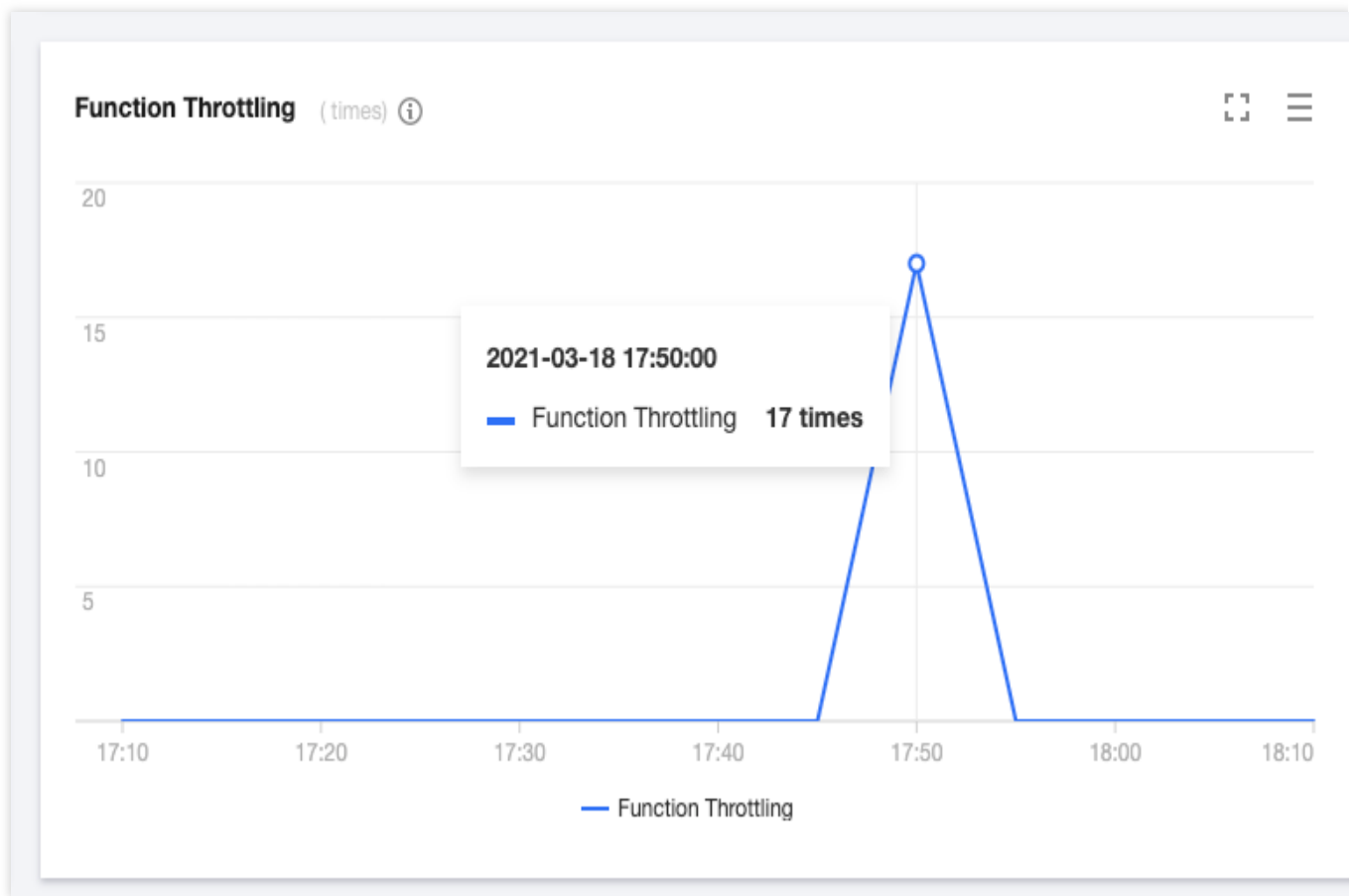
## Concurrency Overrun Troubleshooting

### Viewing concurrency overrun monitoring data

You can view the number of limited times and specific logs of the function in the SCF console.

#### Viewing the number of limited times

1. Log in to the [Serverless console](#) and select **Function Service** on the left sidebar.
2. On the **Function Service** page, select the name of the function you want to view to enter its details page.
3. In **Function Management**, select **Monitoring information** > **Limited times** to view the limited times of the function as shown below:



### Viewing function limit log

1. Log in to the [Serverless console](#) and select **Function Service** on the left sidebar.
2. On the **Function Service** page, select the name of the function you want to view to enter its details page.
3. In **Log Query**, select **Invocation Logs** > **Too many invocations** to view specific limit logs of the function as shown below:

The screenshot shows the 'Invocation Logs' interface. At the top, there are tabs for 'Invocation Logs' and 'Advanced Retrieval'. Below the tabs, there are filters for 'Version' (set to '\$LATEST'), a dropdown menu for 'Too many invocations' (highlighted with a red box), a time range filter (set to 'Last 1 hour'), and a date range (2021-03-18 17:11:35 ~ 2021-03-18 18:11:35). A 'Refresh' button is also present. The main table lists invocation logs with columns for time and status. All listed invocations are marked as 'Invoke failed'. The right panel shows details for a specific request, including the 'Request ID', 'Time', 'Runtime', 'Billed time', and 'Execution memory'. It also displays the 'Returned data' as a JSON object: `{\"errorCode\":-1,\"errorMessage\":\"ResourceLimitReached\",\"statusCode\":432}`. Below this, a 'Log' section shows the request lifecycle: 'START Request', 'ERROR Request', 'END RequestId', and 'Report Request'.

## Fixing concurrency overrun

For **async invocation**, there is a system retry policy for concurrency overrun errors, which can automatically process concurrency overrun and retry the requests. Normally, you don't need to perform any operations for concurrency overrun in async invocation; instead, during the maximum waiting time you set, the SCF platform will automatically retry the requests.

When an error occurs in **sync invocation**, the error message will be directly returned, and the request will not be retried.

### Note:

In async invocation, if your business system is more sensitive to timeliness, you can reduce or minimize the impact of overrun errors on your business system by configuring reserved quota. For example, if you want that important messages will not be lost after the set maximum retention time elapses, you should configure a dead letter queue (DLQ).

## Configuring DLQ

A DLQ is a CMQ queue under your account used to collect error event information and analyze causes of failures. If you have configured a DLQ for a function, messages in retry failures caused by overrun will be sent to it. For more information, please see [Creating Dead Letter Queue](#).

## Configuring reserved quota

The reserved quota is the maximum quota used to guarantee the available concurrency of a function. By configuring the reserved quota, the function can start enough concurrent instances within the quota, and the maximum number of concurrent instances can reach the configured quota. After the reserved quota is set, the function no longer shares the concurrency quota at the account level with other functions, which can reduce the possibility of concurrency overrun and guarantee smooth function execution. For more information, please see [Reserved Concurrency](#).

# Trigger Management

## Creating a Trigger

Last updated : 2024-12-02 20:07:18

After creating a function, you can create a trigger to associate the function with an event source. The associated event source will trigger the function synchronously or asynchronously as specified when an event is generated, and the event will be passed to the entry function as an input parameter upon triggering.

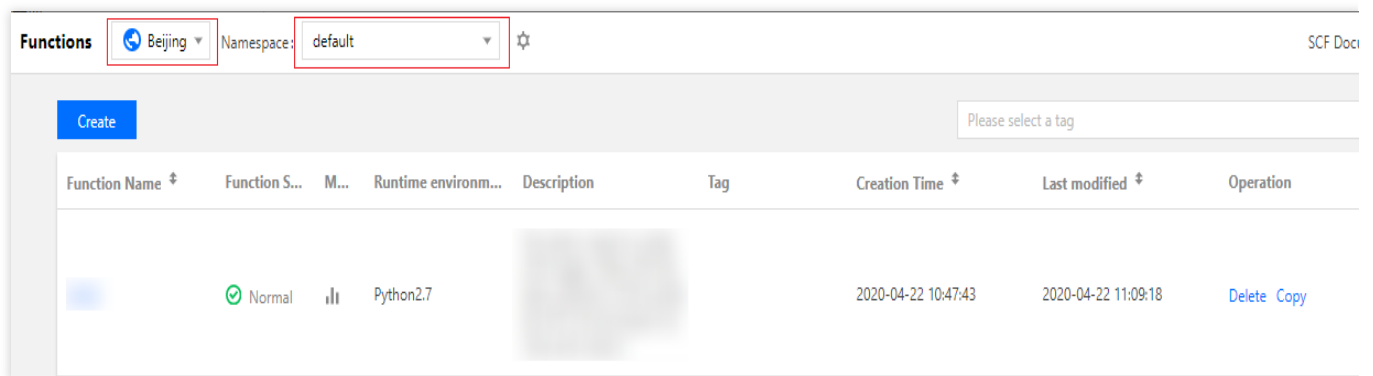
A function trigger can be created in the console or through Serverless Cloud Framework CLI.

### Note:

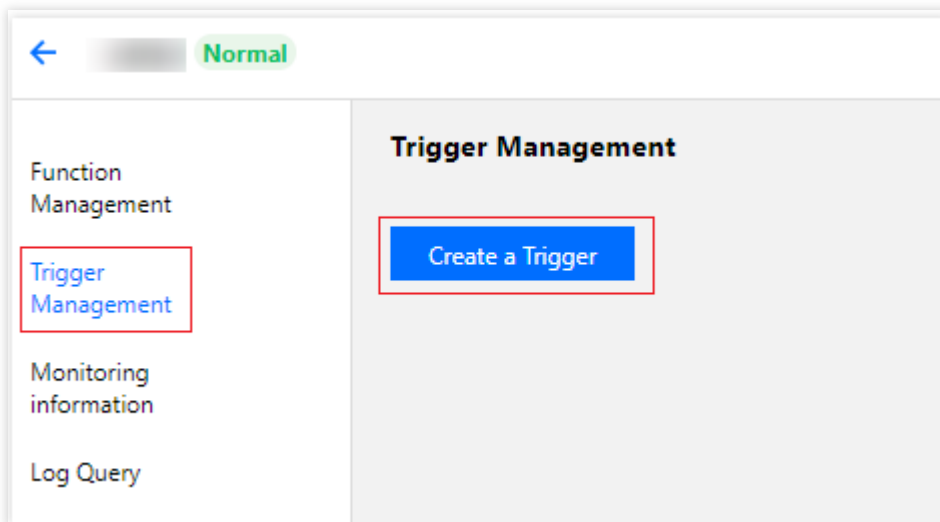
Only API Gateway triggers can be created for HTTP-triggered functions. For more information, see [Creating and Testing Function](#).

## Creating a Trigger in the Console

1. Log in to the [SCF console](#) and click **Functions** on the left sidebar.
2. At the top of the **Functions** page, select the region and namespace where the function resides.



3. Click the function name to enter the function details page.
4. Select **Trigger management** on the left to enter the trigger browsing and operation page. Click **Create trigger** to create a trigger.



5. In the **Create trigger** pop-up window, select the trigger alias/version and trigger method.

### Create trigger

Tencent Cloud CMQ will be discontinued by June 2022. No more CMQ triggers can be created. Existing CMQ triggers are not affected. For detail: see [CMQ Documentation](#).

Triggered alias/version

Alias: Default traffic

Trigger method

Scheduled triggering

The scheduled trigger will trigger the SCF function automatically by the specified period. [Learn More](#)

Scheduled task name

SCF-timer-

Trigger period

Every day (Execute once every day at 01

Remarks

No

Enable now

☒ Enable

If it's checked, the scheduled trigger will be activated and executed at the start point of next period.

Submit

Close

Trigger alias/version: Switch to the desired trigger version. A trigger can be created on a specified version of a function. An event of the trigger created in this manner will trigger the code on the specified version. For more information, see [Overview](#).

**Note:**



There are certain quota limits for the total number of triggers and number of triggers in each type for a function. Triggers created on different versions take up the quota of the function. To increase the limit, [contact us](#) for application.

Trigger method: The information to be entered varies by trigger method. For example, for a timer trigger, you need to enter the trigger name, cycle, and status. For a COS trigger, you need to enter the COS bucket, event type, and prefix/suffix filters. For more information, see [Overview](#).

6. After completing the trigger configuration, click **Submit** to create the trigger.

## Creating Trigger on Serverless Cloud Framework CLI

### Note:

Before starting, install the Serverless Cloud Framework CLI tool first as instructed in [Installation](#).

For local functions, add the trigger description in the `serverless.yml` file. Then, run the `scf deploy` command on Serverless Cloud Framework CLI to add a trigger to the function.

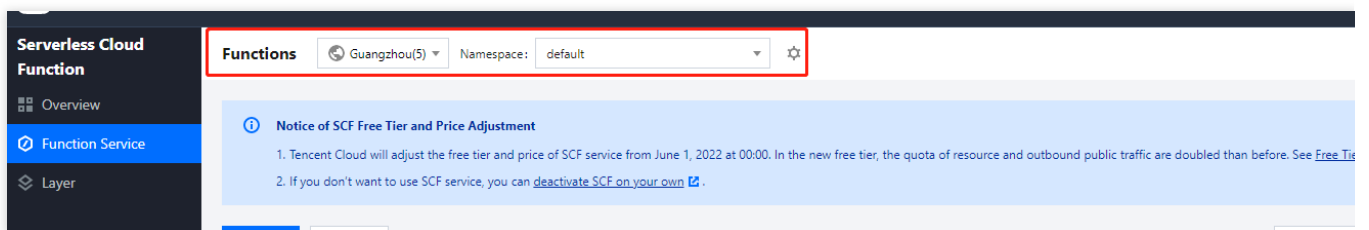
# Deleting Triggers

Last updated : 2024-12-02 20:07:18

You can disassociate a function from an event source by deleting a trigger in the console. After disassociation, the event source will no longer trigger the function.

## Deleting a Trigger in the Console

1. Log in to the SCF console and select **Functions** on the left sidebar.
2. At the top of the **Functions** page, select the region and namespace where the function resides as shown below:



3. Click the function name to enter the function details page.
4. Select **Trigger management** on the left to enter the trigger browsing and operation page. Click **Delete** in the top-right corner of the target trigger as shown below:



In the pop-up window, click **Confirm deletion**.

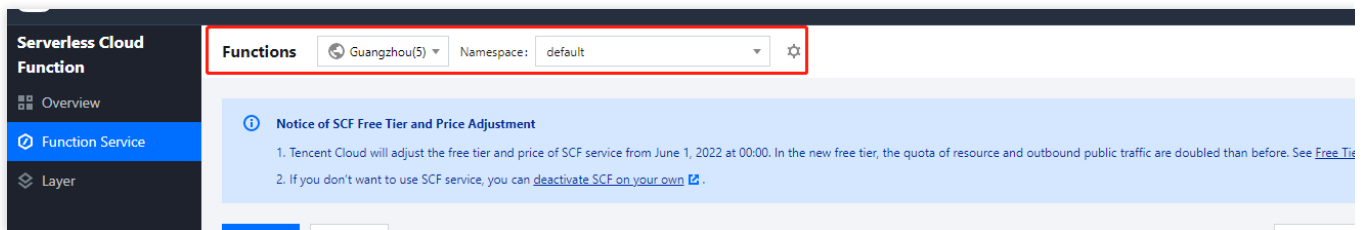
# Enabling/Disabling Triggers

Last updated : 2024-12-02 20:07:18

You can disable a trigger to temporarily prevent a function from being triggered by an event occurring at the event source. This document describes how to do so in the console.

## Enabling or Disabling a Trigger in the Console

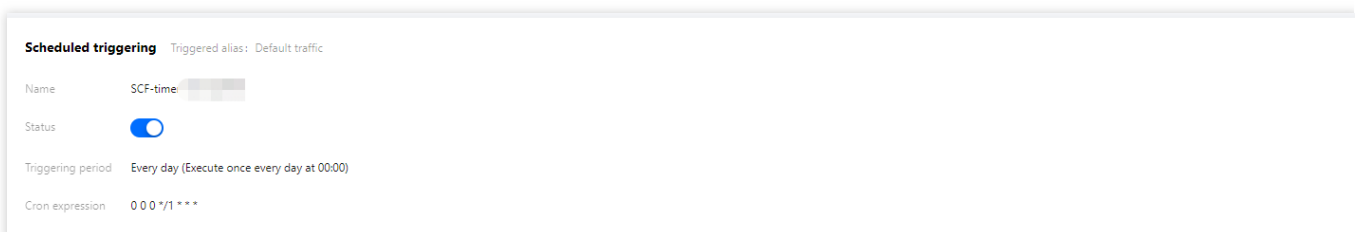
1. Log in to the SCF console and select **Functions** on the left sidebar.
2. At the top of the **Functions** page, select the region and namespace where the function resides as shown below:



3. Click the function name to enter the function details page.
4. Select **Trigger management** on the left to enter the trigger browsing and operation page. Click



in the status of the target trigger to enable or disable it as shown below:



## Setting the Enables/Disabled Status of a Trigger During Creation

When you create a trigger, you can set its enabled/disabled status, and the trigger will be in the set status once created.

For example, when creating a timer trigger, if you want the trigger to take effect later as needed, you can deselect **Enable now** as shown below:

**Create trigger** ✕

Tencent Cloud CMQ will be discontinued by June 2022. No more CMQ triggers can be created. Existing CMQ triggers are not affected. For details, see [CMQ Documentation](#).

Triggered alias/version: Alias: Default traffic ▼

Trigger method: Scheduled triggering ▼

The scheduled trigger will trigger the SCF function automatically by the specified period. [Learn More](#)

Scheduled task name ⓘ: SCF-time

Trigger period: Every day (Execute once every day at 01 ▼)

Remarks ⓘ: No ▼

Enable now: ☐ Enable

If it's checked, the scheduled trigger will be activated and executed at the start point of next period.

[Submit](#) [Close](#)

After the trigger is created, you can enable it by switching its enabled/disabled status.

## Notes

At present, the enabled/disabled status switch is not supported for certain triggers, and the **Enable** button is not displayed for them in the console. When this is supported for them subsequently, the status and button will be displayed accordingly.

# Function URL

## Function URL Overview

Last updated : 2025-06-17 16:40:37

### Overview

A function URL is a dedicated HTTP(S) endpoint for a function. After configuring a function URL for a function, you can invoke the function through its HTTP(S) endpoint using a Web browser, curl, Postman, or any HTTP client. You can create and configure a function URL through the SCF console or SCF API/CLI. Once a function URL is created, its URL endpoint will remain unchanged permanently. The endpoint format of the function URL is as follows:

```
public network: https://<app-id>-<url-id>.<region>.tencentscf.com
private network: https://<app-id>-<url-id>.in.<region>.tencentscf.com
```

Function URL and trigger exist at the same level, suitable for event functions and Web functions. You can enable function URL and concurrently configure triggers such as API Gateway.

Function URL is one-to-one bound to the version and alias of a function. You need to manually turn on or off the function URL for each version and alias. By default, the function URL is off.

#### Notes:

If need to generate a WSS address, please enable a WebSocket support in the function configuration.

## Calling Parameter

### Event Function

#### Request parameters.

When the URL receives a request, the function will be triggered to run. At the same time, the URL will send the relevant information of the request to the triggered function in the form of an event input parameter. The relevant information of the request contains, for example, the specific service and API rule that received the request, the actual path of the request, the method, the request path, headers, query, etc.

```
// Example of Event detailed information [compatible with apigw protocol, remove re
{
  "body": "{\\"test\\":\\"hello world\\"}",
  "headers":{
    "accept": "*/*",
    "accept-encoding": "gzip, deflate, br",
```

```
    "cache-control": "no-cache",
    "connection": "keep-alive",
    "content-length": "17",
    "x-scf-remote-addr": "111.206.96.145" // this field is the client request IP
  },
  "httpMethod": "POST",
  "path": "/",
  "queryString": {
    "a": "1",
    "b": "2"
  }
}
```

## Response Parameters

When the function returns a response, the function parses the response and converts it to an HTTP response.

Standard response payload:

```
{
  "statusCode": 201,
  "headers": {
    "Content-Type": "application/json",
    "My-Custom-Header": "Custom Value"
  },
  "body": "{ \"message\": \"Hello, world!\" }"
}
```

The function will infer the response format for you. If your function returns valid JSON and does not return a `statusCode`, the function assumes the `statusCode` is 200, the content-type is `application/json`, and the body is the function response.

The standard response parameter format of the function response is as follows:

Function output	HTTP response (the content seen by the client)
<pre>"Hello, world!"</pre>	<pre>HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 content-type: application/json content-length: 15  "Hello, world!"</pre>
<pre>{   "message": "Hello, world!" }</pre>	<pre>HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 content-type: application/json</pre>

	<pre>content-length: 34  {   "message": "Hello, world!" }</pre>
<pre>{   "statusCode": 201,   "headers": {     "Content-Type": "application/json",     "My-Custom-Header": "Custom Value"   },   "body": JSON.stringify({     "message": "Hello, world!"   }) }</pre>	<pre>HTTP/2 201 date: Wed, 08 Sep 2021 18:02:24 content-type: application/json content-length: 27 my-custom-header: Custom Value  {   "message": "Hello, world!" }</pre>

## Web Function

When the URL receives an HTTP request, the function will be triggered to execute. At this time, the URL will directly forward the HTTP request without performing event type format conversion. Meanwhile, the request response is also directly forwarded.

# Creating a Function URL

Last updated : 2025-05-09 10:05:01

This document introduces how to create a function URL using the console and using API.

## Creating a Function URL through the Console

1. Log in to Serverless Console, click **Function Service** in the left sidebar.
2. On the **Function Service** page, click the function name to enter the function detail page.
3. Select **Function URL** in the left sidebar, and click **Create Function URL**. As shown below:
4. On the **Create Function URL** page, refer to the following information to proceed with creation.

Configuration Item	Description
Alias / Version	URL can be bound at the alias or version dimension. Each alias or version can create only one URL.
Public network access / Private network access	You can enable public network or private network URL visit based on business needs.
Authorization type	<p>The authorization type supports selecting <b>Open</b> and <b>CAM Authentication</b>.</p> <p><b>Open:</b> No identity verification is required for function requests. Anonymous access is supported. Anyone can initiate an HTTP call to your functions.</p> <p><b>CAM authentication:</b> Authentication validation is required for functions via CAM. Users can perform resource management and usage rights configuration based on the function InvokeFunctionUrl API. For details, see Function URL Authentication and Authorization Configuration.</p>

5. Click **Submit** to complete the creation.

## Using API to Create a Function URL

Creating function URLs and triggers shares the same API interface. For details on shared parameters, see Set Function Trigger Method. For the `Type` parameter, please enter `http`. The configuration instructions for the `TriggerDesc` parameter are as follows:



Name	Type	Required	Description
AuthType	String	Yes	Authorization type. CAM indicates that function URL authentication and authentication configuration is required. NONE means authorization is not required.
NetConfig	NetConfig	Yes	network access configuration. Example value: <pre>{   "EnableIntranet": true, "EnableExtranet": false }</pre>

### NetConfig

Name	Type	Required	Description
EnableIntranet	Bool	Yes	Whether intranet access is enabled
EnableExtranet	Bool	Yes	Whether to enable public network access

### Parameter example

```
trigger_desc = {  
    "AuthType": "NONE",  
    "NetConfig": {  
        "EnableIntranet": true,  
        "EnableExtranet": false  
    }  
}  
  
params = {  
    "FunctionName": "helloworld",  
    "TriggerName": "func_url",  
    "TriggerDesc": json.dumps(trigger_desc),  
    "Type": "http",  
    "Namespace": "default",  
    "Enable": "OPEN",  
}
```

# Function URL Authentication and Authorization Configuration

Last updated : 2025-06-17 16:41:25

## Overview

You can control access to the function URL by configuring an authentication and certification policy.

When configuring the function URL, you must specify one of the following authentication options:

**CAM Authentication:** Authentication validation is required for the function. Users can perform resource management and usage rights configuration through the function `InvokeFunctionUrl` interface. You can open or restrict access to the interface by configuring `InvokeFunctionUrl` policy permissions.

**Open:** No identity verification is required for function requests. Anonymous access is supported. Anyone can initiate an HTTP request to call your function.

## Configure InvokeFunctionUrl Policy Permissions

You can refer to the following steps to configure `InvokeFunctionUrl` policy permissions to open or restrict access to the interface.

1. On [the Policy page of the cloud access management console](#), click **Create Custom Policy** in the upper left corner.
2. In the pop - up window for selecting creation method, click **create by policy generator** to enter the edit strategy page.
3. In the **Visual Strategy Generator**, add a service and operation column, add the following information, and edit an authorization statement.

Effect (required): Select **Allow**.

Service (required): Select Serverless Cloud Function (scf).

Operation (required): Click **Expand** on the right of all operations (scf:\*), search for `InvokeFunctionUrl`, and check it. As shown below:

Resource (required): Select all resources or the specific resource you want to authorize.

Condition (optional): Set the effective condition for the above authorization.

4. Complete the policy authorization statement editing, then click **Next** to enter the basic information and associated users/groups/roles page.

5. On the associated users/groups/roles page, add the policy name and description, and you can simultaneously associate users/groups/roles for quick authorization.

6. Click **Finish** to complete the operation of creating a custom policy using the policy generator.

## Signature Generation and Authentication Process

### Client-Side Signature Generation

signature algorithm: please see Signature Method of Security Credential Service. Sample code is as follows:

Java

Go

NodeJS

Python

```
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class TencentCloudAPITC3Demo {
    private final static Charset UTF8 = StandardCharsets.UTF_8;
    // The environment variable TENCENTCLOUD_SECRET_ID needs to be set, with the va
    private final static String SECRET_ID = System.getenv("TENCENTCLOUD_SECRET_ID")
    // The environment variable TENCENTCLOUD_SECRET_KEY needs to be set, with the v
    private final static String SECRET_KEY = System.getenv("TENCENTCLOUD_SECRET_KEY")
    private final static String CT_JSON = "application/json";

    public static byte[] hmac256(byte[] key, String msg) throws Exception {
        Mac mac = Mac.getInstance("HmacSHA256");
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, mac.getAlgorithm());
        mac.init(secretKeySpec);
        return mac.doFinal(msg.getBytes(UTF8));
    }

    public static String sha256Hex(String s) throws Exception {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] d = md.digest(s.getBytes(UTF8));
        return DatatypeConverter.printHexBinary(d).toLowerCase();
    }
}
```

```

public static void main(String[] args) throws Exception {
    String service = "scf";
    String host = "1253970226-xxxxxxx-cq.scf.tencentcs.com";
    String uin = "xxxxxx"; // Replace with the actual uin to be used
    String algorithm = "TC3-HMAC-SHA256";
    String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    // Note the time zone, otherwise errors are likely to occur.
    sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
    String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

    // ***** Step 1: Construct standard request string *****
    String httpRequestMethod = "POST";
    String canonicalUri = "/";
    String canonicalQueryString = "";
    String canonicalHeaders = "content-type:application/json; charset=utf-8\n"
        + "host:" + host + "\n" ;
    String signedHeaders = "content-type;host";

    // Request body
    String payload = "{\"Limit\": 1, \"Filters\": [{\"Values\": [\"\\\\\\\\u
    String hashedRequestPayload = sha256Hex(payload);
    String canonicalRequest = httpRequestMethod + "\n" + canonicalUri + "\n"
        + canonicalHeaders + "\n" + signedHeaders + "\n" + hashedRequestP
    System.out.println(canonicalRequest);

    // ***** Step 2: Concatenate strings to be signed *****
    String credentialScope = date + "/" + service + "/" + "tc3_request";
    String hashedCanonicalRequest = sha256Hex(canonicalRequest);
    String stringToSign = algorithm + "\n" + timestamp + "\n" + credentialSco
    System.out.println(stringToSign);

    // ***** Step 3: Calculate the signature *****
    byte[] secretDate = hmac256(("TC3" + SECRET_KEY).getBytes(UTF8), date);
    byte[] secretService = hmac256(secretDate, service);
    byte[] secretSigning = hmac256(secretService, "tc3_request");
    String signature = DatatypeConverter.printHexBinary(hmac256(secretSigning,
    System.out.println(signature);

    // ***** Step 4: Concatenate Authorization *****
    String authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" +
        + "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signatur
    System.out.println(authorization);

    StringBuilder sb = new StringBuilder();
    sb.append("curl -X POST https://").append(host+canonicalUri)

```

```
.append(" -H \\\"Authorization: ").append(authorization).append("\\\\")
.append(" -H \\\"Content-Type: application/json; charset=utf-8\\")
.append(" -H \\\"Host: ").append(host).append("\\\\")
.append(" -H \\\"X-Scf-Cam-Uin: ").append(uin).append("\\\\")
.append(" -H \\\"X-Scf-Cam-Timestamp: ").append(timestamp).append("\\\\")
.append(" -d '").append(payload).append("'");
System.out.println(sb.toString());
}
}

package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "os"
    "time"
)

func sha256hex(s string) string {
    b := sha256.Sum256([]byte(s))
    return hex.EncodeToString(b[:])
}

func hmacsha256(s, key string) string {
    hashed := hmac.New(sha256.New, []byte(key))
    hashed.Write([]byte(s))
    return string(hashed.Sum(nil))
}

func main() {
    // need to fill in account UIN
    uin := "xxxx"
    // The environment variable TENCENTCLOUD_SECRET_ID needs to be set, with th
    secretId := os.Getenv("TENCENTCLOUD_SECRET_ID")
    // The environment variable TENCENTCLOUD_SECRET_KEY needs to be set, with t
    secretKey := os.Getenv("TENCENTCLOUD_SECRET_KEY")
    host := "1253970226-xxxxxxx-cq.scf.tencentcs.com"

    algorithm := "TC3-HMAC-SHA256"
    service := "scf"
    var timestamp int64 = time.Now().Unix()

    // step 1: build canonical request string
```

```

httpRequestMethod := "POST"
canonicalURI := "/"
canonicalQueryString := ""
canonicalHeaders := fmt.Sprintf("content-type:%s\\nhost:%s\\n",
    "application/json", host)
signedHeaders := "content-type;host"
payload := `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"]},
hashedRequestPayload := sha256hex(payload)
canonicalRequest := fmt.Sprintf("%s\\n%s\\n%s\\n%s\\n%s\\n%s",
    httpRequestMethod,
    canonicalURI,
    canonicalQueryString,
    canonicalHeaders,
    signedHeaders,
    hashedRequestPayload)
fmt.Println("canonicalRequest => ", canonicalRequest)

// step 2: build string to sign
date := time.Unix(timestamp, 0).UTC().Format("2006-01-02")
credentialScope := fmt.Sprintf("%s/%s/tc3_request", date, service)
hashedCanonicalRequest := sha256hex(canonicalRequest)
string2sign := fmt.Sprintf("%s\\n%d\\n%s\\n%s",
    algorithm,
    timestamp,
    credentialScope,
    hashedCanonicalRequest)
fmt.Println("string2sign ==>", string2sign)

// step 3: sign string
secretDate := hmacsha256(date, "TC3"+secretKey)
secretService := hmacsha256(service, secretDate)
secretSigning := hmacsha256("tc3_request", secretService)
signature := hex.EncodeToString([]byte(hmacsha256(string2sign, secretSignin
fmt.Println(signature)

// step 4: build authorization
authorization := fmt.Sprintf("%s Credential=%s/%s, SignedHeaders=%s, Signat
    algorithm,
    secretId,
    credentialScope,
    signedHeaders,
    signature)
fmt.Println(authorization)

curl := fmt.Sprintf(`curl -X %s https://%s -H "Authorization: %s" -H "Conte
    httpRequestMethod, host, authorization, host, uin, timestamp, paylo
fmt.Println(curl)

```

```
}  
  
const crypto = require('crypto');  
  
function sha256(message, secret = '', encoding) {  
    const hmac = crypto.createHmac('sha256', secret)  
    return hmac.update(message).digest(encoding)  
}  
  
function getHash(message, encoding = 'hex') {  
    const hash = crypto.createHash('sha256')  
    return hash.update(message).digest(encoding)  
}  
  
function getDate(timestamp) {  
    const date = new Date(timestamp * 1000)  
    const year = date.getUTCFullYear()  
    const month = ('0' + (date.getUTCMonth() + 1)).slice(-2)  
    const day = ('0' + date.getUTCDate()).slice(-2)  
    return `${year}-${month}-${day}`  
}  
  
function main(){  
    // key parameters  
    // The environment variable TENCENTCLOUD_SECRET_ID needs to be set, with the va  
    const SECRET_ID = process.env.TENCENTCLOUD_SECRET_ID  
    // The environment variable TENCENTCLOUD_SECRET_KEY needs to be set, with the v  
    const SECRET_KEY = process.env.TENCENTCLOUD_SECRET_KEY  
  
    const endpoint = "1253970226-xxxxxxx-cq.scf.tencentcs.com"  
    const service = "scf"  
    const timestamp = getTime()  
    // Time processing, get the world date and time  
    const date = getDate(timestamp)  
  
    // ***** Step 1: Construct standard request string *****  
    const payload = "{\"Limit\": 1, \"Filters\": [{\"Values\": [\"\\\\\\\\\\\\\\\\u672a\\n  
  
    const hashedRequestPayload = getHash(payload);  
    const httpRequestMethod = "POST"  
    const canonicalUri = "/"  
    const canonicalQueryString = ""  
    const canonicalHeaders = "content-type:application/json\\n"  
        + "host:" + endpoint + "\\n"  
    const signedHeaders = "content-type;host"
```

```

const canonicalRequest = httpRequestMethod + "\\n"
                        + canonicalUri + "\\n"
                        + canonicalQueryString + "\\n"
                        + canonicalHeaders + "\\n"
                        + signedHeaders + "\\n"
                        + hashedRequestPayload

console.log(canonicalRequest)

// ***** Step 2: Concatenate strings to be signed *****
const algorithm = "TC3-HMAC-SHA256"
const hashedCanonicalRequest = getHash(canonicalRequest);
const credentialScope = date + "/" + service + "/" + "tc3_request"
const stringToSign = algorithm + "\\n" +
                    timestamp + "\\n" +
                    credentialScope + "\\n" +
                    hashedCanonicalRequest
console.log(stringToSign)

// ***** Step 3: Calculate the signature *****
const kDate = sha256(date, 'TC3' + SECRET_KEY)
const kService = sha256(service, kDate)
const kSigning = sha256('tc3_request', kService)
const signature = sha256(stringToSign, kSigning, 'hex')
console.log(signature)

// ***** Step 4: Concatenate Authorization *****
const authorization = algorithm + " " +
                    "Credential=" + SECRET_ID + "/" + credentialScope + ", " +
                    "SignedHeaders=" + signedHeaders + ", " +
                    "Signature=" + signature
console.log(authorization)

const curlcmd = 'curl -X POST ' + "https://" + endpoint
                + ' -H "Authorization: ' + authorization + '"'
                + ' -H "Content-Type: application/json"'
                + ' -H "Host: ' + endpoint + '"'
                + ' -H "X-Scf-Cam-Uin: ' + uin + '"'
                + ' -H "X-Scf-Cam-Timestamp: ' + timestamp.toString() +
                + " -d '" + payload + "'"

console.log(curlcmd)
}
main()

# -*- coding: utf-8 -*-
import hashlib, hmac, json, os, sys, time
from datetime import datetime

```



```
# Key parameters
# The environment variable TENCENTCLOUD_SECRET_ID needs to be set, with the value b
secret_id = os.environ.get("TENCENTCLOUD_SECRET_ID")
# The environment variable TENCENTCLOUD_SECRET_KEY needs to be set, with the value
secret_key = os.environ.get("TENCENTCLOUD_SECRET_KEY")

service = "scf"
host = "1253970226-xxxxxxx-cq.scf.tencentcs.com"
endpoint = "https://" + host
algorithm = "TC3-HMAC-SHA256"
timestamp = int(time.time())
date = datetime.utcfromtimestamp(timestamp).strftime("%Y-%m-%d")
params = {"Limit": 1, "Filters": [{"Values": ["unnamed"], "Name": "instance-name"}]}

# ***** Step 1: Concatenate specification request strings *****
http_request_method = "POST"
canonical_uri = "/"
canonical_querystring = ""
ct = "application/json"
payload = json.dumps(params)
canonical_headers = "content-type:%s\\nhost:%s\\n" % (ct, host)
signed_headers = "content-type;host"
hashed_request_payload = hashlib.sha256(payload.encode("utf-8")).hexdigest()
canonical_request = (http_request_method + "\\n" +
                     canonical_uri + "\\n" +
                     canonical_querystring + "\\n" +
                     canonical_headers + "\\n" +
                     signed_headers + "\\n" +
                     hashed_request_payload)
print(canonical_request)

# ***** Step 2: Concatenate strings to be signed *****
credential_scope = date + "/" + service + "/" + "tc3_request"
hashed_canonical_request = hashlib.sha256(canonical_request.encode("utf-8")).hexdigest()
string_to_sign = (algorithm + "\\n" +
                  str(timestamp) + "\\n" +
                  credential_scope + "\\n" +
                  hashed_canonical_request)
print(string_to_sign)

# ***** Step 3 Calculate the signature *****
# Calculate signature digest function
def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
secret_date = sign(("TC3" + secret_key).encode("utf-8"), date)
```

```

secret_service = sign(secret_date, service)
secret_signing = sign(secret_service, "tc3_request")
signature = hmac.new(secret_signing, string_to_sign.encode("utf-8"), hashlib.sha256)
print(signature)

#*****Step 4: Concatenate Authorization *****
authorization = (algorithm + " " +
                 "Credential=" + secret_id + "/" + credential_scope + ", " +
                 "SignedHeaders=" + signed_headers + ", " +
                 "Signature=" + signature)
print(authorization)

print('curl -X POST ' + endpoint
      + ' -H "Authorization: ' + authorization + '"'
      + ' -H "Content-Type: application/json"'
      + ' -H "Host: ' + host + '"'
      + ' -H "X-Scf-Cam-Uin: ' + uin + '"'
      + ' -H "X-Scf-Cam-Timestamp: ' + str(timestamp) + '"'
      + " -d '" + payload + "'")

algorithm := "TC3-HMAC-SHA256"
service := "scf"
var timestamp int64 = time.Now().Unix()

// step 1: build canonical request string
httpRequestMethod := "POST"
canonicalURI := "/"
canonicalQueryString := ""
canonicalHeaders := fmt.Sprintf("content-type:%s\\nhost:%s\\n",
    "application/json", host)
signedHeaders := "content-type;host"
payload := `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"],
    "Name": "instance-name"}]}`
hashedRequestPayload := sha256hex(payload)
canonicalRequest := fmt.Sprintf("%s\\n%s\\n%s\\n%s\\n%s\\n%s",
    httpRequestMethod,
    canonicalURI,
    canonicalQueryString,
    canonicalHeaders,
    signedHeaders,
    hashedRequestPayload)
fmt.Println("canonicalRequest => ", canonicalRequest)

// step 2: build string to sign
date := time.Unix(timestamp, 0).UTC().Format("2006-01-02")
credentialScope := fmt.Sprintf("%s/%s/tc3_request", date, service)

```

```
hashedCanonicalRequest := sha256hex(canonicalRequest)
string2sign := fmt.Sprintf("%s\\n%d\\n%s\\n%s",
    algorithm,
    timestamp,
    credentialScope,
    hashedCanonicalRequest)
fmt.Println("string2sign ==>", string2sign)

// step 3: sign string
secretDate := hmacsha256(date, "TC3"+secretKey)
secretService := hmacsha256(service, secretDate)
secretSigning := hmacsha256("tc3_request", secretService)
signature := hex.EncodeToString([]byte(hmacsha256(string2sign, secretSigning)))
fmt.Println(signature)

// step 4: build authorization
authorization := fmt.Sprintf("%s Credential=%s/%s, SignedHeaders=%s,
Signature=%s",
    algorithm,
    secretId,
    credentialScope,
    signedHeaders,
    signature)
fmt.Println(authorization)

curl := fmt.Sprintf(`curl -X %s https://%s -H "Authorization: %s" -H "Content-
Type: application/json" -H "Host: %s" -H "X-Scf-Cam-Uin: %s" -H "X-Scf-Cam-
Timestamp: %d" -d '%s'`,
    httpRequestMethod, host, authorization, host, uin, timestamp, payload)
fmt.Println(curl)
```

## Client Call Parameters

The following parameters need to be added in the Header of the URL request.

Parameter	Description
Authorization	Signature-related parameters, required. Example: TC3-HMAC-SHA256 Credential=AKID*****/2019-02-25/scf/tc3_request, SignedHeaders=content-type;host;xxx, Signature=be4f67d323c78ab9acb7395e43c0dbcf822a9cfac32fea2449a7bc7726b77
X-Scf-Cam-Timestamp	Timestamp used for signature generation, required.
X-Scf-Cam-Uin	Root account Uin, required.

X-Scf-Cam-Token	If generating a signature using a temporary key, token information is required.
-----------------	---

### Server-Side Signature Verification

Server invocation of CAM service's signature and authentication, please see [Cloud Access Management](#).

# A Custom Domain Name

## Configuring a Custom Domain Name

Last updated : 2024-12-02 20:07:18

### Feature Overview

The function platform now supports the Custom Domain Name feature, allowing users to access your functions through a browser or other terminal devices.

By adding a custom domain name, you can point it to any function with the function URL enabled or use path mapping to point to multiple functions. Additionally, you can enable options like HTTPS protocol and WAF for the domain name to ensure its security.

### Implementation Principle of Accessing Functions via Custom Domain Name

The function platform assigns a CNAME to each account in each region, with the following format:

```
<appid>.<region>.tencentscf.com
```

For example, if your custom domain name is `test.com`, you add a DNS record with your DNS provider to point to this domain name and configure the path mapping to the specified function on the function platform. When users access the specified path of your custom domain name `test.com`, the DNS resolution guides them to the function platform through the CNAME, and the request is ultimately routed to the corresponding function via path mapping. The function processes the request and returns the result to the user.

### Prerequisites

When the public network services are provided in the Chinese Mainland, you need to register your domain name according to national laws and regulations before binding it to the service, ensuring that your users can access the service through your domain name. Custom domain names bound to functions in Hong Kong (China) and overseas regions do not require registration.

### Directions

## Step 1: Adding a Custom Domain Name

1. Log in to the [Function Service Console](#).

2. In the left sidebar, choose **Custom Domain**, then click **Create custom domain**, as shown in the figure below:

### Note:

Domain names have regional attributes and can only point to functions within the same region.

**Custom Domain** Chongqing

1

- The function URL is a dedicated HTTP(S) endpoint for the function. After configuring the function URL, you can use it to call your function through curl, P URL does not support opening pages directly in a browser. If you want your users to be able to access the function through a browser, you can achieve th
- In mainland China, when providing services to the public network, in accordance with relevant national laws and regulations, you need to first complete th ensure that your users can access the service through your domain name.
- After adding a custom domain, you can point the domain to any function that has the function URL enabled, or you can point to multiple functions throug

1 **Obtain a domain**  
Go to [Domain Registration](#), or obtain a domain from another service provider

2 **ICP Filing**  
Register the domain name with the domain registrar. The process can be referred to [Website Filing](#)

3 **CNAME to second-level domain**  
Add a CNAME record to poin the domain to the service's second-level domain

**Create Custom Domain** 2

**scf-test-XXXXX.com**

HTTPS Enabled

Force HTTPS Enabled

SSL Certificate [\[Link\]](#)

Web Application Firewall waf-XXXXX [\[Link\]](#)

Path Mapping

Path	Function	Rewrite Policy
/	default/helloworld-XXXXX/\$DEFAULT	Not enabled
/test/*	default/helloworld-XXXXX/\$DEFAULT	/test/* -> /\$1

3. In **Create custom domain**, enter the custom domain name that has been registered. Single domain names, such as `test.com`, are supported, while wildcard domain names, such as `*.test.com`, are not supported at this time. As shown in the figure below:

### Create Custom Domain

**i** Please confirm that the domain name you wish to bind has completed DNS configuration, pointing to the following second-level domain. Ensure that your domain name has been registered.  
Public Network CNAME Domain: 1253[REDACTED].ap-chongqing.tencentscf.com  
Private Network CNAME Domain: 125[REDACTED].in.ap-chongqing.tencentscf.com

Domain

After modifying the CNAME record, it takes more than TTL time to take effect. Please ensure that it takes effect before proceeding with the configuration operation. Refer to [Official Website Documentation](#).

4. In the pop-up window, obtain the **Public Network CNAME** or **Private Network CNAME**, then go to your DNS resolution platform and map your custom domain name to this CNAME. After confirming the resolution is complete, proceed to the next step.

### Create Custom Domain

**i** Please confirm that the domain name you wish to bind has completed DNS configuration, pointing to the following second-level domain. Ensure that your domain name has been registered.  
Public Network CNAME Domain: 1253[REDACTED].ap-chongqing.tencentscf.com  
Private Network CNAME Domain: 125[REDACTED].in.ap-chongqing.tencentscf.com

Domain

After modifying the CNAME record, it takes more than TTL time to take effect. Please ensure that it takes effect before proceeding with the configuration operation. Refer to [Official Website Documentation](#).

The CNAME formats are as follows:

Public Network CNAME: `<appid>.<region>.tencentscf.com` Example: `123456.ap-guangzhou.tencentscf.com`

Private Network CNAME: `<appid>.in.<region>.tencentscf.com` Example: `123456.in.ap-guangzhou.tencentscf.com`

## Step 2: Adding Path Mapping

You can map different paths to different functions, allowing different request paths to trigger the execution of different functions. By default, you only need to configure the root path (/) to map to the specified version or alias of a function.

### Create Custom Domain

**i** Please confirm that the domain name you wish to bind has completed DNS configuration, pointing to the following second-level domain. Ensure that your domain name has been registered.  
Public Network CNAME Domain: 12539[redacted].ap-chongqing.tencentscf.com  
Private Network CNAME Domain: 12539[redacted].in.ap-chongqing.tencentscf.com

Domain

After modifying the CNAME record, it takes more than TTL time to take effect. Please ensure that it takes effect before proceeding with the configuration operation. Refer to [Official Website Documentation](#).

HTTPS

☐ Enable

Web Application Firewall

☐ Enable

SCF integrates web application firewall services, supporting protection against BOTs, crawlers, malicious registrations, etc., effectively ensuring the secure and stable operation of the business. This capability will incur certain fees. Refer to [official website documentation](#).

Path Mapping

Path	Function	Rewrite Policy	O...
<input type="text" value="/"/>	<div>default ▾ Alias: Default tr ▾</div>	<div>helloworld-172 ▾</div>	<div><input type="checkbox"/> Enable / -&gt; / <b>i</b> <b>X</b></div>

[Add](#)

## Path Mapping Matching Rules

### Exact Path:

The requested path needs to match the configured path exactly in order to trigger the corresponding function.

For example, if the configured path is /a/, with the namespace n1, the function f1, and version 1, then only requests from the path /a/ will trigger version 1 of the f1 function. Requests from /a or /a/b will not trigger version 1 of the f1 function.

For example, if the configured path is /a, with the namespace n1, the function f1, and version 1, then only requests from the path /a will trigger version 1 of the f1 function. Requests from /a/ will not trigger version 1 of the f1 function.

### Fuzzy Path:

The wildcard (\*) can be used to set the path, but it can only be placed at the end of the path; the longest prefix matching principle applies.



For example, if the configured path is `/xxx/*`, with the namespace `n2`, the function `f2`, and version `1`, then any request with the path prefix `/xxx/` (such as `/xxx/a` or `/xxx/b/c/d`) will trigger version `1` of the `f2` function.

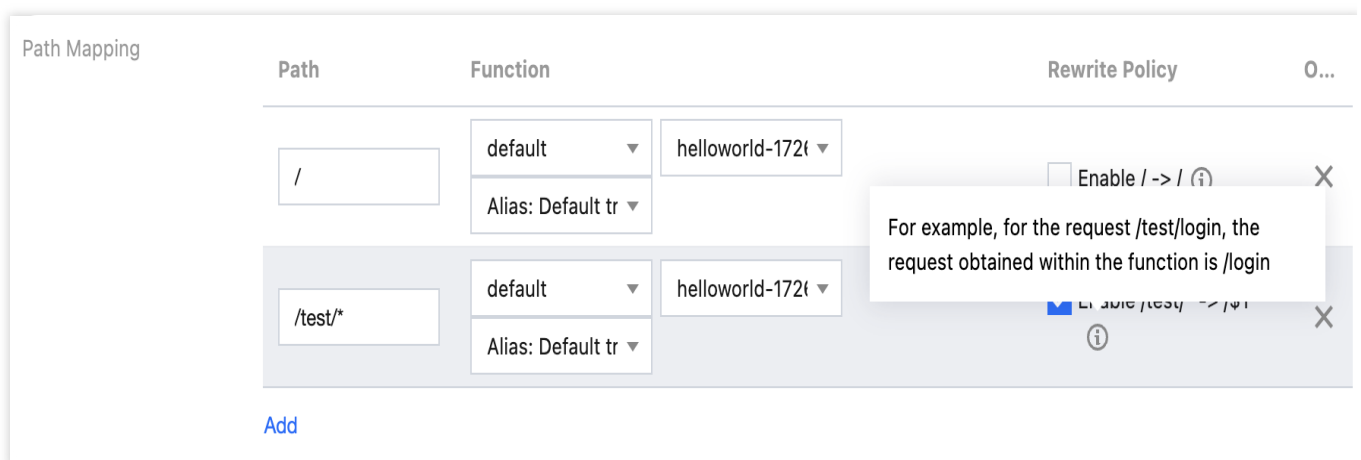
### Path Rewrite Policy

When you point a sub-path to a function, the path in the request received by the function will include that sub-path by default. For example, the behavior in the following two scenes is:

1. When the exact path `/home` points to function `a`, and `abc.test.com/home` is accessed, the path in the request received by function `a` will be `/home`.
2. When the fuzzy path `/test/*` points to function `b`, and `abc.test.com/test/login` is accessed, the path in the request received by function `b` will be `/test/login`.


If you want the function to receive requests without the sub-path, you can enable the Rewrite Policy. After the Rewrite Policy is enabled, the behavior for the above two scenes is:

1. When the exact path `/home` points to function `a`, and `abc.test.com/home` is accessed, the path in the request received by function `a` will be `/`.
2. When the fuzzy path `/test/*` points to function `b`, and `abc.test.com/test/login` is accessed, the path in the request received by function `b` will be `/login`.



### Step 3: (Optional) HTTPS Settings

You can also choose to enable the **HTTPS** protocol for accessing the custom domain name.

HTTPS	<input checked="" type="checkbox"/> Enable
Force HTTPS	<input type="checkbox"/> Enable
After being enabled, HTTP requests will be redirected to HTTPS via 301	
SSL Certificate	<div><div>Hz4[REDACTED]</div><div>▼</div></div> <div> <a href="#">Upload Certificate</a></div>
This certificate list only displays certificates available for matching domains	

After enabling it, you need to select a certificate that matches the domain name from the SSL Certificate Service to complete the configuration. If the certificate dropdown list is empty, it indicates you do not yet have a matching certificate for the domain name in the SSL Certificate Service, and you should first go to the SSL Certificate Service to upload or purchase a certificate.

Once the configuration is complete, the domain name can be accessed via either the HTTP or HTTPS protocol. You can also enable the **Force HTTPS** option. Once it is enabled, only HTTPS protocol access to the domain name is supported, and all HTTP access requests will be redirected to HTTPS with a 301 status code.

#### Step 4. (Optional) WAF Settings

You can also choose to enable WAF, which provides protection against bot attacks, crawlers, and malicious registrations, effectively ensuring the secure and stable operation of your business. This feature will incur some fees. For details, see [WAF](#).

The configuration steps are as follows:

1. Log in to the [Tencent Cloud Console](#).
2. Go to the [WAF purchase page](#).
3. On the WAF purchase page, select CLB Instance as shown in the figure below:

## Web Application Firewall

[Back to Product Details](#)

WAF products cannot be refunded. All domain names using WAF services must go through ICP filing to ensure compliance and normal access. [Obtain ICP Filing](#)

### Plan

#### Select configuration

Instance type

SaaS

CLB

[View comparisons](#)

CLB WAF is suitable for applications that have been deployed on Tencent Cloud and are using or planning to use [Load balancing](#). CLB WAF [Supported regions](#)

Country/Region

Chinese mainland

Outside the Chinese mainland

Plan

Item

Premium

Applicable to medium and small-sized non-operational websites

Enterprise

Applicable to medium and small-scaled operational websites and large portal websites that require custom protection configurations

Ultimate

Applicable to large and ultra large operational websites and complex websites that require custom protection configurations

4. After completing the purchase, return to the SCF console, go to the Custom Domain Name configuration page, and select **Enable WAF**. Then, select the corresponding instance ID from the WAF instance dropdown list.

Web Application Firewall

☒ Enable

SCF integrates web application firewall services, supporting protection against BOTs, crawlers, malicious registrations, et effectively ensuring the secure and stable operation of the business. This capability will incur certain fees. Refer to [official website documentation](#).

WAF Instance

waf\_2k...

[Create waf](#)

## Step 5. Verifying the Custom Domain Name

You can choose to test through a browser or by using the curl command in the command line. After accessing the custom domain name, verify whether the specified function has been called.

### Browser Test:

- Open a browser and enter your custom domain name, such as <https://abc.test.com>.
- Check whether the specified function is correctly called and returns the expected result.

### Command Line curl Test:

```
curl -v https://abc.test.com
```

Check the output to verify whether the specified function is correctly called and returns the expected result.

# Version Management

## Overview

Last updated : 2025-02-12 11:10:42

## Overview

An SCF function version contains function code and configuration. In the actual development process, you can minimize the impact on your business systems by publishing a version with fixed function code and configuration.

## Related Concepts

### Latest version (\$LATEST)

Once created, a SCF has a latest version (\$LATEST) by default. Only the configuration and code of the \$LATEST version can be modified, and based on which, a new version is generated at the time of release.

## Relevant Operations

Operations that can be performed on versions include:

[View version](#)

[Publish version](#)

[Use version](#)

# Viewing a Version

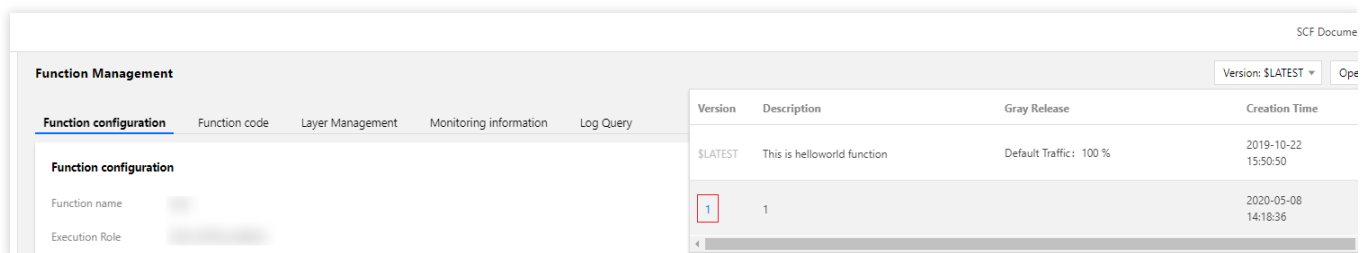
Last updated : 2025-02-12 11:10:43

## Operation Scenarios

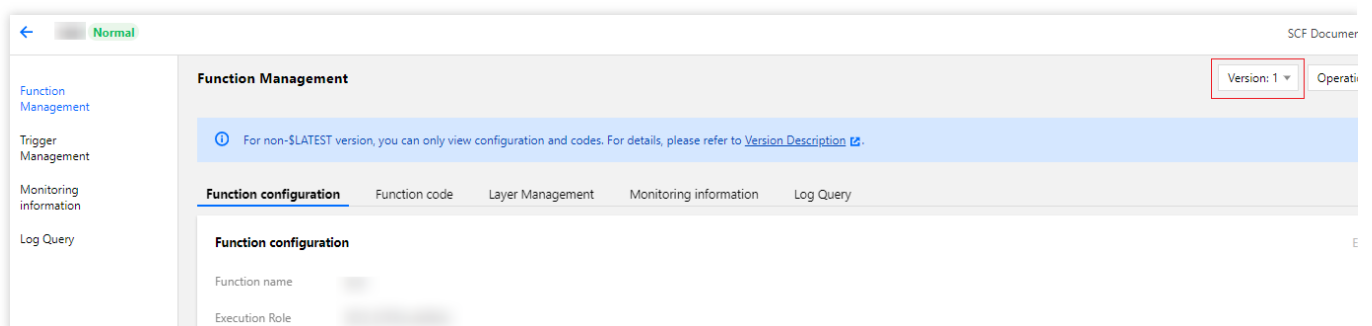
This document describes how to view the version configuration, code, and other information of a specified function.

## Directions

1. Log in to the SCF Console and select **Function Service** on the left sidebar.
2. At the top of the "Function Service" page, select the region and namespace where the function to be viewed resides:
3. Click a function name to enter the function details page.
4. Select the "Version" drop-down list in the top-right corner on the function details page and click the name of the version you want to view. This document takes viewing version **1** as an example as shown below:



Here, you can view version information as shown below:



**Note:**

After you switch to the desired version, the **Function Configuration**, **Function Code**, **Layer Management**, **Monitoring Info**, and **Log Query** tabs will display contents of the corresponding version. For more information on the tabs, please see [Querying Functions](#).

After you switch to a non- `$LATEST` version, the function configuration and code will remain in the state upon release and cannot be modified.

Triggers can be configured differently for different versions.

The logs and monitoring information respectively display the specific invocation logs and monitoring data of the corresponding version.

# Releasing a Version

Last updated : 2025-02-12 11:10:43

## Operation Scenarios

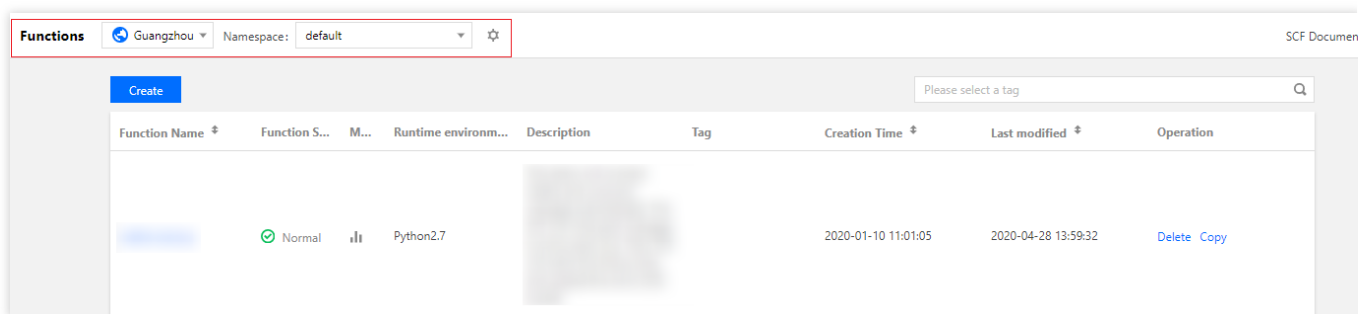
After configuring the function, submitting the code, and passing the online test, you can fix the version of the function by publishing the version to avoid errors or execution failures of your online business caused by subsequent modification and testing of the code. You can publish a version at any time, and in all releases, the `$LATEST` version is always the latest version.

## Directions

### Note:

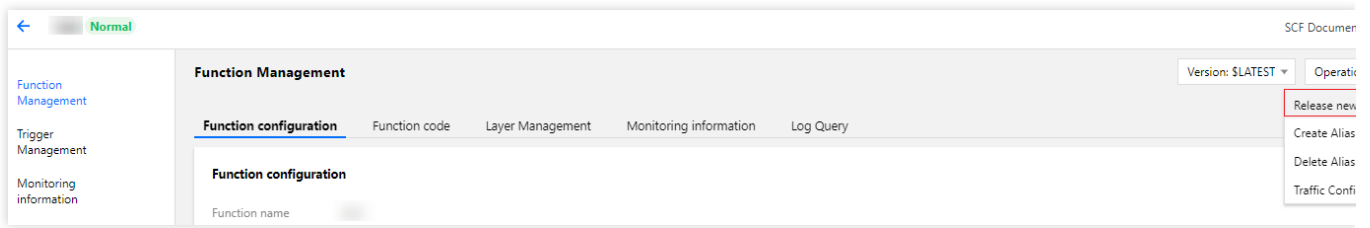
A function has the `$LATEST` version attribute upon its creation. The `$LATEST` version points to the currently editable version and is always present and editable.

1. Log in to the SCF Console and select **Function Service** on the left sidebar.
2. At the top of the "Function Service" page, select the region and namespace where the function to be viewed resides as shown below:



3. Click the function name to enter the function information page.
4. Select **Operation** > **Publish New Version** in the top-right corner of the function information page as shown below:





5. In the pop-up "Publish New Version" window, enter the version description and click **Submit** to publish the version as shown below:

### Release new version

**i** The new version will be generated using the configs and codes of \$LATEST version. For details, please refer to [Version Description](#).

Function name

test

Description \*

Please enter a description of the version.

Up to 1000 letters, numbers, spaces, commas, dots.

Submit

Close

#### Note:

When the release is submitted, the SCF platform will generate a copy of the configuration and code content of the current `$LATEST` version of the function and save it as version content.

After a version is published, the version number of the release will be generated, which starts from 1 and increases progressively with each release. The version number has no upper limit.

The published version only records and fixes the configuration and code of the current `$LATEST` version of the function but not the trigger configuration of the function. A newly published function version does not have any triggers.

# Using a Version

Last updated : 2025-02-12 11:10:43

The version feature is mainly used to fix the function configuration and code and avoid the impact of development, debugging, and testing on the business. After [publishing a function version](#), you can use the version by invoking the function on the specified version.

**Note:**

The `$LATEST` version is used for code development and testing.

## Version trigger

Currently, all published versions of functions can be independently bound to triggers. For the same function, different versions are independent of one another, and each trigger can independently trigger the function.

**Note:**

There is a limit to the number of triggers under one user account. For more information, please see [Quota Limits](#). To increase the maximum number of triggers (i.e., quota limit), you can [submit a ticket](#) for application.

## TencentCloud API-Triggered Versions

When the `InvokeFunction` API of TencentCloud API is used to trigger a function invocation, the specific version to be triggered can be specified with the optional parameter `Qualifier`. Without this parameter, the `$DEFAULT` alias will be triggered by default.

# Alias Management

## Traffic Routing Configuration

Last updated : 2025-02-12 11:16:17

### Operation Scenarios

Serverless Cloud Function (SCF) allows you to configure traffic routing. With this feature, you can easily control the beta launch or rollback process in actual use cases or environments in order to avoid the risks that may be brought by direct launch.

When creating an alias or adjusting the traffic configuration, you can point the traffic to two function versions in the console, so that the traffic can be routed between versions by the specified rule. Currently, two routing schemes are supported, namely, **random routing by weight** and **routing by rule**.

If you want to randomly route requests to any two versions by the specified weight in percentages, you can configure [random routing by weight](#).

If you want to route requests containing certain content to the specified version, you can configure [routing by rule](#).

### Directions

#### Random routing by weight

This document uses configuration during alias creation as an example to describe how to configure routing. After the alias is created, traffic will be randomly routed to two versions by the specified percentages. The steps are as follows:

1. Enter the "Create Alias" window.
2. In the "Create Alias" window, configure traffic routing based on the information below:

### Create Alias

Alias Name

Please enter the alias name

1. 2 to 60 characters  
2. Starts with a letter and ends with a number or letter; supports a-z, A-Z, 0-9, -, \_.

Description

Please enter description of the alias

Up to 1000 letters, numbers, spaces, commas, dots.

Routing Method

By weight

By rules

Version Weight Configurations

\$LATEST

100

%

Please select a version

0

%

Submit

Close

Main parameters include:

**Routing Method:** select **By weight**.

**Version Weight Configurations:** you can select two versions in the drop-down lists and set their weights in percentage.

3. Click **Submit** to complete the settings.

## Routing by rule

The current rule syntax for routing by rule consists of the following three parts:

### Match key

It is the position of the value to be read during the match and is used to determine whether the rule is hit.

The key should be in the format of `invoke.headers.[userKey]` where `[userKey]` can be modified. This format indicates that when the `invoke` API is called, the `userKey` part in `headers` of the HTTP request will be used for match.

### Match method

During the match, the expression is compared in a specified method. Currently, the `exact` and `range` match methods are supported.

`exact` : exact match. When the `exact` method is used, the match expression should be a string. If the value read through the match key is exactly the same as the expression, the rule will be hit.

`range` : range match. When the `range` method is used, the match expression should be in the format of `(a,b)` or `[a,b]` where `a` and `b` need to be integers. If the value read through the match key is an integer in the range specified by the expression, the rule will be hit.

### Match expression

If the specified value is matched, it will be considered as a hit. For more information on how to compose an expression, please see the description in [Match Method](#).

You can configure routing by rule in the following steps:

1. Enter the "Create Alias" window.
2. In the "Create Alias" window, configure traffic routing and click **Submit** to complete the configuration as shown below:

### Create Alias

Alias Name

Please enter the alias name

1. 2 to 60 characters  
2. Starts with a letter and ends with a number or letter; supports a-z, A-Z, 0-9, -, .

Description

Please enter description of the alias

Up to 1000 letters, numbers, spaces, commas, dots.

Routing Method

By weight

By rules

Version Rule Configurations

Please select a version

exact

Please [enter](#) "invoke.headers.User" exact "testuser", and when invoke the input RoutingKey:{"User":"testuser"}

\$LATEST

Use this version when no rules are hit

Submit

Close

Main parameters include:

**Routing Method:** select **By rules**.

**Version Rule Configurations:** set the fields as needed based on the following example:

For example, if you have two versions (version 2 and version 1) and want to set the match rule of version 2 to `invoke.headers.User exact Bob` and set that of version 1 to "miss", you can set the fields as shown below:

### Create Alias

Alias Name

Please enter the alias name

1. 2 to 60 characters  
2. Starts with a letter and ends with a number or letter; supports a-z, A-Z, 0-9, -,

Description

Please enter description of the alias

Up to 1000 letters, numbers, spaces, commas, dots.

Routing Method

By weight

By rules

Version Rule Configurations

2

invoke.headers.User

exact

Bob

Please enter "invoke.headers.User" exact "testuser", and when invoke the input RoutingKey:{"User":"testuser"}

1

Use this version when no rules are hit

Submit

Close

According to the configuration, when SCF invokes the function alias through the `invoke` API, if the `routingKey` parameter is set to `{"User": "Bob"}`, version 2's code and configuration will be used during this execution; if the `routingKey` parameter is not set or set to another value, version 1's code and configuration will be used.

For example, if you have two versions (version 3 and version 2) and want to set the match rule of version 3 to `invoke.headers.userHash range [1, 50]` and set that of version 2 to "miss", you can set the fields as shown below:

### Create Alias

Alias Name

Please enter the alias name

1. 2 to 60 characters  
2. Starts with a letter and ends with a number or letter; supports a-z, A-Z, 0-9, -,

Description

Please enter description of the alias

Up to 1000 letters, numbers, spaces, commas, dots.

Routing Method

By weight

By rules

Version Rule Configurations

3

invoke.headers.userHash

range

[1,50]

Please enter "invoke.headers.User" exact "testuser", and when invoke the input RoutingKey:{"User":"testuser"}

2

Use this version when no rules are hit

Submit

Close

According to the configuration, when SCF invokes the function alias through the `invoke` API, if the `routingKey` parameter is set to `{"userHash":30}`, version 3's code and configuration will be used during this execution; if the `routingKey` parameter is not set or set to another value such as `{"userHash":80}`, version 2's code and configuration will be used.



# Using Alias to Implement SCF Grayscale Release

Last updated : 2025-02-12 11:14:33

## Overview

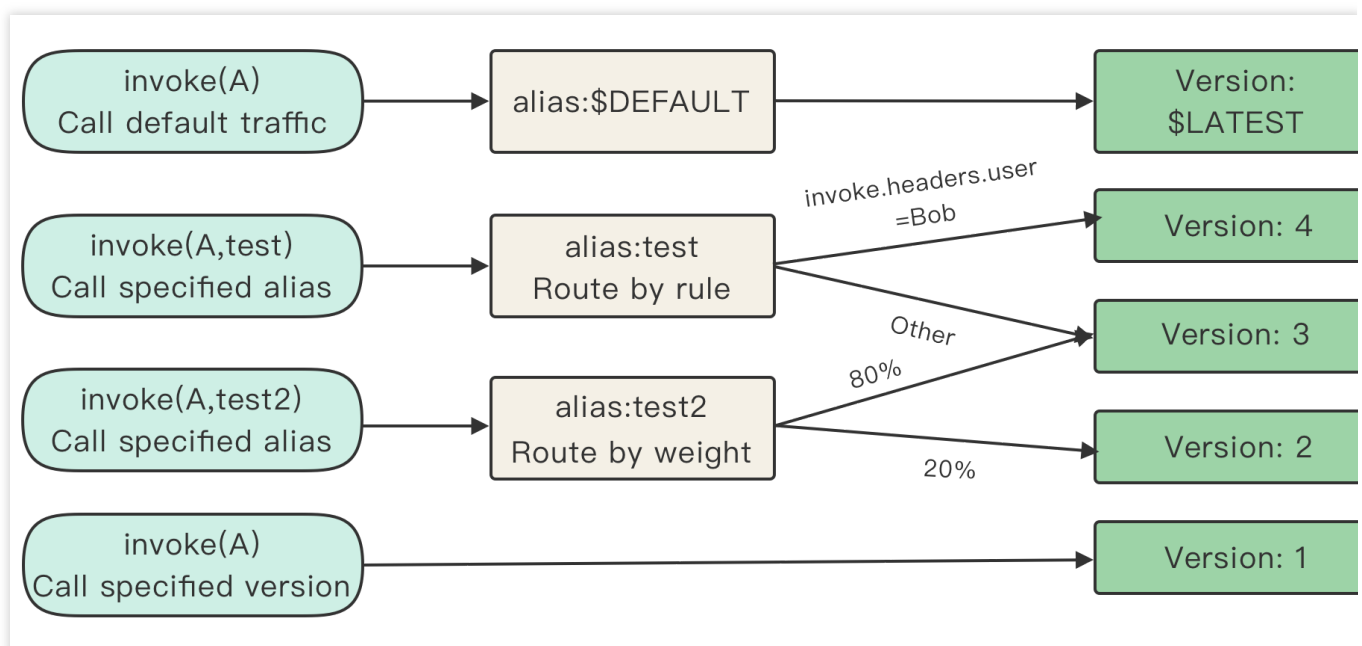
You can use the alias of an SCF function to implement the grayscale release scheme for the function, which has the following advantages:

You can distribute traffic among multiple versions as needed, with no need to frequently modify the settings externally or at each trigger location.

The traffic can be distributed smoothly to avoid traffic miss.

By using the same traffic switch scheme, you can quickly roll back the version when a failure occurs.

The following is the scheme diagram:



## Glossary

### Function

It refers to an SCF function you create.

### Version

An SCF function version contains the code and function configuration information and is specified by a specific version number. You can generate a specific version and version number through release. Code and configuration only on the latest version can be modified, but all versions can be called. For more information on the SCF function version, please see [Version Management Overview](#).

### **Latest version (\$LATEST)**

It refers to the version whose code and configuration can be modified. After a function is created, it will have the latest version by default. When the function is published, it will be published on the latest version with the specific version number.

### **Alias**

An alias can have a custom name starting with a letter. It is a reference that can be configured to point to one or two versions. If it points to two versions, the traffic percentages can be set for them. All aliases can be called.

### **Default traffic/default alias (\$DEFAULT)**

It is a special alias. If a call request does not point to any version or another alias, the default alias will be used to point to the latest version by default. The version pointed to can be modified.

## Scheme Use Cases

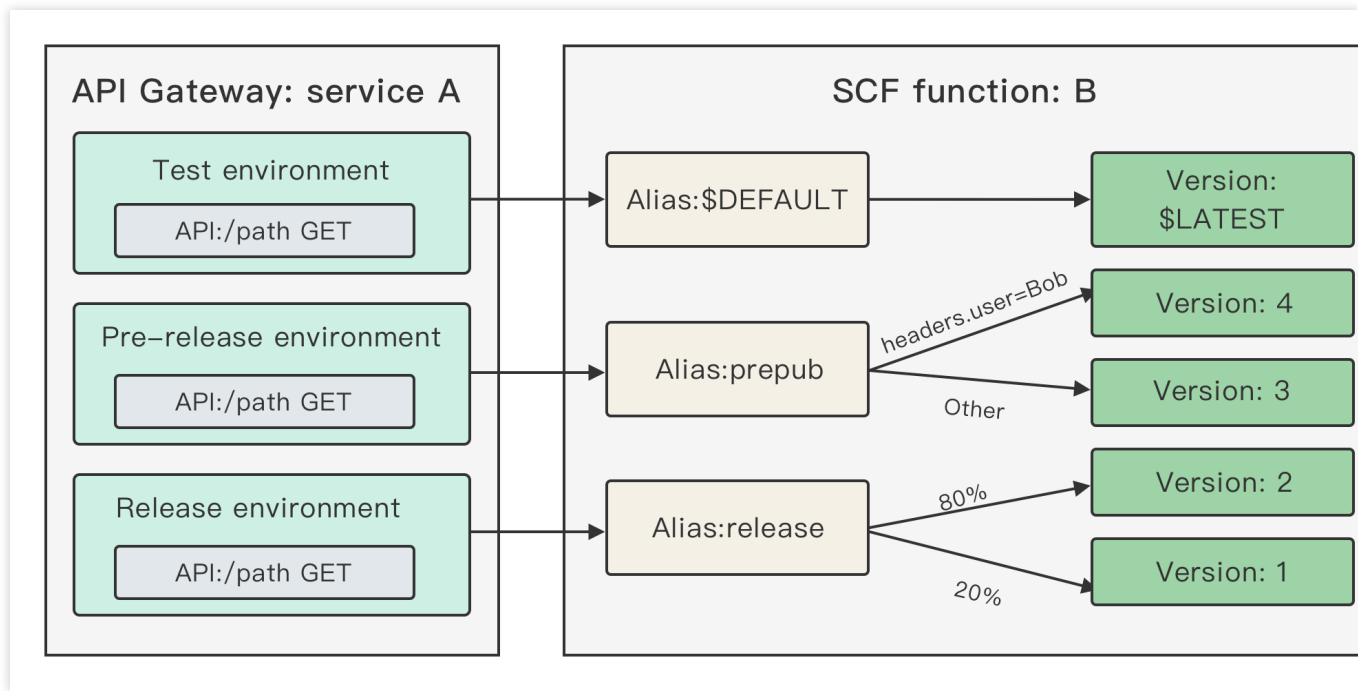
### **Use case based on API Gateway trigger**

#### **Background**

You have [created an SCF function](#), but have not published its new version or created an alias.

You want to have separate test, pre-release, and release environments, where the SCF function can enter the next stage only after completing the test at the current stage. In addition, you want to implement grayscale release to ensure that your function can be launched smoothly.

The following is the diagram of the overall scheme:



## Initial configuration

### 1. Create an alias:

Create aliases `release` and `prepub` in SCF function B, which temporarily point to the `$LATEST` version.

### 2. Create an API gateway:

Create API service A in API Gateway, configure the API to point to the `release` alias of function B, and publish it to the `release` stage of the API service. For more information on how to create and publish an API, please see [API Creation](#) and [API Release](#).

### 3. Modify API configuration:

- Point the API to the `prepub` alias of function B and publish it into the `prepub` stage of the API service.
- Point the API to the default traffic of function B and publish it into the `dev` stage of the API service.

At this point, the test, pre-release, and release environments have been separated, but they all point to the `$LATEST` version. API Gateway configuration is completed, and there will be no need to modify and publish the API Gateway configuration again.

## Continuous development, test, release, and launch

### 1. Publish versions:

In SCF, continuously develop and publish versions 1, 2, 3, and 4 in sequence. Suppose version 1 is in the release environment, version 2 is tested in the pre-release environment, and versions 3 and 4 are tested in the test environment.

### 2. Develop the latest version that needs to be tested:

By configuring the `$DEFAULT` alias to point it to the `$LATEST` version, you can perform continuous development

based on this version. After development is completed, the version can be published.

### 3. Test in the pre-release environment:

When version 3 can enter the pre-release environment, if the `prepub` alias of function B is configured to point to version 3, version 3 can be tested and trialed in the pre-release environment.

### 4. Implement grayscale release by user in the pre-release environment:

If version 4 can enter the pre-release environment, calls of user Bob need to be routed to version 4 of function B, calls of other users need to be routed to version 3, and the `prepub` alias of function B needs to be configured for routing by rule with the content `invoke.headers.User exact Bob`. For more information on routing by rule, please see [Routing by Rule](#).

### 5. Enter the release environment from the pre-release environment and implement grayscale release:

If version 2 has been trialed in the pre-release environment and can be launched, gradually switch the traffic configuration of the `release` alias of function B from version 1 to 2 and continuously observe the function status during the grayscale release. For more information on how to configure alias traffic, please see [SCF Traffic Routing Configuration](#).

### 6. Continuously monitor during release:

View the grayscale release process through monitoring information and logs and check whether the traffic of version 2/1 increases/decreases normally, errors of each version during the release, and overall errors.

### Prompt rollback upon failure during release

Suppose a failure occurs when version 2 is launched. In this case, you need to roll back to the previous version by modifying the `release` alias of function B to point all of its traffic to version 1.

## Use case with invoke API

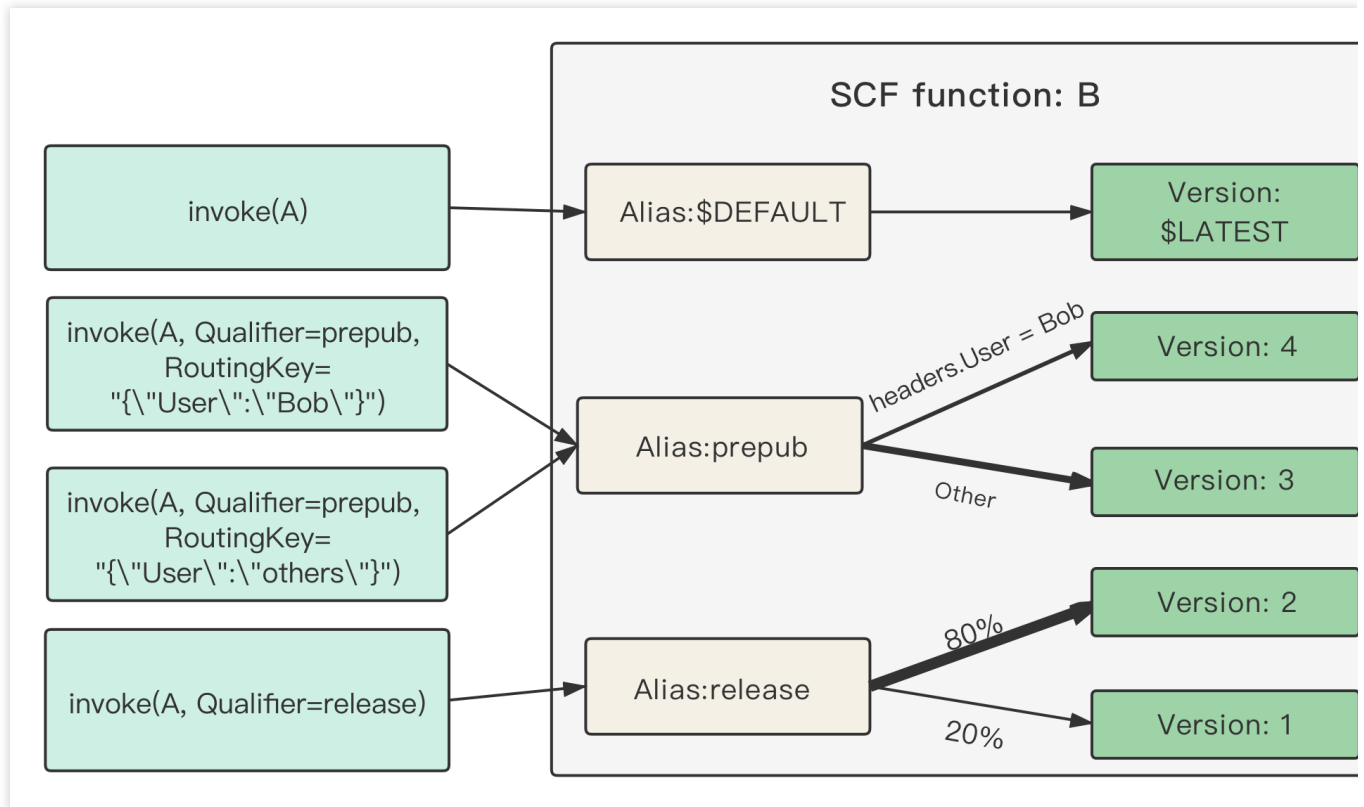
### Background

You have [created an SCF function](#), but have not published its new version or created an alias.

You need to directly use an API or the SDK to run SCF functions.

You want to have separate test, pre-release, and release environments, where the SCF function can enter the next stage only after completing the test at the current stage. In addition, you want to implement grayscale release to ensure that your function can be launched smoothly.

The following is the diagram of the overall scheme:



## Initial configuration

Create aliases `release` and `prepub` in SCF function B, which temporarily point to the `$LATEST` version.

## Continuous development, test, release, and launch

### 1. Publish versions:

In SCF, continuously develop and publish versions 1, 2, 3, and 4 in sequence. Suppose version 1 is in the release environment, version 2 is tested in the pre-release environment, and versions 3 and 4 are tested in the test environment.

### 2. Develop the latest version that needs to be tested:

By configuring the `$DEFAULT` alias to point it to the `$LATEST` version, you can perform continuous development based on this version. After development is completed, the version can be published.

### 3. Test in the pre-release environment:

When version 3 can enter the pre-release environment, if the `prepub` alias of function B is configured to point to version 3, version 3 can be tested and trialed in the pre-release environment.

### 4. Route by rule in the pre-release environment:

If version 4 can enter the pre-release environment, you can configure routing by rule for function B. Customize the `key` and `value` passed in to point them to version 4. When calling the `invoke` API, retain the key-value pair in the `RoutingKey` input parameter in `json` format. If `RoutingKey` has any key-value pair matching the rule, version 4 will be routed to. For more information on routing by rule, please see [Routing by Rule](#) and [Running](#)

[Function Through API.](#)

### 5. Enter the release environment from the pre-release environment and implement grayscale release:

If version 2 has been trialed in the pre-release environment and can be launched, gradually switch the traffic configuration of the `release` alias of function B from version 1 to 2 and continuously observe the function status during the grayscale release. For more information on how to configure alias traffic, please see [SCF Traffic Routing Configuration](#).

### 6. Continuously monitor during release:

View the grayscale release process through monitoring information and logs and check whether the traffic of version 2/1 increases/decreases normally, errors of each version during the release, and overall errors.

### Prompt rollback upon failure during release

Suppose a failure occurs when version 2 is launched. In this case, you need to roll back to the previous version by modifying the `release` alias of function B to point all of its traffic to version 1.

## Serverless Cloud Framework use case

When using Serverless Cloud Framework, you can use stages to distinguish between the test, pre-release, and release environments. When implementing grayscale release in the release environment, you can use the following commands for gradual switch. For detailed directions, please see [Using tencent-express Component to Deploy Express Website](#).

```
scf deploy --inputs.traffic=0.1 # Deploy and switch 10% traffic to the `latest` ve
scf deploy --inputs.traffic=1.0 # Deploy and switch 100% traffic to the `latest` v
scf deploy --inputs.traffic=1.0 # Deploy and switch 100% traffic to the `latest` v
```

# Permission Management

## Permission Management Overview

Last updated : 2024-12-02 20:11:41

### Overview

SCF uses [Tencent Cloud Access Management \(CAM\)](#) to manage permissions. CAM is a permission and access management service that helps you securely manage the access permissions to resources under your Tencent Cloud account. With CAM, you can create, manage and destroy users and user groups and use identity and policy management to control user access to Tencent Cloud resources.

### Manageable Permissions for SCF

You can assign different SCF permissions to sub-accounts or collaborators through your root account. Currently, SCF supports the following permission granularities:

Service	Policy Syntax	TencentCloud API	Console	Authorization Granularity	Temporary Credentials
SCF	✓	✓	✓	Resource level	✓

Currently, SCF supports the following TencentCloud APIs:

API Name	Description	Level
ListFunctions	Gets the function list under the account	Account
GetAccountSettings	Gets the quota configuration under the account	Account
CreateFunction	Creates a function	Resource
DeleteFunction	Deletes a specified function	Resource
InvokeFunction	Triggers a function synchronously or asynchronously	Resource
UpdateFunction	Updates a function, including its configuration and/or code	Resource
SetTrigger	Configures a trigger for a specified function	Resource
DeleteTrigger	Deletes a trigger for a specified function	Resource

GetFunction	Gets the configuration information of a specified function	Resource
ListVersion	Gets the version information of a specified function	Resource
GetFunctionLogs	Gets the log information of a specified function	Resource

## Roles and Authorization

SCF implements the access between services and user resources by using the role capability of CAM. SCF offers the **configuration role** and the **execution role**. You can use the configuration role to enable SCF to access user resources in the configuration process. You can also use the execution role to enable SCF to apply for the temporary authorization for executing the code, so that the code can implement permission and resource access through the role authorization mechanism.



# Role and Authorization

Last updated : 2024-12-02 20:11:42

## Operation Scenarios

[Role](#) is a virtual identity with a set of permissions provided by [CAM](#), which is mainly used to grant access permissions of services, operations, and resources in Tencent Cloud to role entities. After these permissions are added to a role, the role can be configured to Tencent Cloud services, allowing the services to perform operations on authorized resources on your behalf.

When creating an SCF function, you may need the permissions to manipulate other Tencent Cloud services.

Examples include COS permissions to create and delete COS triggers, API Gateway permissions to create and delete API Gateway triggers, and COS permissions to read zipped code packages.

## Role Details

Role name: `SCF_QcsRole`

Role entity: `service-scf.qcloud.com`

Role description: default configuration role of SCF. This service role is used to grant the SCF configuration the permissions to connect with other resources in the cloud, including but not limited to code file access and trigger configuration. The preset policy of the configuration role can support basic operations of function execution.

Role policy: this role has the `QcloudAccessForScfRole` policy that can:

Write trigger configuration information to the bucket configuration if a COS trigger is configured.

Read the trigger configuration information from the COS bucket.

Read the code zip package from the bucket when the code is updated through COS.

Create API Gateway services and APIs and publish services if an API Gateway trigger is configured.

### Note:

You can log in to the [CAM Console](#) to view and modify the policy associated with the current configuration role

`SCF_QcsRole`; however, modifying the associated policy of the role may cause SCF to fail; therefore, you are not recommended to modify it.

## Directions

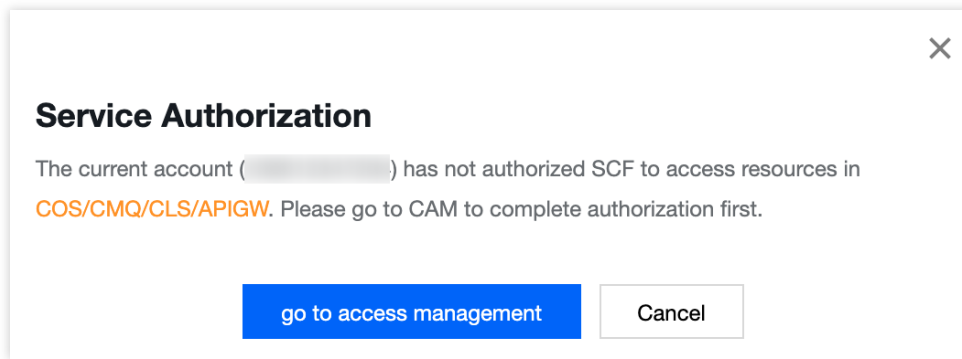
The `SCF_QcsRole` role is used to grant SCF the permissions to read and manipulate user resources during configuration. If you receive an error for missing role or permission when managing functions (such as using TCCLI or

VS Code plugin to update function code), you need to configure the `SCF_QcsRole` role.

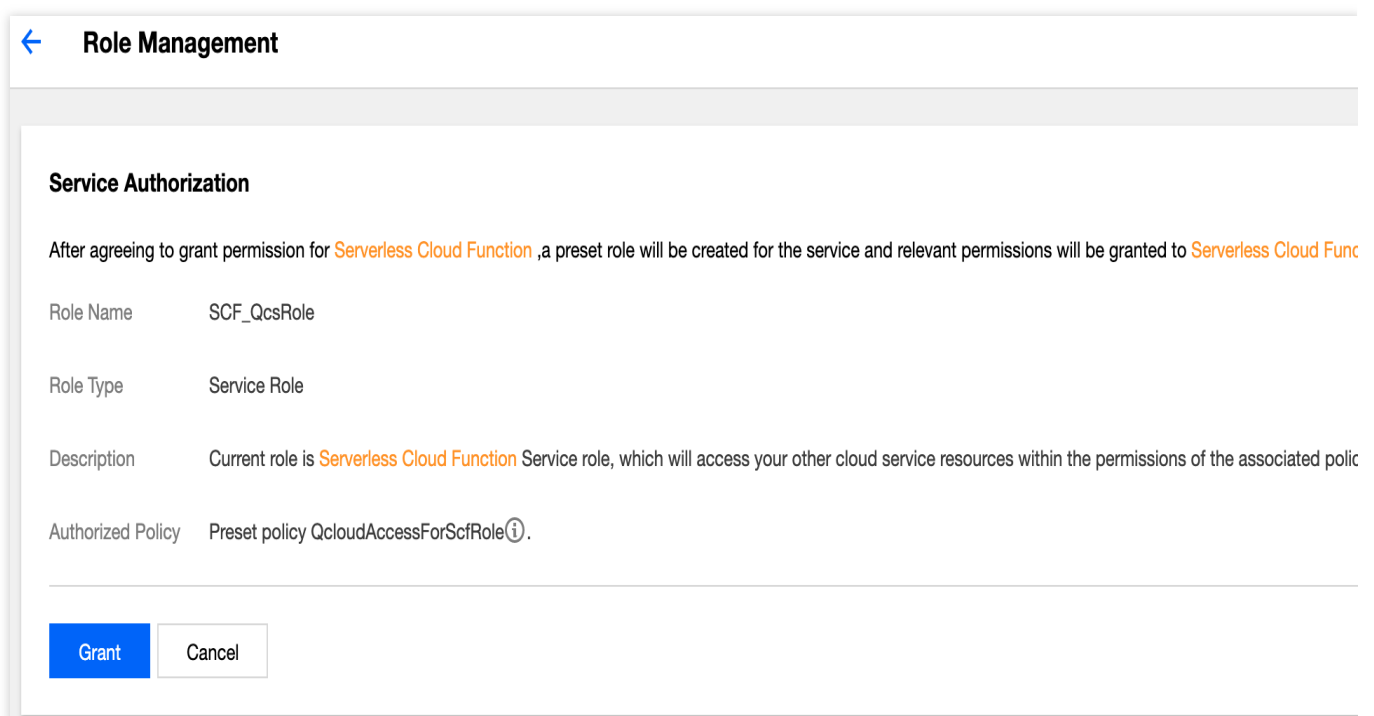
**Note:**

If you are currently a sub-user/collaborator, authorization should be performed by the root account in the following steps. After the authorization is completed, both the root account and sub-user can log in and use the SCF service.

1. If you are using SCF for the first time, you will be prompted for service authorization when you open the [SCF Console](#) as shown below:



2. Select **Go to CAM** to enter the "Role Management" page and click **Agree to Authorize** to confirm the authorization as shown below:



3. After the authorization is confirmed, the role `SCF_QcsRole` will be automatically created for you as shown below:

**Why are there new roles in my account?**

When you perform a specific operation in a service, such as authorizing to create service roles, the service may create service-related roles for you. Or, if you have been using a service before it supports service-related roles, the service may automatically create roles in your account.

Create Role

Support multi-keywords search by role name

Role Name	Role Entity	Description	Operation
<a href="#">SCF_QcsRole</a>	Product Service - scf.qcloud.com	SCF permissions (including but not li...	<a href="#">Delete</a>

1 in total

10 / page

1 / 1 page

## Appendix

### Notes on user policy update

SCF improved the preset permission policies in April 2020. The preset policies `QcloudSCFFullAccess` and `QcloudSCFReadOnlyAccess` were modified, and the `QcloudAccessForScfRole` policy was added for the configuration role `SCF_QcsRole` as shown below:

Currently, the preset policy `QcloudSCFFullAccess` has the following permissions:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "scf:*",
        "tag:*",
        "cam:DescribeRoleList",
        "cam:GetRole",
        "cam:ListAttachedRolePolicies",
        "apigw:DescribeServicesStatus",
        "apigw:DescribeService",
        "apigw:DescribeApisStatus",
        "cmqtopic:ListTopicDetail",
        "cmqueue:ListQueueDetail",
        "cmqtopic:GetSubscriptionAttributes",
        "cmqtopic:GetTopicAttributes",
        "cos:GetService",
        "cos:HeadBucket",
        "cos:HeadObject",

```

```

        "vpc:DescribeVpcEx",
        "vpc:DescribeSubnetEx",
        "cls:getTopic",
        "cls:getLogset",
        "cls:listLogset",
        "cls:listTopic",
        "ckafka:List*",
        "ckafka:Describe*",
        "monitor:GetMonitorData",
        "monitor:DescribeBasicAlarmList",
        "monitor:DescribeBaseMetrics",
        "monitor:DescribeSortObjectList",
        "monitor:DescribePolicyConditionList",
        "cdb:DescribeDBInstances"
    ],
    "resource": "*",
    "effect": "allow"
}
]
}

```

Currently, the preset policy `QcloudSCFReadOnlyAccess` has the following permissions:

```

{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "scf:Get*",
        "scf:List*",
        "ckafka:List*",
        "ckafka:Describe*",
        "monitor:GetMonitorData",
        "monitor:DescribeBasicAlarmList",
        "monitor:DescribeBaseMetrics",
        "monitor:DescribeSortObjectList",
        "cam:GetRole",
        "cam:ListAttachedRolePolicies",
        "vpc:DescribeVpcEx",
        "vpc:DescribeSubnetEx",
        "cls:getLogset",
        "cls:getTopic",
        "cls:listTopic",
        "apigw:DescribeService",
        "cmqtopic:GetTopicAttributes",
        "cmqtopic:GetSubscriptionAttributes",
        "cos:HeadBucket",

```

```

        "cos:GetService",
        "cos:GetObject"
    ],
    "resource": "*",
    "effect": "allow"
}
]
}

```

Currently, the preset policy `QcloudAccessForScfRole` has the following permissions:

```

{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "ckafka:List*",
        "ckafka:Describe*",
        "ckafka:AddRoute",
        "ckafka:CreateRoute",
        "apigw:ReleaseService",
        "apigw:CreateService",
        "apigw:CreateApi",
        "apigw>DeleteApi",
        "cls:*",
        "cos:List*",
        "cos:Get*",
        "cos:Head*",
        "cos:PutBucket",
        "cos:OptionsObject",
        "cmqqueue:*",
        "cmqttopic:*"
      ],
      "resource": "*",
      "effect": "allow"
    }
  ]
}

```

The preset policy `QcloudAccessForScfRole` can:

Write trigger configuration information to the bucket configuration if a COS trigger is configured.

Read the trigger configuration information from the COS bucket.

Read the code zip package from the bucket when the code is updated through COS.

Create API Gateway services and APIs and publish services if an API Gateway trigger is configured.

Create consumers if a CKafka trigger is configured.

# Role and Authorization

Last updated : 2024-12-02 20:11:42

## Concepts

### Roles

A [role](#) is a virtual identity with a set of permissions provided by Tencent Cloud [Cloud Access Management \(CAM\)](#). Roles can be associated with policies to grant access permissions on services, actions (operations), and resources in Tencent Cloud to [role entities](#). After these permissions are added to a role, the role can be assigned to Tencent Cloud services, allowing the services to perform operations on authorized resources on your behalf. SCF offers **configuration roles** and **execution roles**. You can use a configuration role to enable SCF to access user resources in the configuration process. You can also use an execution role to enable SCF to apply for temporary authorization for executing the code, so that the code can implement permission and resource access through the role authorization mechanism.

### Policies

A [policy](#) is a syntax rule used to define and describe one or more permissions. CAM supports two types of policies: preset policies and custom policies. The former is some common permission sets created and managed by Tencent Cloud, such as super admin and cloud resource admin, which are read-only and cannot be written. The latter is user-defined permission sets that describe resource management in a more refined way. The former cannot specifically describe individual resources and has a coarse granularity, while the latter can flexibly meet differentiated permission management needs.

### Permissions

A [permission](#) describes whether to allow or refuse the execution of certain operations to access certain resources under certain conditions. By default, a root account is the resource owner and has full access to all resources under it, while a sub-account does not have access to any resources. A resource creator does not automatically possess the access to the created resource and should be authorized by the resource owner instead.

## Overview

When creating an SCF function, you may manipulate certain Tencent Cloud services other than SCF. Different operations may require different permissions, such as COS permissions to create and delete COS triggers, API Gateway permissions to create and delete API Gateway triggers, and COS permissions to read zipped code packages, which can be granted by configuring and selecting roles.

# Configuration Roles

A configuration role is used to grant the SCF configuration the permissions to connect with other Tencent Cloud resources to access such resources within the scope of the permissions in the associated policies, including but not limited to code file access and trigger configuration. The preset policy of the configuration role supports the basic operations of function execution and covers the basic permissions required in common SCF scenarios.

## Role details

The default configuration role of SCF is `SCF_QcsRole` , as detailed below:

Role name: `SCF_QcsRole`

Role entity: `service-scf.qcloud.com`

Role description: SCF default configuration role. This service role is used to grant SCF the permissions to configure and connect with other resources in the cloud, including but not limited to code file access and trigger configuration. The preset policy of the configuration role can support the basic operations of function execution.

Associated policies: This role is associated with the `QcloudAccessForScfRole` policy, which can:

Write trigger configuration information to the bucket configuration when a COS trigger is configured.

Read the trigger configuration information from the COS bucket.

Read the code zip package from the bucket when the code is updated through COS.

Create API Gateway services and APIs and publish services when an API Gateway trigger is configured.

Perform operations such as configuring and using CLS read/write access.

Perform operations such as configuring and using CMQ read/write access.

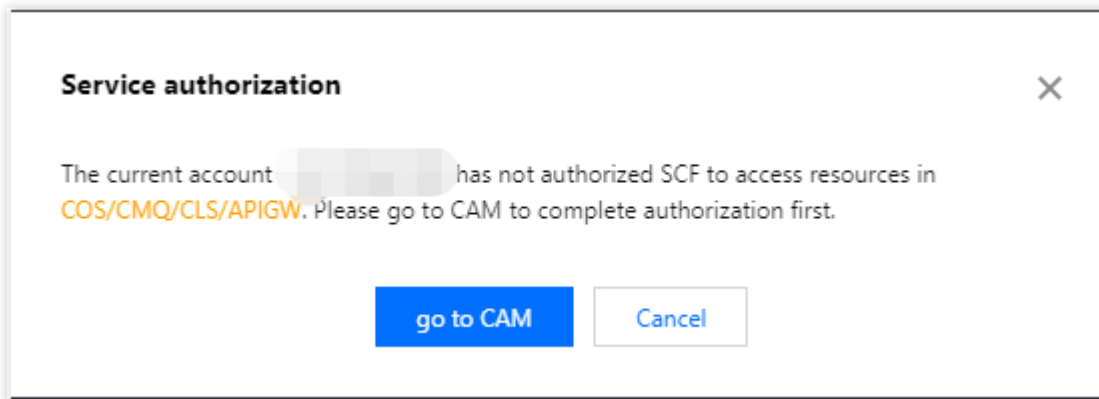
Perform operations such as configuring and using CKafka creation and lists.

### Note:

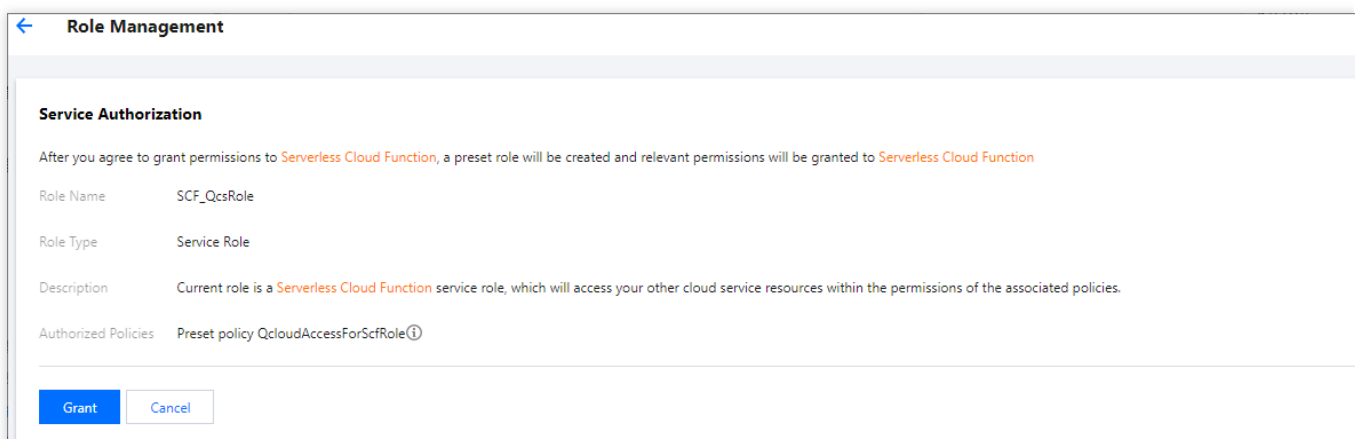
You can log in to the [CAM console](#) to view and modify the policies associated with the current configuration role `SCF_QcsRole` . However, modifying the associated policies may cause SCF to fail. Therefore, we do not recommend modifying the associated policies.

## Service authorization


1. If you are using SCF for the first time, you will be prompted for service authorization when you open the [SCF console](#), as shown below:



2. Click **go to CAM** to go to the “Role Management” page, and click **Grant** to confirm the authorization, as shown below:



3. After the authorization is confirmed, the role `SCF_QcsRole` will be automatically created, which can be viewed in [Roles](#), as shown below:



Role Name	Role ID	Role Entity	Description	Tag Information	Max session du...	Creation Time	Operatio
SCF_QcsRole	[redacted]	Product Service - scf	SCF permissions (including but not limited to): COS(create trigger); API Gateway(cre...		2 hours	2022-12-19 15:2...	Delete

## Execution Roles

An execution role is used for user code, and the role entity is `service-scf.qcloud.com`. After you add the corresponding execution role to the function, SCF will apply for the temporary authorization for your code to run within the scope of the permissions in the policies associated with the execution role, so that the code can get the required

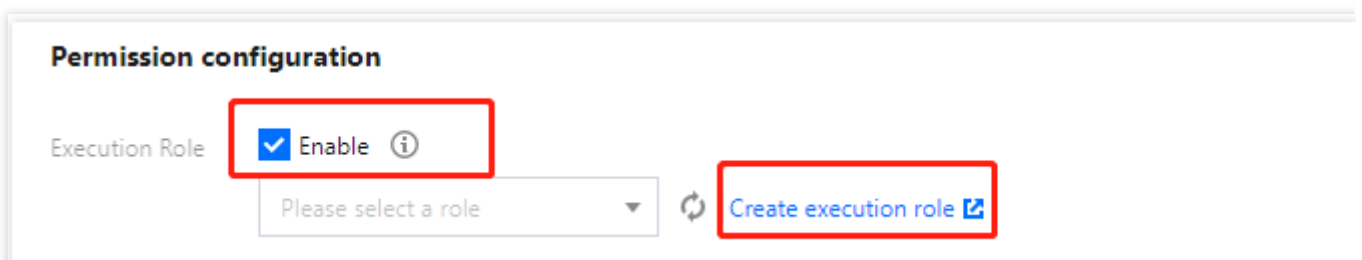


permissions and access other Tencent Cloud resources through the role authorization mechanism.

Take `SCF_QcsRole` as an example. You can also select `SCF_QcsRole` as the function execution role to grant the corresponding permissions of the policies associated with the `SCF_QcsRole` to SCF, so that SCF can get the permission to apply for access to other Tencent Cloud resources for your code.

## Creating execution roles

1. Log in to the [SCF console](#) and click **Functions** in the left sidebar.
2. On the **Functions** page, click the name of the target function to go to the function configuration page.
3. Click **Edit** in the upper-right corner of the **Function configuration** page.
4. Select **Enable** for **Execution Role** and click **Create execution role**, as shown below:



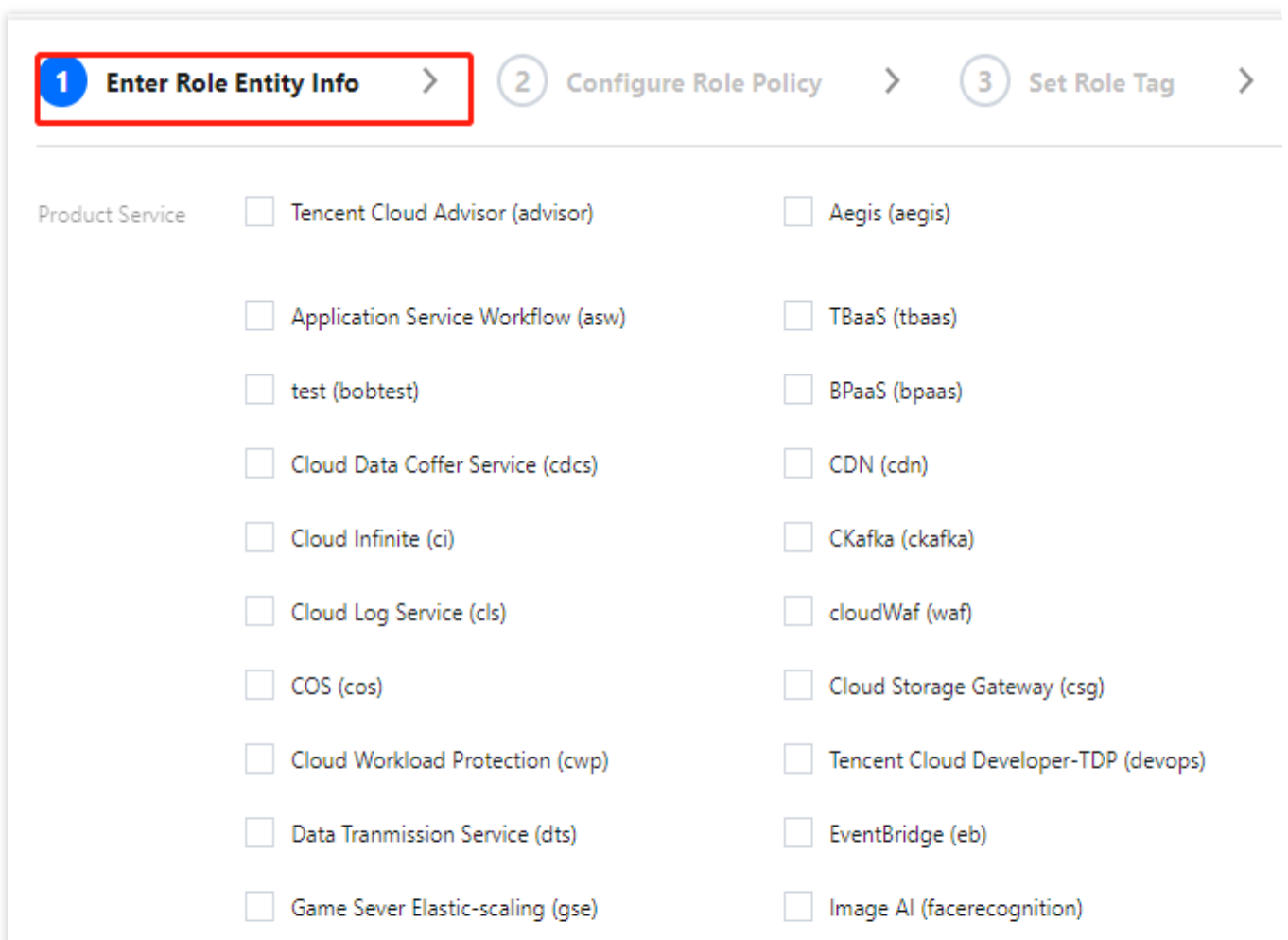
**Permission configuration**

Execution Role ☒ Enable ⓘ

Please select a role ▼

↻ [Create execution role](#) ↗

5. In the **Enter Role Entity Info** step, select **Serverless Cloud Function (scf)** and click **Next**.



**1 Enter Role Entity Info** > **2 Configure Role Policy** > **3 Set Role Tag** >

Product Service

<input type="checkbox"/> Tencent Cloud Advisor (advisor)	<input type="checkbox"/> Aegis (aegis)
<input type="checkbox"/> Application Service Workflow (asw)	<input type="checkbox"/> TBaaS (tbaas)
<input type="checkbox"/> test (bobtest)	<input type="checkbox"/> BPaaS (bpaas)
<input type="checkbox"/> Cloud Data Coffer Service (cdcs)	<input type="checkbox"/> CDN (cdn)
<input type="checkbox"/> Cloud Infinite (ci)	<input type="checkbox"/> CKafka (ckafka)
<input type="checkbox"/> Cloud Log Service (cls)	<input type="checkbox"/> cloudWaf (waf)
<input type="checkbox"/> COS (cos)	<input type="checkbox"/> Cloud Storage Gateway (csg)
<input type="checkbox"/> Cloud Workload Protection (cwp)	<input type="checkbox"/> Tencent Cloud Developer-TDP (devops)
<input type="checkbox"/> Data Transmission Service (dts)	<input type="checkbox"/> EventBridge (eb)
<input type="checkbox"/> Game Sever Elastic-scaling (gse)	<input type="checkbox"/> Image AI (facerecognition)

<input type="checkbox"/> Internet of Things Video (iotvideo)	<input type="checkbox"/> Intelligent Viewdata Computing (iss)
<input type="checkbox"/> StreamLive (mdl)	<input type="checkbox"/> StreamPackage (mdp)
<input type="checkbox"/> Cloud MongoDB (mongodb)	<input type="checkbox"/> Media Processing Service (mps)
<input type="checkbox"/> Publicly Accessible Instance-PAI (pai)	<input checked="" type="checkbox"/> Serverless Cloud Function (scf)
<input type="checkbox"/> Secrets Manager (ssm)	<input type="checkbox"/> Tencent Cloud Base (tcb)
<input type="checkbox"/> Tencent Container Security Service (tcss)	<input type="checkbox"/> Tencent Database Middleware (tdm)
<input type="checkbox"/> Tencent Cloud Infrastructure as Code (tic)	<input type="checkbox"/> TencentCloud TI Platform TI-ONE (tione)
<input type="checkbox"/> Tencent Service Framework (tsf)	<input type="checkbox"/> Tencent Support System (tss)
<input type="checkbox"/> Vulnerability Scan Service (vss)	<input type="checkbox"/> WeData Data Development Platform (wedata)
<input type="checkbox"/> Cloud Operations Console (zhiyun)	

[Next](#)

6. In the **Configure role policy** step, select all the policies required by the function and click **Next** as shown below:

**Note:**

In this example, `QcloudCOSFullAccess` (full access permissions of COS) is selected. Please select the policies as needed.

1 Enter Role Entity Info > 2 Configure Role Policy > 3 Set Role Tag > 4 Review

Select Policies (49 Total)

cos

Policy Name	Policy type
<input checked="" type="checkbox"/> QcloudCOSFullAccess Full read-write access to Cloud Object Storage (COS)	Preset Policy
<input type="checkbox"/> QcloudCOSGetServiceAccess Access to the bucket list of Cloud Object Storage (COS)	Preset Policy
<input type="checkbox"/> QcloudCOSListOnly Grant LIST-only access on COS (List Buckets & Objects)	Preset Policy
<input type="checkbox"/> QcloudCOSMetaAccMgmt This preset policy is used to authorize sub-accounts to operate the CHDFS file system associated with COS buck...	Preset Policy
<input type="checkbox"/> QcloudCOSReadOnlyAccess	Preset Policy

Support for holding shift key down for multiple selection

[Back](#) [Next](#)

1 selected

Policy Name	Policy type
QcloudCOSFullAccess Full read-write access to Cloud Object Storage (COS)	Preset Policy

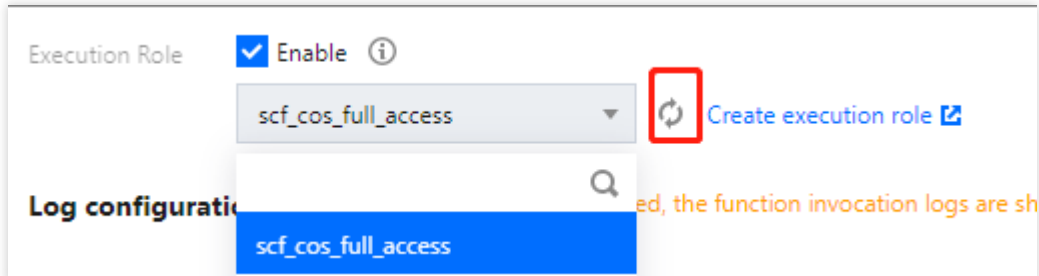
7. Specify **Role name** in the **Review** step and click **Complete**. This document uses the role name

`scf_cos_full_access` as an example.

8. Return to the **Function configuration** page, click



on the right of "Execution Role", and then select the execution role created just now from the drop-down list, as shown below:



**Note:**

When adding policies to an execution role, in addition to preset policies, you can also select custom policies to configure permissions in a more refined manner. SCF policy syntax follows the [syntax structure](#) and [resource description method](#) of CAM, and is based on the JSON format. For more information, see [SCF Policy Syntax](#).

## Getting the temporary key information of an execution role

When a function is running, the SCF service will use the selected execution role to apply for the temporary

`SecretId` , `SecretKey` , and `SessionToken` .

For **functions not created based on an image**:

They are passed in to the runtime environment in the form of environment variable as shown below:

```
TENCENTCLOUD_SECRETID: 'AKIDhOeFP...',
TENCENTCLOUD_SECRETKEY: 'r24xDBulc...',
TENCENTCLOUD_SESSIONTOKEN: 'GJmlp...'
```

Take Python as an example. You can run the following code to pass in the above information to the function runtime environment and get it in the form of environment variables:

```
secret_id = os.environ.get('TENCENTCLOUD_SECRETID')
secret_key = os.environ.get('TENCENTCLOUD_SECRETKEY')
token = os.environ.get('TENCENTCLOUD_SESSIONTOKEN')
```

For **functions created based on an image**:

They are passed in to the input parameter `context` in the form of an HTTP header. For more information, see [Feature Description](#).

# SCF Policy Syntax

Last updated : 2024-12-02 20:11:42

## Policy Syntax

You can create custom policies by using JSON syntax. SCF policies follow the CAM [syntax structure](#) and [resource description method](#). You can check [Creating Custom Policy](#) for the direction. All resources are described in the six-segment style, as shown in the sample below:

```
qcs::scf:region:uin/uin-id:namespace/namespace-name/function/function-name
```

### Note:

To configure the policy syntax, you also need to use the monitor APIs to get the monitoring information under the account (See [sample policy](#)).

## Sample Policies

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "scf:ListFunctions",
        "scf:GetAccountSettings",
        "monitor:*"
      ],
      "resource": ["*"]
    },
    {
      "effect": "allow",
      "action": [
        "scf:DeleteFunction",
        "scf:CreateFunction",
        "scf:InvokeFunction",
        "scf:UpdateFunction",
        "scf:GetFunctionLogs",
```

```

        "scf:SetTrigger",
        "scf:DeleteTrigger",
        "scf:GetFunction",
        "scf:ListVersion"
    ],
    "resource":
    [
        "qcs::scf:ap-guangzhou:uin/*****:namespace/default/function/Test1"
        "qcs::scf:ap-guangzhou:uin/*****:namespace/default/function/Test2"
    ]
    }
}

```

If the `action` needs to be associated with a resource, the resource can be defined as `*`, indicating that all resources are to be associated.

If the `action` does not need to be associated with a resource, the resource needs to be defined as `*`.

This sample allows the sub-account to have the operation permissions of certain functions under the root account. The resource in `resource` is described as a function under the root account.

## Specifying Conditions

The access policy language allows you to specify conditions when granting permissions, such as limiting the user access source or authorization time. The list below contains supported condition operators as well as general condition keys and examples.

Condition Operator	Description	Condition Name	Example
<code>ip_equal</code>	IP is equal to	<code>qcs:ip</code>	<code>{"ip_equal":{"qcs:ip ":"10.121.2.0/24"}}</code>
<code>ip_not_equal</code>	IP is not equal to	<code>qcs:ip</code>	<code>{"ip_not_equal":{"qcs:ip ":"["10.121.1.0/24", "10.121.2.0/24"]}}</code>
<code>date_not_equal</code>	Time is not equal to	<code>qcs:current_time</code>	<code>{"date_not_equal":{"qcs:current_time":"2016-06-01T00:01:00Z"}}</code>
<code>date_greater_than</code>	Time is later than	<code>qcs:current_time</code>	<code>{"date_greater_than":{"qcs:current_time":"2016-06-01T00:01:00Z"}}</code>
<code>date_greater_than_equal</code>	Time is later than or equal to	<code>qcs:current_time</code>	<code>{"date_greater_than_equal":{"qcs:current_time":"2016-06-01T00:01:00Z"}}</code>

date_less_than	Time is earlier than	qcs:current_time	{"date_less_than": {"qcs:current_time": "2016-06-01T00:01:00Z"}}
date_less_than_equal	Time is earlier than or equal to	qcs:current_time	{"date_less_than": {"qcs:current_time": "2016-06-01T00:01:00Z"}}
date_less_than_equal	Time is earlier than or equal to	qcs:current_time	{"date_less_than_equal": {"qcs:current_time": "2016-06-01T00:01:00Z"}}

To allow access only by IPs in the `10.121.2.0/24` IP range, use the following syntax:

```
"ip_equal": {"qcs:ip": "10.121.2.0/24"}
```

To allow access only by IPs `101.226.*.*.185` and `101.226.*.*.186`, use the following syntax:

```
"ip_equal": {
  "qcs:ip": [
    "101.226.*.*.185",
    "101.226.*.*.186"
  ]
}
```

## User Policy Update

SCF improved the preset permission policies in April 2020. The preset policies `QcloudSCFFullAccess` and `QcloudSCFReadOnlyAccess` were modified, and the `QcloudAccessForScfRole` policy was added for the configuration role `SCF_QcsRole`, as shown below:

### Preset policy `QcloudSCFFullAccess`

Current permissions:

```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "scf:*",
        "tag:*",
```

```

        "cam:DescribeRoleList",
        "cam:GetRole",
        "cam:ListAttachedRolePolicies",
        "apigw:DescribeServicesStatus",
        "apigw:DescribeService",
        "apigw:DescribeApisStatus",
        "cmqtopic:ListTopicDetail",
        "cmqqueue:ListQueueDetail",
        "cmqtopic:GetSubscriptionAttributes",
        "cmqtopic:GetTopicAttributes",
        "cos:GetService",
        "cos:HeadBucket",
        "cos:HeadObject",
        "vpc:DescribeVpcEx",
        "vpc:DescribeSubnetEx",
        "cls:getTopic",
        "cls:getLogset",
        "cls:listLogset",
        "cls:listTopic",
        "kafka:List*",
        "kafka:Describe*",
        "kafka:ListInstance",
        "monitor:GetMonitorData",
        "monitor:DescribeBasicAlarmList",
        "monitor:DescribeBaseMetrics",
        "monitor:DescribeSortObjectList",
        "monitor:DescribePolicyConditionList",
        "cdb:DescribeDBInstances"
    ],
    "resource": "*",
    "effect": "allow"
}
]
}

```

### Preset policy `QcloudSCFReadOnlyAccess`

Current permissions:

```

{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "scf:Get*",
        "scf:List*",
        "kafka:List*",

```

```

        "ckafka:Describe*",
        "monitor:GetMonitorData",
        "monitor:DescribeBasicAlarmList",
        "monitor:DescribeBaseMetrics",
        "monitor:DescribeSortObjectList",
        "cam:GetRole",
        "cam:ListAttachedRolePolicies",
        "vpc:DescribeVpcEx",
        "vpc:DescribeSubnetEx",
        "cls:getLogset",
        "cls:getTopic",
        "cls:listTopic",
        "apigw:DescribeService",
        "cmqtopic:GetTopicAttributes",
        "cmqtopic:GetSubscriptionAttributes",
        "cos:HeadBucket",
        "cos:GetService",
        "cos:GetObject"
    ],
    "resource": "*",
    "effect": "allow"
}
]
}

```

### Preset policy `QcloudAccessForScfRole`

Current permissions:

```

{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "cos:GetBucket*",
        "cos:HeadBucket",
        "cos:PutBucket*",
        "apigw:*",
        "cls:*",
        "cos:List*",
        "cos:Get*",
        "cos:Head*",
        "cos:OptionsObject",
        "cmqqueue:*",
        "cmqtopic:*",
        "ckafka:List*",
        "ckafka:Describe*",

```



```
        "ckafka:AddRoute",
        "ckafka:CreateRoute"
    ],
    "resource": "*",
    "effect": "allow"
}
]
```

The preset policy `QcloudAccessForScfRole` can:

Write trigger configuration information to the bucket configuration when a COS trigger is configured.

Read the trigger configuration information from the COS bucket.

Read the code zip package from the bucket when the code is updated through COS.

Create API Gateway services and APIs and publish services if an API Gateway trigger is configured.

Create consumers if a CKafka trigger is configured.

# Sub-users and Authorization

Last updated : 2024-12-02 20:11:42

## Note:

The root account needs to check on the [Role](#) page whether the `SCF_QcsRole` policy is associated, and if not, grant the permissions as instructed in Service Authorization in [Role and Authorization](#); otherwise, sub-users will not be able to use the SCF console and call other Tencent Cloud resources through SCF.

## Creating a Sub-user and Granting it All SCF Permissions

### Step 1. Create a sub-user by using the root account

1. Log in to the [CAM console](#) and select **Users > User List** on the left sidebar.
2. On the **User List** page, select **Create User > Custom** to enter the **Create Sub-User** page.
3. In the **User Type** step, after selecting **Access Resources and Receive Messages**, click **Next** to enter the user information.
4. Enter and confirm the information as prompted and click **Complete**.

## Note:

For more information, see [Creating Sub-User](#).

### Step 2. Create a custom policy

1. Log in to the [CAM console](#). Click **Create Custom Policy** in the top-left corner.
2. In the pop-up window, click **Create by Policy Generator** to go to the **Edit Policy** page.
3. Select the service in the Visual Policy Generator, enter the following information, and edit an authorization statement.

**Effect:** Allow

**Service:** SCF

**Action:** All

**Resource Description:** \*

**\*\*Condition (optional)\*\*:** Empty

4. After editing the policy authorization statement, click **Next** to enter the **Associate User/User Group/Role** page.
5. On the **Associate User/User Group/Role** page, add the policy name and description, and you can associate users, user groups, or roles for quick authorization at the same time.
6. Click **Complete** to complete the custom policy creation.

### Step 3. Add CAM read-only permissions for the sub-user

1. Log in to the CAM console and enter the [User List](#) page.

2. Locate the sub-user you want to grant permission to.
3. Click **Authorize** in the **Operation** column on the right.
4. In the **Associate Policy** pop-up window, select `QcloudCamReadOnlyAccess`.
5. Click **OK**.

## Completion

After the settings above are configured, you can log in to the sub-account to view the permissions.

Log in to the CAM console and select [Overview](#) on the left sidebar to access the overview page and view the sub-user login address.

# Creating a Sub-user and Granting it Certain SCF Permissions

## Step 1. Create a sub-user by using the root account

1. Log in to the [CAM console](#) and select **Users > User List** on the left sidebar.
2. On the **User List** page, select **Create User > Custom** to enter the **Create Sub-User** page.
3. In the **User Type** step, after selecting **Access Resources and Receive Messages**, click **Next** to enter the user information.
4. Enter and confirm the information as prompted and click **Complete**.

### Note:

For more information, see [Creating Sub-User](#).

## Step 2. Create a custom policy

1. Log in to the [CAM console](#). Click **Create Custom Policy** in the top-left corner.
2. In the pop-up window, click **Create by Policy Generator** to go to the **Edit Policy** page.
3. Copy the code of the sample policy in [SCF Policy Syntax](#) and edit the policy content in **Edit Policy > JSON**.

### Note:

The resource description in `resource` needs to be replaced with the ID of the root account and the names of the functions under it. The `region` needs to be the same as that of the functions.

4. Click **Next** to enter the **Associate User/User Group/Role** page.
5. On the **Associate User/User Group/Role** page, add the policy name and description, and you can associate users, user groups, or roles for quick authorization at the same time.
6. Click **Complete** to complete the custom policy creation.

## Step 3. Add CAM read-only permissions for the sub-user

1. Log in to the CAM console and enter the [User List](#) page.
2. Locate the sub-user you want to grant permission to.
3. Click **Authorize** in the **Operation** column on the right.

4. In the **Associate Policy** pop-up window, select `QcloudCamReadOnlyAccess` .
5. Click **OK**.

## Completion

After the settings above are configured, you can log in to the sub-account to view the permissions. Click [Overview](#) on the left sidebar to access the overview page and view the sub-user login address.

### Note:

After the policy takes effect, the current sub-account will be able to see all the function names but will only be able to operate on and view the functions listed in `resource` .

# Users and Permissions

Last updated : 2024-12-02 20:11:41

## Operation Scenarios

When you use the SCF Console or CLI, you may need to manipulate the permissions to SCF or other products. For example, you can grant the permissions to query the CMQ topic list, VPC information, or Cloud Monitor statistics. SCF provides preset policies for granting access to SCF and other associated resources. For more information, please see [Notes on User Policy Update](#). SCF updates preset policies as needed to ensure that you have access to newly released features. You can also create custom policies to manage user permissions.

## Directions

### Note:

If you are currently a sub-user/collaborator, authorization should be performed by the root account in the following steps. After the authorization is completed, both the root account and sub-user can log in and use the SCF service. You may receive `you are not authorized to perform xxx` or other prompts for "no access" while accessing the SCF Console to view function monitoring or execution information.

You can create a custom policy as instructed in the [CAM documentation](#) or configure two preset policies for fast authorization by following the steps below:

1. Log in to the CAM Console and select **Users** > [User List](#) on the left sidebar.
2. Select **Authorize** to the right of the row of the user for whom to add a policy.
3. In the "Associate Policy" window that pops up, check one of the preset policies in the table below and click **OK** to complete the authorization:

This document uses the `QcloudSCFFullAccess` preset policy as an example, and you can choose an appropriate policy based on your actual needs.

Preset Policy	Purpose
QcloudSCFFullAccess	Grants full access to SCF and other relevant resources
QcloudSCFReadOnlyAccess	Grants read-only access to SCF and other relevant resources

## Notes on User Policy Update

SCF improved the preset permission policies in December 2019. The preset policies `QcloudSCFFullAccess` and `QcloudSCFReadOnlyAccess` were modified, and the `QcloudAccessForScfRole` policy was added for the configuration role `SCF_QcsRole`. For more information, please see [Notes on User Policy Update](#).

# Creating a Sub-user and Granting It All Permissions of SCF

Last updated : 2024-12-02 20:11:41

## Step 1. Create a sub-user by using the root account

1. Log in to the [CAM Console](#) and select **Users > User List** on the left sidebar.
2. On the **User List** page, select **Create User > Custom Creation** to enter the sub-user creation page.
3. Select a user type and click **Access resources and receive messages**.
4. Enter the user information. You can create sub-users and set access type and console password in batches. Please configure based on your actual needs.
5. Set the permissions. Make appropriate settings according to different business scenarios and click **Next** to save the settings. You can also change the relevant permission settings later. There are three ways to set permissions:
  - Add the sub-user to an existing or new user group.
  - Copy the permissions of an existing user.
  - Authorize in the list of policies.
6. After the creation is completed, the console will display the username, password, TencentCloud API key and other information of the sub-user. Click **Complete** to exit the page.

### Note:

For more information, please see [Creating Sub-Users](#).

## Step 2. Create a custom policy

1. Log in to the CAM Console and select **Policy** on the left sidebar.
2. On the policy management page, select **Create Custom Policy > Create by Policy Builder** to enter the creation page.
3. Select services and operations.

Set the corresponding items as follows and select **\*\*Add Statement> Next** to enter the policy editing steps:

**Effect:** allowed

**Service:** SCF

**Action:** all

**Resource:** \*

**Condition (optional):** empty

4. Edit the policy name and remarks (you are recommended to use an easy-to-understand name) and click **Create Policy** to complete the policy creation.

## Step 3. Associate a policy with a user/user group

1. On the [Policy](#) management page, select **Associate User/User Group** on the right of the created policy to pop up the prompt box for association.
2. Select the user to be associated with and click **OK** to complete the association. You can also switch between users and user groups for selection.

## Completion

After the settings above are made, you can log in to the sub-account to view the permissions.

Log in to the CAM Console and select [Overview](#) on the left sidebar to enter the overview page and view the sub-user login address.



# Creating a Sub-user and Granting It Permissions to Operate Certain Functions

Last updated : 2024-12-02 20:11:42

## Step 1. Create a sub-user by using the root account

1. Log in to the [CAM Console](#) and select **Users > User List** on the left sidebar.
2. On the **User List** page, select **Create User > Custom Creation** to enter the sub-user creation page.
3. Select a user type and click **Access resources and receive messages**.
4. Enter the user information. You can create sub-users and set access type and console password in batches. Please configure based on your actual needs.
5. Set the permissions (required). Make appropriate settings according to different business scenarios and click **Next** to save the settings. You can also change the relevant permission settings later. There are three ways to set permissions:

Add the sub-user to an existing or new user group.

Copy the permissions of an existing user.

Authorize in the list of policies.

6. After the creation is completed, the console will display the username, password, TencentCloud API key and other information of the sub-user. Click **Complete** to exit the page.

### Note:

For more information, please see [Creating Sub-user](#).

## Step 2. Create a custom policy

1. Log in to the CAM Console and select [Policy](#) on the left sidebar.
2. On the policy management page, select **Create Custom Policy > Create by Policy Builder** to enter the creation page.
3. Select services and operations.

Set the corresponding items as follows and click **\*\*Add Statement> Next** to enter the policy editing steps:

**Effect:** allowed

**Service:** SCF

**Action:** all

**Resource:** \*

**Condition (optional):** empty

4. Edit the policy name and remarks (you are recommended to use an easy-to-understand name) and modify the content of the policy by replacing it with the code in the sample policy in [Permission Management Overview](#).

**Note:**

The resource description in `resource` needs to be replaced with the ID of the root account and the names of functions under it. The `region` needs to be the same as that of the functions.

## Step 2. Associate a policy with a user/user group

1. On the [Policy Management](#) page, click **Associate User/User Group** on the right of the created policy to pop up the prompt box for association.
2. Select the user to be associated with and click **OK** to complete the association. You can also switch between users and user groups for selection.

## Completion

After the settings above are made, you can log in to the sub-account to view the permissions. Click [Overview](#) on the left sidebar to enter the overview page and view the sub-user login address.

**After the policy takes effect, the current sub-account can see all the function names but can only operate on and view the functions listed in `resource` .**

# Plugin Management

## Configure Function Plug-Ins

Last updated : 2025-06-17 16:31:12

### Feature Overview

The function platform now supports the "plug-in" feature, allowing users to upload auxiliary services such as log collection and metric reporting as plug-ins and bind them in the function. Complete the feature implementation similar to the Sidecar in the container ecosystem in this mode, decoupling the business function from the auxiliary services. At the same time, manage the lifecycle of the plug-in in combination with that of the function instance. Users can separately declare the running time of the plug-in after the main function returns the result to ensure the real-time data and integrity of the auxiliary services.

### Working Mode

#### Create and Bind

The mirror file of a plug-in will be stored based on the name and version of the plug-in. When binding a plug-in to a function, it will be bound based on the specific plug-in version and function version. Currently, a maximum of 5 plug-in versions can be bound to one function version, and they must be unique.

#### Recommended Usage Method

We recommend that users focus on compiling the core business logic in the main function and decouple the core business logic from auxiliary components. These auxiliary components can collaborate with the main function by functioning in the form of function plugins. Common plugin usage scenarios include:

Log gathering: Collection, formatting, and submission of logs.

Metric reporting: Collect service performance metrics and send them to the monitoring system. Centralize monitoring logic management and reduce the burden on the main service.

APM tool: Real-time monitor and analyze application performance metrics, promptly discover and resolve potential performance bottlenecks and failures, and underwrite the stable operation of applications.

Tencent Cloud Mesh (TCM): Service communication, traffic management.

Configuration management: Dynamically manage service configurations, support hot updates, and avoid frequent service restarts to apply new configurations.

Probe health: Check whether specific components are working properly.

Other system integrations.

## Loading and Accessing at Runtime

In Serverless Cloud Function (SCF), the plug-in feature is implemented through a dynamic loading mechanism, achieving capabilities similar to the Kubernetes Sidecar mode. When a function instance starts up, all plug-ins are started asynchronously, and the plug-in startup phase does not block the invocation of the main function. Plug-ins support running independent processes (such as log agents, monitoring services) and interact with the main function via the local network (127.0.0.1:port) or a shared file directory (/tmp). Unlike static dependencies in "layers", plug-ins emphasize dynamic service capabilities (such as traffic proxy, real-time security verification), allowing backend logic to run continuously within the function lifecycle, unifying feature expansion and resource isolation.

## Related Documentation

You can create a plugin through the Serverless Console [Create Plugin](#).

When creating or updating a function, [bind a plugin](#).

# Create Plugin

Last updated : 2025-06-17 16:31:30

This article introduces how to create a plugin via Serverless Console.

## Operation Steps

1. Log in to [Serverless Console](#) and select **Advanced Capability > Plug-in** in the left sidebar.
2. On the **Plugin** page, select the region where you need to use the plugin and click **Create**.
3. On the **Create plugin** page, set the plugin information according to actual needs. As shown below:

**Plugin name:** Manually input custom layer name.

**Description of Plugin version:** Version description. Fill in according to the actual situation.

**Deployment Type:** Currently, only image deployment can be selected.

**Repository address:** Plugin image address.

**Entrypoint:** Enter the container's startup command. Follow the writing norm for parameters, and fill in a runnable instruction, such as python. This parameter is optional; if not filled, the Entrypoint in the Dockerfile will be used by default.

**CMD:** Fill in the container's startup parameter. Follow the writing norm for the parameter, using a space as the segmentation identifier for the parameter. For example, -u app.py. This parameter is optional. If not filled, use the CMD in the Dockerfile by default.

**Tag:** A tag is in the plug-in dimension. All versions under this plug-in have the same tag.

4. Click **OK** to submit the add-on creation and automatically return to the plug-in list. You can view the add-on creation situation in the plug-in list.

# Bind a plugin

Last updated : 2025-06-17 16:31:48

This article introduces how to bind a layer to SCF via Serverless Console.

## Directions

1. Log in to Serverless Console, select **Function Service** in the left sidebar, and enter the "Function Service" list page.

2. Can bind a plugin directly in advanced configuration when creating a function, or edit an existing function and perform binding on the **Plugin Management** tab.

3. When creating a new function, as shown below:

Select the corresponding plug-in to bind, and set the CPU and memory specifications, delay termination time, and whether to share CLS configuration assigned to the plug-in.

4. Complete the function configuration item filling, then click OK to finish creating the function. At this point, you can view the plugin details bound to this function version in Function Management - Plugin Management.

# Managing Monitors and Alarms

## Descriptions of monitoring metrics

Last updated : 2024-12-02 20:11:42

Tencent Cloud Monitor provides the following monitoring metrics for SCF:

Currently, two levels of monitoring metrics are supported. Monitoring metrics at the function level can be viewed in specific functions, while those at the region level can be viewed by selecting a specific region on the overview page to display the statistics of all function monitoring metrics.

Metric Name	Parameter	Description	Unit	Dimension
Execution duration	duration	The execution duration at the function or region level refers to the function code execution time from start to end, averaged by granularity (1 minute or 5 minutes).	Milliseconds	Function Region
Number of invocations	invocation	The number of requests at the function or region level, summed by granularity (1 minute or 5 minutes).	-	Function Region
Number of errors	error	The number of error requests after the function is executed, which currently includes those on the client side and on the platform, summed by granularity (1 minute or 5 minutes).	-	Function Region
Number of concurrent executions	concurrent_executions	The number of requests processed concurrently at the same time point, summed by granularity (1 minute or 5 minutes) and determined by the maximum value at the function or region level.	-	Function Region
Number of restricted requests	throttle	The number of requests that reaches the bandwidth limit at the function or region level,	-	Function Region

		summed by granularity (1 minute or 5 minutes).		
Execution memory	mem	The actual memory used by the function runtime, whose maximum value is calculated by granularity (1 minute or 5 minutes).	MB	Function
Time memory	mem_duration	Resource usage, which is calculated by multiplying the function execution duration by the execution memory, summed by granularity (1 minute or 5 minutes).	MBms	Function
Outbound traffic	out_flow	The outbound traffic generated by accessing resources on the public network from a function, summed by granularity (1 minute or 5 minutes).	KB	Function
System internal error (HTTP 5xx)	syserr	The number of 5xx status codes returned after function execution, summed by granularity (1 minute or 5 minutes).	-	Function
Function error (HTTP 4xx)	http_4xx	The number of 4xx status codes returned after function execution, summed by granularity (1 minute or 5 minutes).	-	Function
Successful invocations (HTTP 2xx)	http_2xx	The number of 2xx status codes returned after function execution, summed by granularity (1 minute or 5 minutes).	-	Function
Resource limit exceeded (HTTP 432)	http_432	The number of 432 status codes returned after function execution, summed by granularity (1 minute or 5 minutes).	-	Function



Function execution timed out (HTTP 433)	http_433	The number of 433 status codes returned after function execution, summed by granularity (1 minute or 5 minutes).	-	Function
Memory limit exceeded (HTTP 434)	http_434	The number of 434 status codes returned after function execution, summed by granularity (1 minute or 5 minutes).	-	Function

**Note:**

For more information on **how to get the required monitoring data**, please see [SCF Monitoring Metrics](#).

# Configuring Alarms

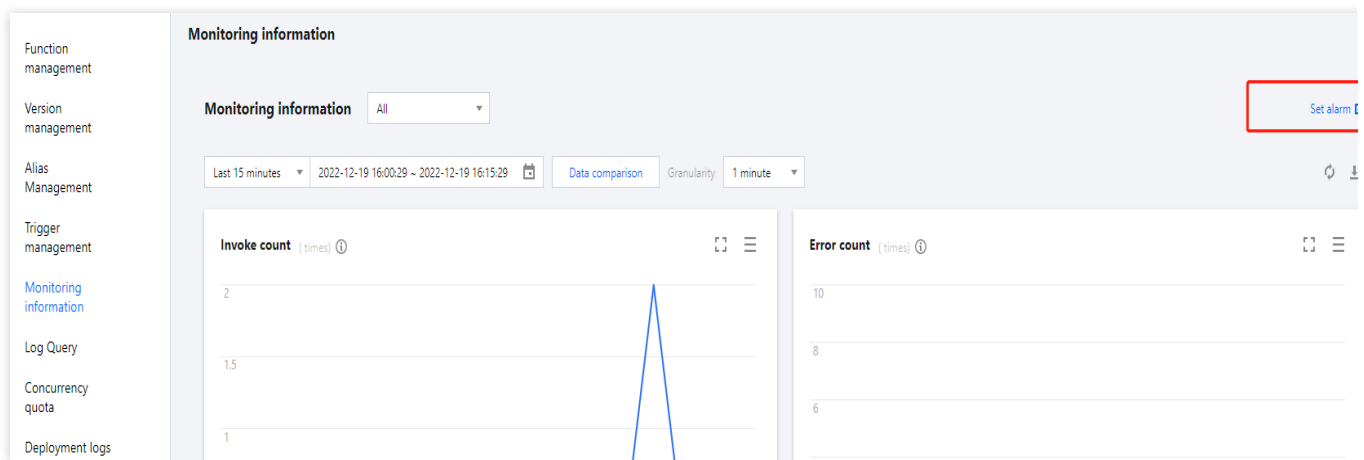
Last updated : 2024-12-02 20:11:41

## Overview

You can configure alarm policies for SCF through [Cloud Monitor](#) to monitor the execution status of functions. Currently, the monitoring metrics that can be configured for SCF include runtime, invoke count, and error count. For the full list of supported metrics, see [Descriptions of monitoring metrics](#). In addition, you can select user groups that will receive alarms through email, SMS, or WeChat.

## Directions

1. Log in to the [SCF console](#) and select **Functions** on the left sidebar.
2. On the **Functions** page, click the target function name to enter the function details page.
3. Select **Monitoring information** on the left and click **Set alarm** on the details page as shown below:



4. On the **Create alarm policy** page, configure a new alarm policy as detailed below:

**Policy Name:** Custom.

**Monitoring Type:** Select **Tencent Cloud services**.

**Policy Type:** Select **Function/Version or Function/Alias**.

**Alarm Object:** Select the functions for which to apply the policy as needed. If **Instance ID** is selected, the region will be set to Guangzhou by default. You can view corresponding functions in different regions. For more information on detailed alarm policy configuration, see [Creating Alarm Policy](#).

5. Click **Complete**, and you can see the configured policy in **Alarm Management** > [Policy Management](#) and choose to enable/disable it at any time.



# Viewing Execution Logs

Last updated : 2024-12-02 20:11:41

## Overview

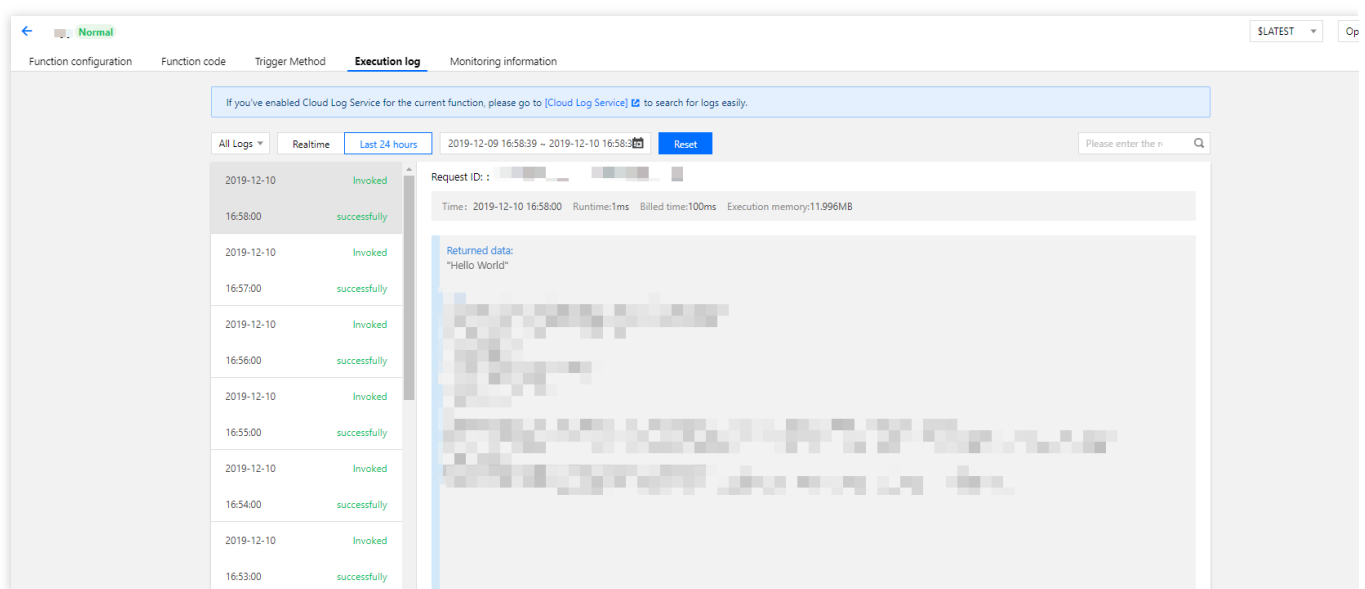
You can view the function execution logs in the SCF console. You can filter to display real-time logs or logs from the last 24 hours. You can also specify a custom time range. You can choose to view all logs, or logs of successful, failed, timed-out, and excessive invocations and code exceptions in the SCF console.

## Directions

You can view function logs in the SCF console.

### Viewing execution logs in console

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Click a function name to enter the function details page.
3. On the function details page, select **Log Query** on the left to open the **Invocation Logs** page as shown below:

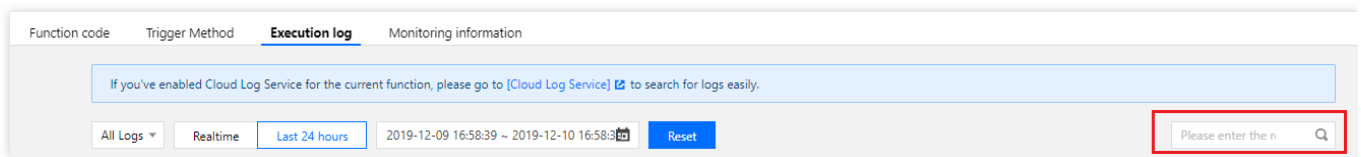


### Finding execution log

Follow the steps below to find an execution log as needed.

### Invocation log

In the top-right corner search box, enter the `requestID` of the execution log you want to view and press Enter:



You can also set custom search criteria in the top-left corner based on actual needs.

**All Logs:** you can select logs of successful invocations or failed invocations.

**Select a date:** you can view the execution logs generated in the last 6 days up to today. Currently, you can only search for logs in a time range of no more than 24 hours.

**Realtime:** you can view the current execution log of a function.

**Last 24 hours:** you can view the execution logs generated in the last 24 hours.

### Advanced search

You can search for SCF logs by keyword or use query syntax to combine keywords for search. For more information, please see [Log Search Guide](#).

# Network Configuration

## Network Configuration Management

Last updated : 2024-12-02 20:11:41

### Overview

By default, a newly created cloud function has only the public network access permission. That is, when the cloud function is being executed, it can only access public network resources such as Tencent Cloud.

SCF supports the following network configurations:

Configuration Item	Description
Only public network access is enabled.	-
Public network access with a fixed public outbound IP is enabled.	The cloud function can access public network resources using a fixed IP address.
Only private network access is enabled.	The cloud function can access resources configured with the private network, such as databases and Redis.
Both public and private network accesses are enabled.	The cloud function can access public network resources and resources configured with the private network, such as databases and Redis.
Private network access and public network access with a fixed public outbound IP are enabled.	The cloud function can access public network resources using a fixed IP address and access resources configured with the private network, such as databases and Redis.

#### Note:

For more information about how the cloud function obtains the fixed public outbound IP, please see [Fixed Public Outbound IP](#).

### Prerequisites

You have registered a Tencent Cloud account. If you have not, please go to the [Sign up](#) page.

You have [created a cloud function](#).

## Procedure

### Performing network configuration

1. Log in to the [SCF console](#) and click **Function Service** in the left sidebar.
2. Choose a region at the top of the page and click the function to be configured.
3. In the **Function configuration** module, click **Edit** in the upper right corner.
4. Configure the network as needed by referring to the related [network configuration documentation](#).

### Viewing network configuration

1. Log in to the [SCF console](#) and click **Function Service** in the left sidebar.
2. Choose the region at the top of the page and click the function. The network configuration is displayed in the **Function configuration** module.

## Network Restrictions

### Concurrent connection count limits

Currently, a maximum of 60,000 connections can access the same ip:port concurrently. For non-persistent connections, since the release of intermediate devices takes time, the number of supported concurrent connections is smaller.

If multiple functions (or multiple requests of a function) need to access the same ip:port concurrently, you should pay attention to this limit. You can take the following measures to avoid using up the connections and making code errors. Use persistent connections as much as possible. During function initialization, complete and continuously reuse the connections. This avoids frequent connections and releases caused by the use of non-persistent connections during invocation. This measure gives full play to the supported connection count, but it is still limited by the connection count limit.

Provide multiple ip:port pairs. In this way, connections are spread to multiple ip:port pairs to avoid reaching the connection count limit.

### Private network bandwidth limits

If the VPC is connected to a private network, the bandwidth for a single VPC is limited to 100 MB. The bandwidth is shared by the functions as well as the concurrent instances of these functions configured with the same VPC. To increase the private network bandwidth, please [submit a ticket](#).

# Fixed Public Outbound IP

Last updated : 2024-12-02 20:11:42

## Overview

When accessing a database API or third-party service in SCF, you can use the fixed public outbound IP feature of SCF to control and manage the SCF network configuration.

The fixed public outbound IP feature of SCF has the following capabilities:

If fixed public outbound IP is enabled for an SCF function, the function will get a random EIP. The traffic generated by the function accessing the public network will be forwarded based on the EIP.

If both public network access and private network access as well as the fixed public outbound IP are enabled for the function, the traffic generated by accessing the public network will be forwarded based on the EIP, while that generated by accessing the private network will be forwarded based on the VPC.

## Limits

An EIP is shared under the same account in the same region.

Under the same account in the same region, functions with fixed public outbound IP enabled share the same EIP.

If you want to change the fixed outbound IP of a function, you need to disable the fixed public outbound IP feature for all functions under the same account in the same region. After you enable this feature again, a new EIP will be generated randomly.

The EIP is shared based on the subnet of the VPC.

If a function is configured with a VPC and has fixed public outbound IP enabled, it will get a random EIP. If another function in the same VPC subnet has fixed public outbound IP enabled, it will share the same fixed outbound IP.

## Sample

The following example shows the use limits of fixed public outbound IP:

Assume your account has the following objects in a region:

Functions a and b have been created under namespace A.

Functions c and d have been created under namespace B.

EIPs IP-x and IP-y represent two different EIPs.

Their EIP and function binding relationships are as shown below:

Network Configuration	Namespace A		Namespace B	
	Function a	Function b	Function c	Function d



Only public network access is enabled	No EIP	No EIP	No EIP	No EIP
Only private network access is enabled	No EIP	No EIP	No EIP	No EIP
Public network access and fixed public outbound IP are enabled	EIP IP-x	EIP IP-x	EIP IP-x	EIP IP-x
The same VPC is used for access, and fixed public outbound IP is enabled	EIP IP-y	EIP IP-y	EIP IP-y	EIP IP-y

## Directions

**Note:**

You can have up to 5 fixed pubic outbound IPs in a region.

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Select the function region at the top of the page and click the function name.
3. Enter the **Function Configuration** tab and click **Edit** in the top-right corner.
4. Configure the function network as needed as shown below:

**Note:**

After public network access is enabled for the function, you can enable fixed public outbound IP.

You cannot manually select or edit the randomly generated EIP.

Public network

☒ Enable ⓘ

Fixed outbound IP

☒ Enable ⓘ

VPC

☒ Enable ⓘ

Please select the VPC

Please select a subnet

[Create VPC](#)

After completing the configuration, click **Save**.

# VPC Communication

Last updated : 2024-12-02 20:11:41

## Overview

SCF is deployed in the public network by default. This document describes how to enable SCF to access resources in the private network through VPC configuration, such as TencentDB, CVM, TencentDB for Redis, and CKafka, which helps ensure the data and connection security.

## Notes

When configuring a VPC, pay attention to the following points:

A function deployed in a VPC is isolated from the public network by default. If you want the function to have access to both private and public networks, you can do so in the following two ways:

Configure the public network access of SCF and make sure that the egress address for public network access is unique. For more information, please see [Fixed Public Outbound IP](#).

Add a NAT gateway through VPC. For more information, please see [Granting a Function in VPC Access to Public Network](#).

Currently, functions cannot be connected with resources on the classic network.

## Prerequisites

You have [created a function](#).

## Directions

### Modifying network configuration

1. Log in to the SCF console and click [Function Service](#) on the left sidebar.
2. Select the region at the top of the page and click the name of the function to be configured.
3. On the **Function Configuration** page, click **Edit** in the top-right corner.
4. Enable **VPC** and select the VPC to be accessed and the subnet you want to use.

### Using VPC

After you configure the private network access for a function and start to use the VPC, the function will switch from the current independent network environment to the configured VPC. When the function starts, an IP address in your VPC subnet will be used as the IP address of the function runtime environment. In order to reduce the function's usage of subnet IP addresses, running function instances will share a proxy pair and scale the proxy pair based on the network bandwidth utilization.

After the function is started, you can use the code and private IP address to access resources whose access entries are in the VPC, such as [TencentDB for Redis](#), TDSQL, and CVM.

The following is the sample code for accessing [TencentDB for Redis](#), where the IP address of the Redis instance in the VPC is `10.0.0.86`.

```
# -*- coding: utf8 -*-
import redis
def main_handler(event, context):
    r = redis.StrictRedis(host='10.0.0.86', port=6379, db=0, password="crs-i4kg86dg")
    print(r.set('foo', 'bar'))
    print(r.get('foo'))
    return r.get('foo')
```

### Accessing custom domain name in VPC

Using Private DNS to access custom domain name in VPC (recommended)

Setting Name Server in SCF environment

In VPC, if you need to access a self-built service on the private network at a domain name, you can use the [Private DNS](#) provided by Tencent Cloud to configure and resolve the custom domain name on the private network.

If you want to connect to a custom DNS server, you need to customize the `name server` configuration in the SCF environment. Currently, you can implement this by configuring the `OS_NAMESERVER` environment variable as shown below:

Environment Variable	Value Rule	Description
OS_NAMESERVER	It can be one or more IP addresses or domain names separated by ;. A maximum of 5 custom name servers can be configured.	It configures the custom name server.

As shown in the following code implemented in Python, the configuration can be checked for effect by printing out the `/etc/resolv.conf` file.

```
with open("/etc/resolv.conf") as f:
    print(f.readlines())
```

## Relevant Operations

### Viewing network configuration

1. Log in to the SCF console and click **Function Service** on the left sidebar.
2. Select a region at the top of the page and click the name of the function for which private network access has been configured to view the specific configuration through the corresponding **network** and **subnet**.

# Granting a Function in VPC Access to Public Network

Last updated : 2024-12-02 20:11:41

## Operation Scenarios

A function deployed in a VPC is isolated from the public network by default. If you want the function to have access to both private network and public network, you can add a NAT gateway to the VPC.

## Prerequisites

You have [created a function](#).

## Directions

### Creating NAT Gateway

NAT Gateway is a network cloud service that supports IP address translation and enables high-performance internet access for resources in Tencent Cloud. It can translate the private IP address in a VPC to a public IP address if the private and public networks are isolated from each other, enabling the VPC to access the internet. For more information, please see [NAT Gateway Overview](#).

1. Log in to the [NAT Gateway Console](#) and click **+ Create**.
2. Enter relevant information on the pop-up page. This document creates an NAT gateway as shown below:

**Note:**

The NAT gateway should be deployed in the same region as the function and VPC.

The network to which the NAT gateway belongs should be the VPC where the function is located.

### Create an NAT gateway ✕

Gateway Name

Internet\_for\_SCF

43 more chars allowed

Network

vpc-fbwsyhd8 (test | 10.0.0.0/16)

Region

East China (Shanghai)

Gateway Type

Small-scale (Max concurrent connections 100)

Outbound Bandwidth Cap

100Mbps

?

NAT gateway network fee is calculated according to the outbound bandwidth

Elastic IPs

Create an EIP

+ Bind IP

 Up to 10 IPs can be bound 

?

Gateway Fee

Network Fee

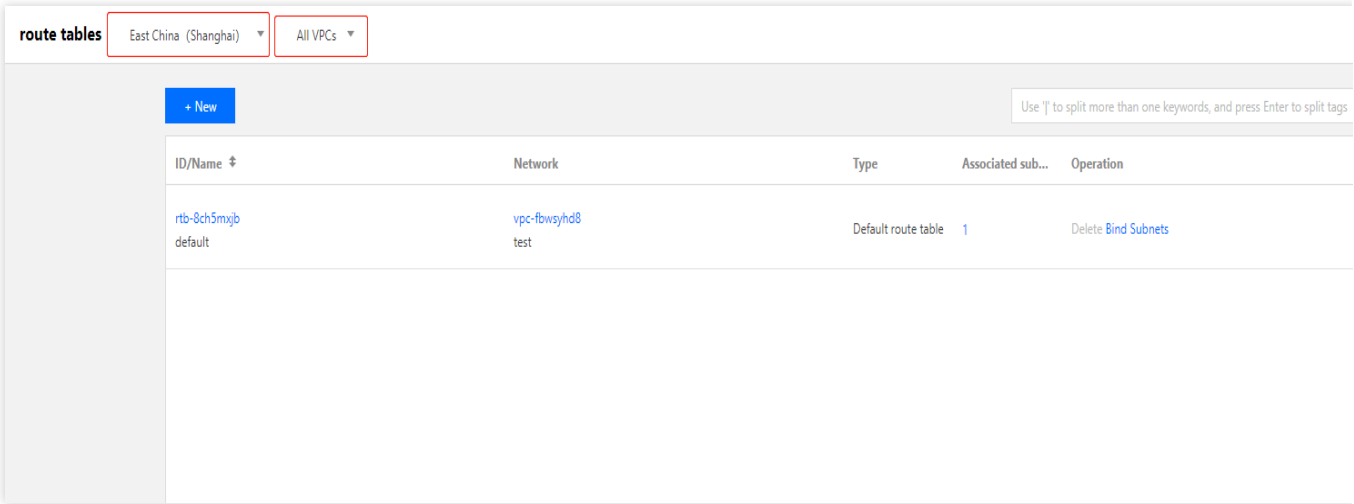
After creation, you need to configure routing rules and point the subnet traffic to NAT gateway  
To get notified about abnormal NAT gateway behaviors instantly, please [Configure Alarms](#).

Create

Cancel

## Creating Routing Policy

Select **Route Table** on the left sidebar in the VPC Console, select the region where the route table is located, and click **+ Create** as shown below:



On the pop-up page, you can choose the corresponding configuration in the following two setting methods to manage the SCF access to the public network.

Enabling SCF to access all public IP addresses

If you want SCF to have access to all public IP addresses, you can configure IP : 0 . 0 . 0 . 0 / 0 in the destination in the routing table and associate the routing table with the created NAT gateway and SCF subnet as shown below:



**Create a route table**

Name

60 more chars allowed

Network

**Routing Rules**

If CVMs in the associated subnet of the route table need to access internet via public gateway, please DO NOT select the public gateway of the associated subnet of the route table. Click [Learn More](#).

Destination	Next hop type	Next hop	Notes	Operation
Local	Local	Local	Released by the system by default, indicating that CVMs in the VPC is connected	
<input type="text" value="0.0.0.0/0"/>	<input type="text" value="NAT Gateway"/>	<input type="text" value="No available NAT gateway"/>	<input type="text"/>	
<a href="#">+ New Line</a>				

**Enabling SCF to access certain public IP addresses**

If you want SCF to have access to certain public IP addresses, you need to add the accessible IP addresses to the routing table and associate the routing table with the created NAT gateway and SCF subnet as shown below:

Create a route table

Name

test-internet

47 more chars allowed

Network

vpc-fbwsyhd8 (test | 10.0.0.0/16)

Routing Rules

If CVMs in the associated subnet of the route table need to access internet via public gateway, please DO NOT select the public gateway of the associated subnet of the route table. Click [Learn More](#).

Destination	Next hop type	Next hop	Notes	Operation
Local	Local	Local	Released by the system by default, indicating that CVMs in the VPC is connected	
<div>14.215.177.0/24</div>	<div>NAT Gateway</div>	<div>nat-6ddubrbr (Internet)</div>	<div></div>	

+ New Line

Create

Cancel

After completing the selection, click **Create**.

# Layer Management Overview

Last updated : 2024-12-02 20:11:41

## Overview

If your SCF service has a lot of dependent libraries or common code files, you can manage them by using the layers in SCF. With layer management, you can place dependencies in layers instead of the deployment package, ensuring that the deployment package remains small in size. For Node.js, Python, and PHP functions, as long as you keep the deployment package size below 10 MB, you can edit the function code online in the SCF Console.

## How It Works

### Creation and binding

Compressed files for layer creation are stored by layer version. Layers on specific versions can be bound to the functions on matched versions. A function can be bound to up to 5 specific layer versions in a certain sequence.

### Loading and access during runtime

When a function bound to a layer is triggered to run and start a concurrent instance, its runtime code will be decompressed to and loaded in the **/var/user/** directory, and the layer content will be decompressed to and loaded in the **/opt** directory.

If the **file** file that needs to be used or accessed is placed in the root directory of the compressed file during layer creation, you can directly access it in the **/opt/file** directory after the decompression and loading are completed. If it is compressed with its folder as **dir/file** during layer creation, you need to access it in **/opt/dir/file** during function execution.

If a function is bound to multiple layers, the decompression and loading sequence of the files in layers will be the same as the binding sequence. They will be sorted in ascending order by serial number. The lower the ranking, the later the loading time, but all files will be loaded before the concurrent instances of the function start. You can use the files in layers during function code initialization.

### Recommended usage

Layers are generally used to store static files or code dependent libraries that rarely change. When storing code dependent libraries, you can directly package available ones and upload the package to a layer. For example, in a Python environment, you can directly package the code package folder of the dependent libraries and create the

package as a layer, and then it can be imported directly through `import` in the function code. In a Node.js environment, you can package the `node_modules` dependent library folder of the project and create the package as a layer, and then it can be imported directly through `require` in the function code.

You can use layers to separate function code, dependent libraries, and dependent static files, so as to keep the function code small in size. You can implement quick upload and update when editing a function in the command line tool, IDE plugin, or console.

## Notes

The files in a layer will be added to the `/opt` directory, which is accessible during function execution.

If your function is bound to multiple layers, these layers will be merged into the `/opt` directory in sequence. If the same file appears in multiple layers, SCF will retain the version in the highest-numbered layer.

## Relevant Operations

For more information on how to use layers in the Serverless Console, please see [Create a layer](#).

# Creating Layer

Last updated : 2024-12-17 14:19:45

This document describes how to create a layer in the SCF console.

## Directions

1. Log in to the [SCF console](#) and select **Advanced capabilities** > **Layer** on the left sidebar.
2. On the **Layer management** page, select the target region and click **Create**.
3. On the **Create layer** page, set the layer information based on your actual needs as shown below:

[←](#) **Create layer**<sup>(i)</sup>

Layer name \*

Please enter the layer name

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

Description

Please enter the description

Up to 1000 characters ([a-z], [A-Z], [0-9], [.,] and spaces)

Submitting method<sup>(i)</sup>

Local ZIP file ▼

Layer code

Upload

Please upload a code package in zip format (up to 50M)

Runtime environment<sup>(i)</sup> \*

+Add runtime environment (0/5)

Tag<sup>(i)</sup>

Tag Key

Tag Value

✕

+ Add

📄 Paste

OK

**Layer name:** Enter the layer name.

**Description:** Enter the descriptive information of the layer as needed.

**Submitting method:** **Local ZIP file**, **Local folder**, and **Upload a ZIP pack via COS** are supported. Select an appropriate method based on your actual needs.

**Local ZIP file:** Click **Upload**, and upload a code pack in ZIP format. It supports a maximum of 50 MB.

**Local Folder:** Click **Upload**, and select a folder. It supports a maximum of 250 MB. This method does not retain the file executable permissions after uploading. If any executable file is included, set the executable permissions locally before uploading them using the ZIP pack method.

**Upload a ZIP pack via COS:** Choose the COS bucket as the event source. The bucket must be in the same region as the function. Enter the COS object file path starting from the root directory of the bucket ("/") to the complete ZIP

code file path.

**Add runtime environment:** Add the runtime environments (up to 5) that are compatible with this layer.

**Tag:** It is the business tag to which the resources of this layer belong. Only the users with matching tags can see the resources. When adding the layer resources to function services, you can only add the layer resources that match the user's tags.

4. Click **OK**, and then you can see the created layer in the layer list.

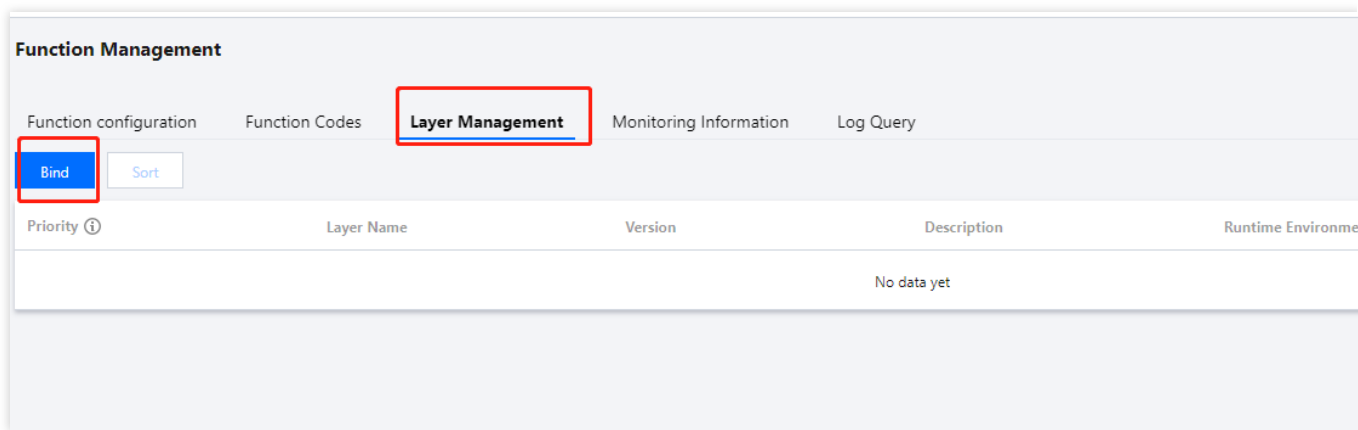
# Binding function to layer

Last updated : 2024-12-02 20:11:41

## Directions

This document describes how to bind a layer in the Serverless console.

1. Log in to the Serverless console and select **Function Service** on the left sidebar to enter the **Function Service** list page.
2. Select the function ID for layer management to enter the function management page.
3. Select the **Layer Management** tab and click **Bind** as shown below:



4. In the **Bind Layer** window that pops up, select the corresponding **Layer Name** and **Layer Version** as shown below:



**Bind a layer** ⓘ

Layer Name

Please select ▼

Layer Version

Please select ▼

Runtime Environment

-

Description

-

OK

Cancel

5. Click **OK**.

# Using layer

Last updated : 2024-12-02 20:11:42

This document describes how to use a layer in the Serverless console.

## Instructions

Files in the layer are all under the `/opt/` directory, which can be accessed through their absolute paths in the function code. In addition, the built-in environment variables of each runtime also include layer paths, so files can be uploaded according to such paths, and then they can be imported through their relative paths in the code.

For the environment variables in Python, Java, and Node.js, see the table below:

Environment Variable	Path
PYTHONPATH	<code>/var/user:/opt</code>
CLASSPATH	<code>/var/runtime/java8:/var/runtime/java8/lib/*:/opt</code>
NODE_PATH	<code>/var/user:/var/user/node_modules:/var/lang/node6/lib/node_modules:/opt:/opt/node_modules</code>

## Procedure

### Node.js

The following takes importing the `cos-nodejs-sdk-v5` dependency from `node_modules` in a layer in the code in the Node.js runtime environment as an example:

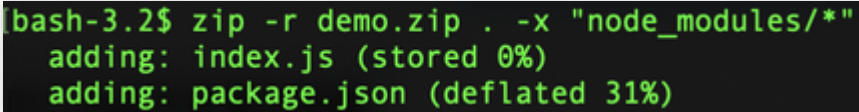
1. Upload `node_modules` to generate a layer as instructed in [Creating layer](#). The local function directory structure is as shown below:

```
[bash-3.2$ ls
index.js      node_modules  package.json
```

2. Package and upload the local function code as instructed in [Deploying Function](#). During the packaging, run the following command to exclude the `node_modules` folder:

```
zip -r package name.zip . -x "node_modules/*"
```

See the figure below:



```
bash-3.2$ zip -r demo.zip . -x "node_modules/*"  
adding: index.js (stored 0%)  
adding: package.json (deflated 31%)
```

3. Bind the created layer to the deployed function as instructed in [Binding function to layer](#).

4. You can import files at the layer in the function after completing the steps above.

```
'use strict'  
var COS = require('cos-nodejs-sdk-v5')
```

#### Note:

As the `NODE_PATH` environment variable already includes the `/opt/node_modules` path, there is no need to specify the absolute path of the dependency. SCF will load the file according to the path specified in the environment variable during execution.

If the file path in the layer and the path included in the environment variable are different, you need to use the absolute path when importing the file.

## Python

The following takes importing the `cos-python-sdk-v5` dependency from a layer in the code in the Python runtime environment as an example:

1. Upload `cos-python-sdk-v5` to generate a layer as instructed in [Creating layer](#).
2. Package and upload the local function code as instructed in [Deploying Function](#). Files that have already been uploaded to the layer don't need to be uploaded again together with the function code.
3. Bind the created layer to the deployed function as instructed in [Binding function to layer](#).
4. You can import files at the layer in the function after completing the steps above.

```
# -*- coding: utf8 -*-  
import cos-python-sdk-v5
```

#### Note:

As the `PYTHONPATH` environment variable already includes the `/opt` path, there is no need to specify the absolute path of the dependency. SCF will load the file according to the path specified in the environment variable during execution.

If the file path in the layer and the path included in the environment variable are different, you need to use the absolute path when importing the file.

# Sample

## Using layer and testing function

1. Go to [scf\\_layer\\_demo](#) and select **Clone or download** > **Download ZIP** to download the demo and decompress it.
2. Create a layer as instructed in [Creating layer](#). Set the parameters as shown below:

**Create Layer**

Layer Name \*   
It supports 2 to 60 characters, including letters, numbers, underscores and hyphens. It must start with a letter and end with a number or letter.

Description   
Up to 1000 letters, numbers, spaces, commas, dots.

Submitting Method

Layer code   
Please upload a code package in zip format (up to 50M)

Runtime Environment \*   
[+Add runtime environment \(1/5\)](#)

**Layer Name:** enter a custom name. This document uses `demo` as an example.

**Submission Method:** select **Local folder** and select and upload the `layer` folder in the folder obtained in [step 1](#).

**Runtime Environment:** select **Nodejs12.16**.

3. Go to the [Function Service](#) page and click **Create** to enter the **Create Function** page.

4. Set the basic information of the function in **Basic Info** on the **Create Function** page and click **Next** as shown below:

Template  
Use demo template to create a function or application

Create from scratch  
Start from a Hello World sample

Use TCR image  
Create a function based on a TCR image

### Basic Configurations

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

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

Function name \*   
It supports 2 to 60 characters, including letters, numbers, underscores and hyphens. It must start with a letter and end with a number or letter.

Region \*

Runtime Environment \*

### Function Codes

**Advanced Configuration** ⓘ Function invocation logs are published to the SCF-specific topic of CLS, which will use the free tier of CLS. See [CLS Billing Details](#)

### Trigger Configurations

Complete

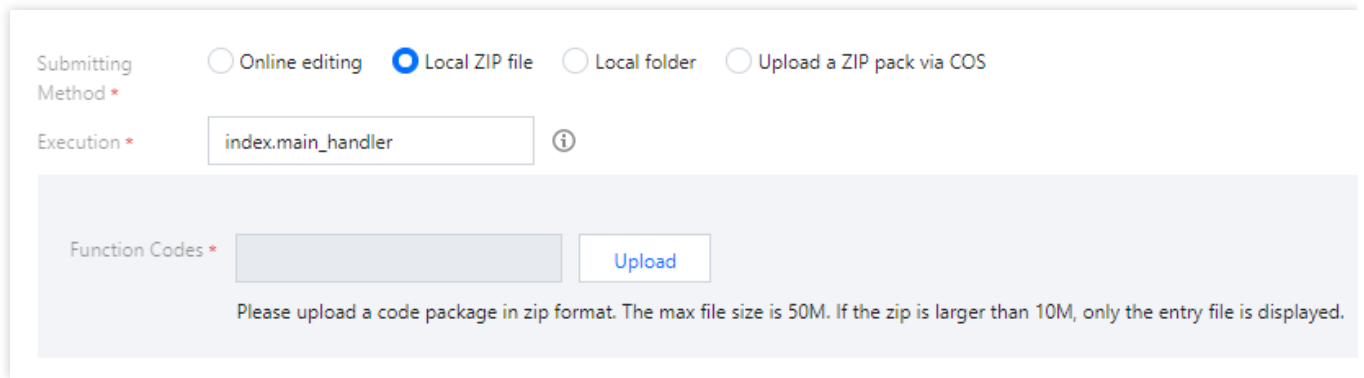
Cancel

**Function Name:** enter a custom name. This document uses `layerDemo` as an example.

**Runtime Environment:** select **Nodejs 12.16**.

**Creation Method:** select **Blank function**.

5. In **Function Configuration**, select **Local folder** as **Submission Method** and select and upload the `function` folder in the folder obtained in [step 1](#).



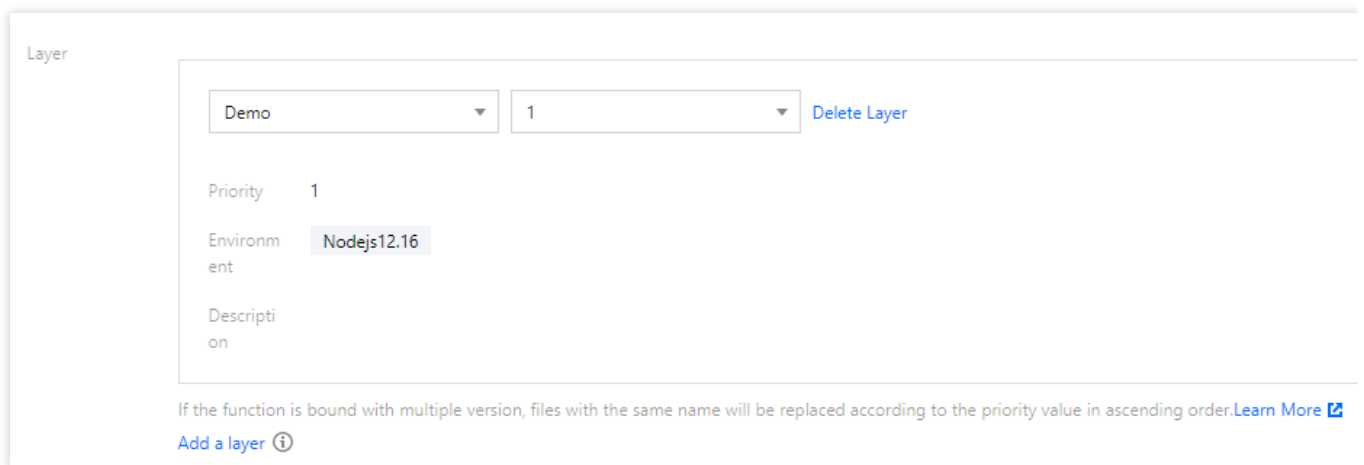
Submitting Method \* ☐ Online editing ☒ Local ZIP file ☐ Local folder ☐ Upload a ZIP pack via COS

Execution \*  ⓘ

Function Codes \*  Upload

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

6. Click **Advanced Settings** and add the function layer in **Layer Configuration** as shown below:



Layer

Demo	1	<a href="#">Delete Layer</a>
------	---	------------------------------

Priority 1

Environment Nodejs12.16

Description

If the function is bound with multiple version, files with the same name will be replaced according to the priority value in ascending order. [Learn More](#) ⓘ

[Add a layer](#) ⓘ

**Layer Name:** select the layer `demo` created in [step 2](#).

**Layer Version:** select v1.

7. Click **Complete** at the bottom to complete the function creation.

8. Select the **Function Code** tab on the **Function Management** page. You can click **Test** at the bottom to view the result as shown below:

Deploy

Upload Method ⓘ

Upload All ▾

Switch to New Ex

Access path <https://service-7adizo19-1300858599.sh.apigw.tencentcs.com/release/> [↗](#)

Test

Test

Request MethodGET ▾

path/

headers

keyvalue

Please enter the keyPlease enter the value

params

Returned result [Learn More](#) [↗](#)

Pending

# Execution Configuration

## Async Execution

Last updated : 2024-12-02 20:11:41

### Use Cases

In single-task computation-intensive scenarios such as audio/video transcoding, massive data ETL, and AI reasoning, the runtime of a single function instance requires more computing power and longer stable execution. If the invoker of the function is jammed to wait for the execution result, this will not only continue to consume the invoker's resources but also raise the high requirements for the stability of the invocation linkage.

SCF provides a new function execution mechanism. You can use the async function execution mode provided by SCF to extend the execution timeout period and solve the problems with existing execution mechanisms.

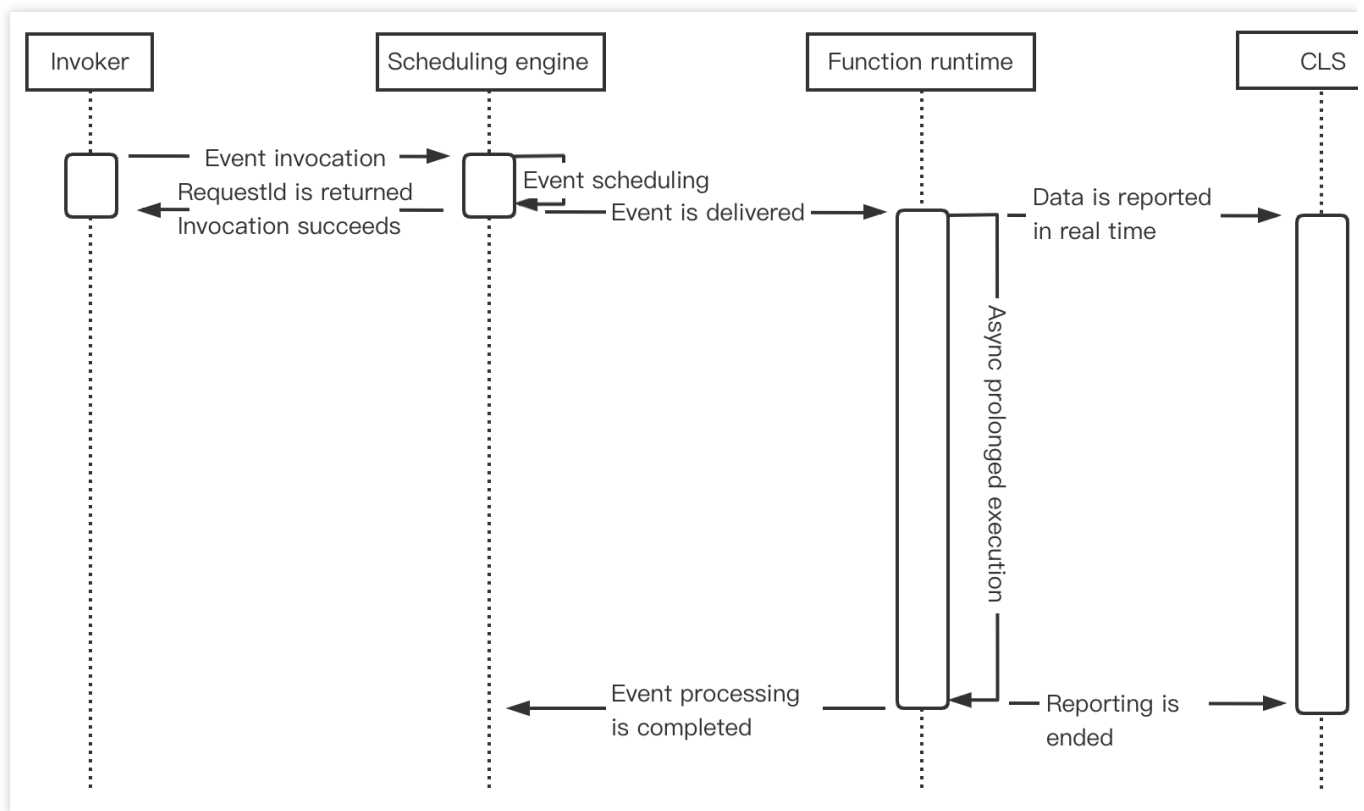
### Execution Mechanism

#### How it works

After async execution is enabled for a function, after an event is invoked by a sync invoker (such as API Gateway) or async invoker (such as COS, CKafka, and Timer), the function will respond to the event asynchronously.

In other words, after event scheduling is completed, the event invocation identifier `RequestId` will be returned immediately, and the invocation operation will be ended, thus eliminating the need for the invoker to wait. When the `RequestId` is returned, the engine will concurrently deliver the event to the function runtime to start the execution of the function logic. After the function enters the async execution status, the execution log will be reported to the log service in real time to provide real-time feedback on the execution of the event executed asynchronously. It works as shown below:





## Notes

Due to differences in the execution mechanisms:

You cannot switch between sync/async execution. You can choose whether to enable the "async execution" feature only when creating a function. This configuration item will be locked and cannot be modified or updated after the function is created.

You cannot retry function execution during async invocation in case of errors.

Any exceptional execution of async functions will trigger instance repossession.

If the event is invoked successfully, the returned message will only contain `RequestId` and the event execution result. You need to configure the function code logic to call back specific APIs or send notification messages by yourself.

The maximum execution duration currently supported for async execution is 24 hours. If you need a longer execution duration, you can [submit a ticket](#) for application.

If you use a function execution role to get the permission to access other components of Tencent Cloud services, the role's key is valid for up to 6 hours. If the actual execution duration is longer, we recommend you use a permanent key. The maximum QPS of asynchronously executed functions is 1,000, and any excess will be limited, resulting in response failures.

## Directions

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select an **empty function** or **function template** to create a function.
4. On the **Function Configuration** page, expand **Advanced Settings** and select **Async Execution**.
5. Click **Complete**.

# Status Trace

Last updated : 2024-12-02 20:11:41

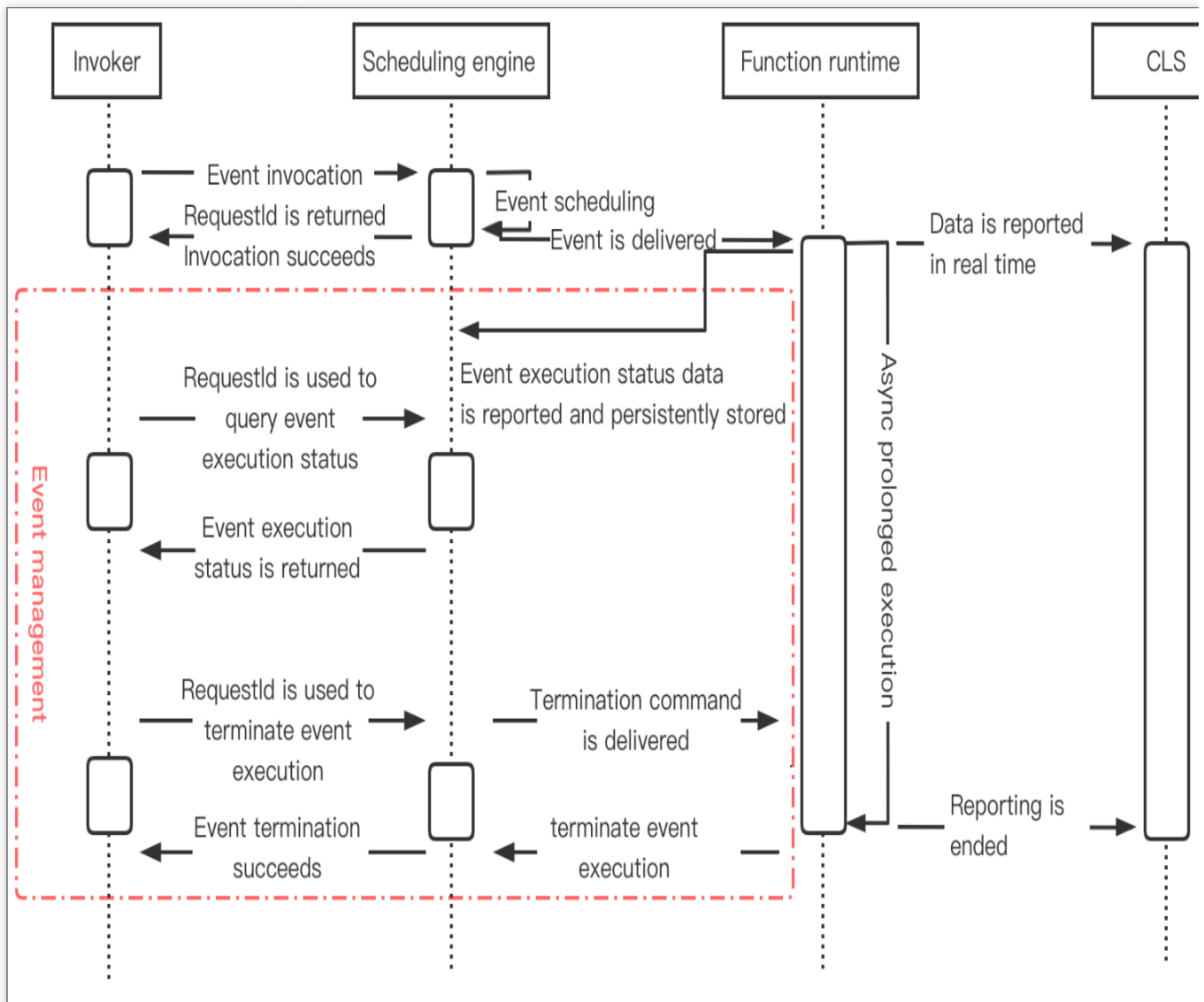
## Use Cases

Asynchronously executed functions are usually used to process a large number of async time-consuming tasks. In order to better manage such tasks, SCF provides a status trace feature, which records and reports the real-time status of event responses and provides event management services such as event status statistics collection and query.

## Execution Mechanism

### How it works

After the status trace feature is enabled for asynchronously executed functions, the platform will start recording and reporting the real-time status of events. It works as shown below:



## Must-Knows

The status data of asynchronously executed events is retained for only 3 days and will be cleared on a rolling basis in a time window of 3 days. If you want to keep all records, you need to periodically pull them and save them to your own storage.

After status trace is disabled, event management services such as recording, collecting, and querying asynchronously executed events will no longer be available, and the generated event status data will be cleared in 3 days.

If the limit on QPS is exceeded, or if your account falls into arrears, the corresponding exception will be returned by the scheduling engine directly after you invoke an event, and no event status records will be generated.

## Directions

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

3. Select **Template** or **Create from scratch** to create a function.
4. On the **Function configuration** page, expand **Advanced settings** and select **Async execution** > **Status trace**.
5. Click **Complete**. After the function is created, you can click **Event Management** to view the list of async events.

The screenshot shows the 'Event Management' page in the Tencent Cloud console. On the left is a navigation sidebar with the following items: Function Management, Trigger Management, Monitoring Information, Log Query, Concurrency Quota, **Event Management** (highlighted with a red box), and Deployment Logs. The main content area is titled 'Event Management' and contains two sections: 'Event Overview' and 'Event List'.

**Event Overview** Metric data displayed are the summary of data in the latest 72 hours.

Running	Invoked successfully	Invoke failed	Invocation stopped
0	0	0	0

**Event List**

Filters: All versions, All Status, Invocation, 2022-01-04 18:01:39 ~ 2022-01-05 18:11:39. Search: Please enter the rec.

RequestId	Status	Version	Event Source	Invocation Time	End Time	Operation
No data yet						

Total items: 0. 20 / page. Page 1 / 1 page.

# Async Event Management

Last updated : 2024-12-02 20:11:42

SCF provides the features of getting the list and status of and terminating asynchronously executed function events for easier event management.

## Note:

Features involved in this document are supported only for [asynchronously executed](#) functions.

## Getting Async Event List and Status

An asynchronously executed function event has the following status:

Running: the event is being executed asynchronously.

Invoked successfully: the event is asynchronously executed successfully with a normal response.

Invocation failed: the event failed to be asynchronously executed with an exceptional response.

Invocation terminated: the user actively terminated the event in progress, and async execution stopped.

## Relevant APIs

API	Description	Documentation
ListAsyncEvents	This API lists the information of an asynchronously executed event. It can query the information by conditions such as `RequestId`, function name, function version, event status, and event invocation/end time. Only data within three days after event tracking is enabled can be queried.	<a href="#">ListAsyncEvents</a>
GetAsyncEventStatus	This API is used to get the async event execution status based on the `RequestId`. The event status will be retained for 72 hours (after the end of the event).	GetAsyncEventStatus

## Note:

When using an API, pay attention to the API call rate limit. We recommend you not call the `ListAsyncEvents` API frequently. To query the execution result of an async event, call the `GetAsyncEventStatus` API instead.

## Terminating async function event

SCF provides two async event termination methods: **terminating invocation** and **sending termination signal**, whose differences and usage are as detailed below:

## Relevant APIs

API	Description	Documentation
TerminateAsyncEvent	This API is used to terminate an asynchronously executed event in progress according to the returned `RequestId`. The default behavior of this API is to terminate invocation. If the `GraceShutdown` parameter is set to `True`, the `SIGTERM` termination signal will be sent to the request. You can listen on the signal and customize the signal processing logic inside the function.	<a href="#">TerminateAsyncEvent</a>

## Terminating invocation

When a function is invoked, SCF will assign an instance to process the function request or event. After the function code is executed and its response is returned, the instance will process other requests. If all instances are running when a request arrives, SCF will assign a new concurrent instance. For more information on instances, see [Concurrency Overview](#).

After the asynchronously executed function event receives the invocation termination instruction, SCF will forcibly stop instance operations and repossess the instance. When the next request arrives, if there are no idle instances, SCF will assign a new instance to process the request.

### Use cases

This method is suitable for scenarios where function execution needs to be stopped in advance, such as infinite loop and execution exception of an asynchronously executed function.

### Notes

Invocation termination will forcibly terminate an instance and trigger instance repossession, which means that cached information in the instance cannot be obtained normally (such as files in the `/tmp` folder). If you want to use this feature, promptly write the files in the instance cache to other persistent storage media to avoid file loss after instance repossession.

## Sending termination signal

When you call the `TerminateAsyncEvent` API and set the `GraceShutdown` parameter to `True`, SCF will send the termination signal `SIGTERM` to the event specified in the input API parameter. You can listen on the signal and customize the processing logic after receiving the signal, including but not limited to function execution termination.

After an asynchronously executed function event receives the `SIGTERM` signal:

If a signal processing function is defined and listened on in the function code, the corresponding signal processing function logic will be executed.

If the signal isn't listened on in the function code, the function process will exit and return the error code 439 ( `User process exit when running` ), indicating that the user process exits).

SCF records the event signal reception conditions into the function execution log:

Received the signal successfully: "[PLATFORM] Signal received successfully" will be logged.

Failed to receive the signal: "[PLATFORM] Signal reception failed" will be logged.

## Use cases

This method is suitable for scenarios where function execution needs to be stopped for business requirements and custom processing logic needs to be run before the stop.

## Usage

The following sample code describes how to use a custom signal processing function to stop function execution after the `SIGTERM` signal is listened on:

## Code deployment

Python

Golang

```
# -*- coding: utf8 -*-
import time
import signal

class GracefulKiller:
    kill_now = False
    def __init__(self):
        # Register signal processing function
        signal.signal(signal.SIGTERM, self.graceshutdown)
    def gracefulshutdown(self, *args):
        print("do something before shutdown.")
        self.kill_now = True

def main_handler(event, context):
    killer = GracefulKiller()

    while not killer.kill_now:
        time.sleep(1)
        print(killer.kill_now)

    print("Graceful shutdown.")
    return("END")
```



```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "os/signal"
    "syscall"
    "time"

    "github.com/tencentyun/scf-go-lib/cloudfunction"
)

type DefineEvent struct {
    // test event define
    Key1 string `json:"key1"`
    Key2 string `json:"key2"`
}

func hello(ctx context.Context, event DefineEvent) (string, error) {
    go graceshutdown()
    sleepNum := 0
    for {
        sleepNum++
        fmt.Println("sleep:", sleepNum)
        time.Sleep(time.Second)
    }
}

// Register signal processing function
func graceshutdown() {
    sigs := make(chan os.Signal, 1)
    signal.Notify(sigs, syscall.SIGTERM)
    sig := <-sigs
    log.Printf("receive signal %s", sig.String())
    //do something before shutdown.
    os.Exit(0)
}

func main() {
    // Make the handler available for Remote Procedure Call by Cloud Function
    cloudfunction.Start(hello)
}
```

## Image deployment

### Python

```
# -*- coding: utf8 -*-
from flask import Flask, request
import time
import signal
app = Flask(__name__)

class GracefulKiller:
    kill_now = False
    def __init__(self):
        # Register signal processing function
        signal.signal(signal.SIGTERM, self.graceshutdown)
    def graceshutdown(self, *arg):
        print("do something before shutdown.")
        self.kill_now = True

@app.route('/event-invoke', methods = ['POST'])
def invoke():
    while not killer.kill_now:
        time.sleep(1)
        print(killer.kill_now)

    print("Graceful shutdown.")
    return("END")

if __name__ == '__main__':
    killer = GracefulKiller()
    app.run(host='0.0.0.0', port=9000)
```

# Extended Storage Management

## Mounting CFS File System

Last updated : 2024-12-02 20:11:42

### Overview

[Tencent Cloud File Storage \(CFS\)](#) provides a scalable shared file storage service that can be used with various Tencent Cloud services such as CVM, TKE, and batch operations. The standard NFS file system access protocol used by CFS offers shared data sources for multiple computing nodes. It supports elastic capacity and performance scaling. Your existing applications can be mounted for use without modification required. As a highly available and reliable distributed file system, CFS is suitable for various scenarios such as big data analysis, media processing, and content management.

CFS is very cost-effective and pay-as-you-go on an hourly basis, so you only need to pay for the actually used storage space. For more information on CFS billing, see [Billing Overview](#).

Tencent Cloud SCF can be seamlessly integrated with CFS. After proper configuration, your functions can easily access files stored in CFS. You can enjoy the following advantages of CFS:

The execution space of functions is unlimited.

Multiple functions can share the same file system to share files.

### Directions

#### Associating authorization policy

**Note:**

To use the CFS service, SCF needs permission to operate on your CFS resources.

Follow the steps below to grant the permission to your account:

1. Associate the `SCF_QcsRole` role with the `QcloudCFSReadOnlyAccess` policy as instructed in [Modifying a Role](#). The result of a successful association is shown below:

If you don't perform this operation for your currently used account, problems such as the failure to save functions and the unavailability of CFS features may occur.

Role Info

Role Name

SCF\_QcsRole

Role Arn

Role ID

Description

Current role is Serverless Cloud Function service role, which will access your other cloud service resources within the permissions of the associated policies.

Creation Time

2020-12-10 08:44:36

Tag

No tag

Permission

Role Entity (1)

Revoke Session

Service

Permissions Policy

Associate Policy

Disassociate Policies

QcloudCFSReadOnlyAccess

Simulate Poli

Policy Name	Description	Session Expiration Time	Association Time	Operation
QcloudCFSReadOnlyAccess	Read-only access to Cloud File Storage (CFS)	-	2022-01-23 10:47:16	Disassociate

0 selected, 1 in total

10 / page

1 / 1 page

2. If the currently used account is a sub-account, request the root account to associate the sub-account with the `QcloudCFSReadOnlyAccess` policy as instructed in [Setting Sub-user Permissions](#). The result of a successful association is shown below:

If you don't perform this operation for your currently used sub-account, problems such as the unavailability of CFS features may occur.

online\_TEST

Sub-user

Edit Info

Account ID

Remarks

Access Method

Console access, Programming access

Tag

No tag

Mobile

Email

Quick Action

Subscribe to Messages

Delete User

Disable User

Quick Login

Permission

Service

Group (0)

Security

API Key

Permissions Policy

Associate Policy

Disassociate Policy

Search for policy

Simulate Policy

Policy Name	Description	Association Type	Policy type	Association Time	Operation
QcloudCFSReadOnlyAccess	Read-only access to Cloud File Storage (CFS)	Associated directly	Preset Policy	2022-12-19 15:01:02	Disassociate
AdministratorAccess	This policy allows you to manage all users under your ...	Associated directly	Preset Policy	2022-07-12 15:24:22	Disassociate

10 / page

1 / 1 page

## Creating a VPC

Build a VPC as instructed in [Building Up an IPv4 VPC](#).

## Creating CFS resources

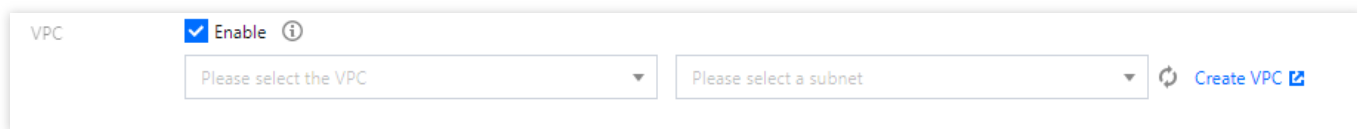
Create a CFS file system as instructed in [Creating File Systems and Mount Targets](#).

### Note:

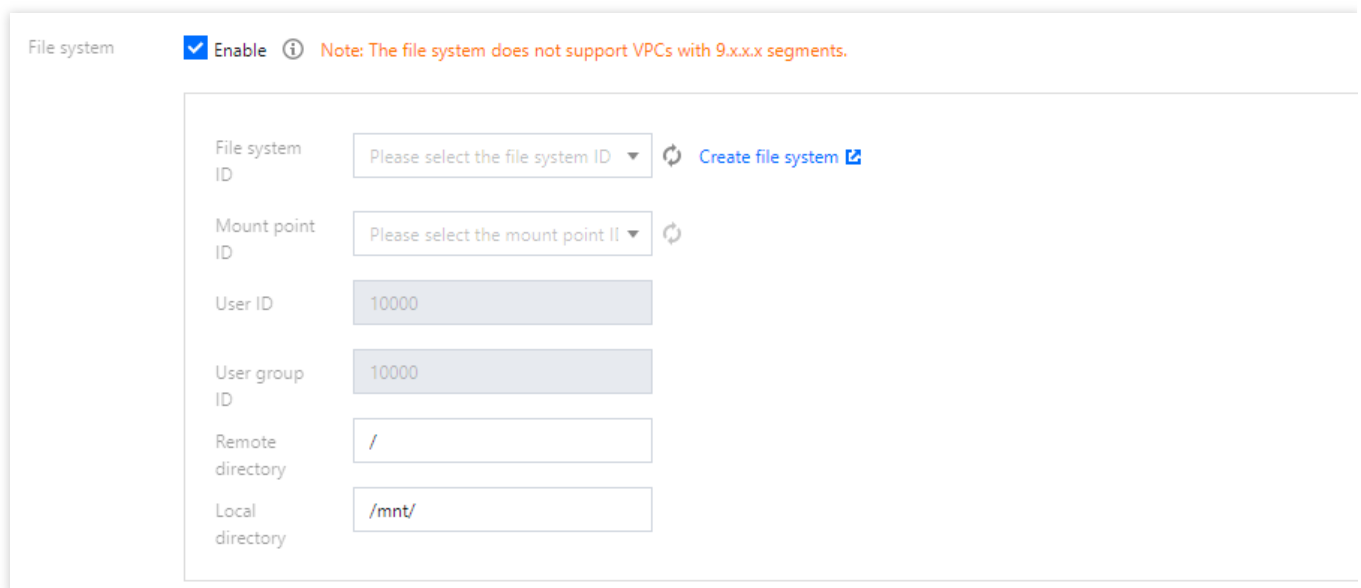
Currently, SCF allows only CFS file systems whose network type is VPC to be added as mount targets. When creating a CFS file system, select the same VPC as that of the target function to enable communication.

## Mounting and using CFS file system

1. Log in to the SCF console and select **Function Service** on the left sidebar.
2. On the **Function Service** page, select the name of the function to be configured.
3. On the **Function configuration** tab of the **Function management** page, click **Edit** in the top-right corner.
4. Check **Enable** for **VPC**, and select the VPC where your CFS file system resides, as shown below:



5. Check **Enable** for **File System**, and enter the following information to mount the file system, as shown below:



**User ID** and **User group ID**: IDs of the user and user group in CFS file system. SCF uses "10000" for both the user ID and user group ID by default to manipulate your CFS file system. Set the file owner and corresponding group permission as needed and ensure that your CFS file system has the required permission. A simple example is to run the `chown 10000:10000 -R /mnt/folder` command. For more information, see [Managing Permissions](#).

**Remote directory**: The remote directory in the CFS file system to be accessed by the function, which consists of a file system and a remote directory.

**Local Directory:** Mount target of the local file system. You can use a subdirectory in the `/mnt/` directory to mount the CFS file system.

**File System ID:** Select the file system to be mounted in the drop-down list.

**Mount Target ID:** Select the ID of the mount target corresponding to the file system in the drop-down list.

6. Click **Save** at the bottom of the page.

You can run the following function code to start using the CFS file system.

```
'use strict';
var fs = require('fs');
exports.main_handler = async (event, context) => {
  await fs.promises.writeFile('/mnt/myfolder/file1.txt',
    JSON.stringify(event));
  return event;
};
```

## Performance test for using the CFS file system on SCF

You can use this [Demo](#) to test how well CFS performs on SCF.

# DNS Caching Configuration

Last updated : 2024-12-02 20:11:42

## Overview

When the client initiates an access request to an address, it will query whether the local DNS cache has relevant records, and if so, it will directly access the corresponding IP; otherwise, it will delegate global query to the recursive server.

As DNS resolution uses the UDP protocol for communication, it is greatly affected by the network environment and may have a delay of several seconds in extreme conditions. In SCF use cases, the domain resolution delay may cause function execution failures due to timeout and affect the normal business logic. When a function is invoked frequently, the DNS server's resolution frequency may exceed the limit, which also will cause function execution failures. SCF offers DNS caching configuration to solve the above problems. DNS caching can improve the domain resolution efficiency and increase the domain resolution success rate by mitigating network jitters.

## Use Cases

This feature is applicable to scenarios where an address is requested in the function code and the function is invoked frequently.

## Directions

As code deployment-based event-triggered functions, HTTP-triggered functions, and image deployment-based functions have different implementation mechanisms, you should enable DNS caching in the corresponding steps:

### Code deployment-based event-triggered function

1. Log in to the [Serverless console](#) and select the target function to enter the function details page.
2. On the function configuration page, click **Edit** in the top-right corner and select **Enable DNS Caching**.
3. Click **Save**.

### HTTP-triggered function

1. Add the following command to the HTTP-triggered function's bootstrap file [scf\\_bootstrap](#) to start the nsd process and enable DNS caching:

```
/var/lang/bin/nsd -f /var/lang/conf/nsd.conf
```

2. Deploy the updated `scf_bootstrap` and the function code together in the cloud. Then, DNS caching will be enabled for new invocations.

## Image deployment-based function

1. Install `nscd` during image creation. Taking CentOS as an example, you can run the following command to install `nscd`:

```
yum install nscd -y
```

2. Update the default `/etc/nscd.conf` file with the following content:

```
#
# /etc/nscd.conf
#
# An example Name Service Cache config file. This file is needed by nscd.
#
# WARNING: Running nscd with a secondary caching service like sssd may lead to
# unexpected behaviour, especially with how long entries are cached.
#
# Legal entries are:
#
# logfile <file>
# debug-level <level>
# threads <initial #threads to use>
# max-threads <maximum #threads to use>
# server-user <user to run server as instead of root>
# server-user is ignored if nscd is started with -S parameters
# stat-user <user who is allowed to request statistics>
# reload-count unlimited|<number>
# paranoia <yes|no>
# restart-interval <time in seconds>
#
# enable-cache <service> <yes|no>
# positive-time-to-live <service> <time in seconds>
# negative-time-to-live <service> <time in seconds>
# suggested-size <service> <prime number>
# check-files <service> <yes|no>
# persistent <service> <yes|no>
# shared <service> <yes|no>
# NOTE: Setting 'shared' to a value of 'yes' will accelerate the lookup,
# but those lookups will not be counted as cache hits
# i.e. 'nscd -g' may show '0%'.
# max-db-size <service> <number bytes>
# auto-propagate <service> <yes|no>
#
# Currently supported cache names (services): passwd, group, hosts, services
```



```
#
# logfile /var/log/nscd.log
# threads 4
# max-threads 32
server-user root
# stat-user somebody
debug-level 0
reload-count 2
paranoia no
# restart-interval 3600
enable-cache passwd no
positive-time-to-live passwd 600
negative-time-to-live passwd 20
suggested-size passwd 211
check-files passwd yes
persistent passwd yes
shared passwd yes
max-db-size passwd 33554432
auto-propagate passwd yes
enable-cache group no
positive-time-to-live group 3600
negative-time-to-live group 60
suggested-size group 211
check-files group yes
persistent group yes
shared group yes
max-db-size group 33554432
auto-propagate group yes
enable-cache hosts yes
positive-time-to-live hosts 300
negative-time-to-live hosts 0
suggested-size hosts 211
check-files hosts no
persistent hosts no
shared hosts yes
max-db-size hosts 8388608
enable-cache services no
positive-time-to-live services 600
negative-time-to-live services 3
suggested-size services 211
check-files services yes
persistent services yes
shared services yes
max-db-size services 33554432
enable-cache netgroup no
positive-time-to-live netgroup 28800
negative-time-to-live netgroup 20
```

```
suggested-size netgroup 211
check-files netgroup yes
persistent netgroup yes
shared netgroup yes
max-db-size netgroup 33554432
```

3. Add the following command to the bootstrap file `scf_bootstrap` to start `nscd` process and enable DNS caching.

Taking CentOS as an example, add the following command to the bootstrap file:

```
${PATH}/nscd -f /etc/nscd.conf
```

**Note:**

`${PATH}` is the absolute path where `nscd` is installed.

# Running Instance Management

## Instance Level Monitoring

Last updated : 2025-06-17 16:37:33

### Feature Overview

The function platform now supports the "instance-level monitoring" feature. Through instance-level metrics, you can view core metric information such as vCPU usage, memory usage, and instance network conditions. This article introduces the use cases, definitions, metric information, and configuration methods of instance-level metrics.

### Use Cases

Monitor CPU, memory, and network metrics of a single instance in real-time to accurately diagnose resource usage and identify matching issues between specification and workload.

Continuously track the instance lifecycle status, fully record key events such as startup, destruction, and abnormal exit, and ensure the observability of running health status;

Based on multidimensional data correlation analysis, quickly locate the root cause of failures, effectively distinguish between code defects and environment-related issues, and provide a basis for decision-making for failure recovery;

### Instance-Level Monitoring Metric Descriptions

Instance-level metrics are performance metrics at the function instance dimension. They perform real-time monitoring and performance data collection on function instances, provide visual displays, and offer you an end-to-end monitoring and troubleshooting path for function instances.

Instance-level metrics support the following dimensionalities.

Instance dimension: Metrics of a specific certain function instance.

Function dimension or function version / Alias dimension: Refers to the aggregation performed by the function dimension. For example, if there are two instances of Function A executing at the same time, then the vCPU metric in the function dimension is the maximum value of vCPU usage among these two instances (awaiting release).

Metric Meaning	Meaning of Metric	Unit	Dimension
vcpu usage	Instance vCPU consumption. (vCPU quota, maximum vCPU, average vCPU)	vcpu	Instance Dimension
vcpu utilization	vCPU usage. Represent the actual number of vCPUs in use, which may exceed 100%. (Maximum utilization, Average utilization rate)	%	Instance Dimension

Memory Usage	Consumed memory of the instance. Unit: MB. (Memory quota, Maximum used memory, Average used memory)	MB	Instance Dimension
Memory Usage	Memory Utilization Rate. That is, actual consumed memory/total memory (Maximum Utilization Rate, Average Utilization Rate)	%	Instance Dimension
Network Traffic	Inbound Traffic Traffic received by the cloud function instance since its start-up Outbound Traffic Traffic sent by the cloud function instance since its start-up	KBytes	Instance Dimension
Bandwidth	Inbound Bandwidth Traffic bandwidth received by the cloud function instance since its start-up Outbound Bandwidth Traffic bandwidth sent by the cloud function instance since its start-up	Mbps	Instance Dimension

## Configure Instance-Level Metrics

1. Log in to the SCF console and select Function Service in the left sidebar.
2. On the list page of "Function Service", enable log delivery when creating/updating a function.
3. Perform code testing, trigger a function call.
4. On the Function Management page, select **Running Instances**, click **Monitoring**.
5. In the pop-up, you can check corresponding monitoring metrics. Click in the upper right corner to configure alarms.

# Running Instance Command Line Operation

Last updated : 2025-04-30 19:16:32

## Feature Overview

You can log in to the function instance within the SCF console and execute corresponding command line operations. This article introduces how to log in to the function instance in the console and run corresponding commands.

## Use Cases

**Running Instance:** A running instance is the minimum unit used by SCF to run functions. Your requests are processed through function instances. Before a request starts execution, SCF assigns the most appropriate instance for each request. Pay-as-you-go instances are frozen after processing requests. If they do not process requests for a period of time (usually 3 to 5 minutes), they are automatically destroyed. The running instance command line operation feature supports executing specified commands in the actual runtime environment of the instance, such as logging in to the instance to view instance environment information.

## Feature Description and Limitations

1. Instance command line operations can only be performed on instances in a healthy status, including resident instances with preset concurrency and instances in pay-as-you-go mode. If a pay-as-you-go instance is released due to idle timeout or is about to be destroyed because it is in an unhealthy state, you may fail to log in to the instance to perform operations.
2. The request for instance command line operation does not occupy instance concurrency. Therefore, even if the instance concurrency of the function is set to 1, you can execute function invocation and instance command line operation simultaneously.
3. An instance command line operation is regarded as a function invocation. As long as the WebSocket connection established by the instance command line operation request is not disconnected from the function instance, the function instance will remain in an active state and follow the same measurement rules as function invocation. When operating through the console, if there is no data transfer on the console log-in instance interface, the function instance will be disconnected by default after being idle for 10 minutes.

### Note:

Initiating an instance command line operation on an instance that is processing online requests may cause tasks being executed on the instance to fail due to changes in the online environment, directly affecting the success rate of subsequent tasks on the instance. If the request execution fails due to the instance command line operation, it will not be counted in the product SLA.

## Operation Steps

1. Log in to the SCF console and select Function Service in the left sidebar.
2. On the "Function Service" list page, when creating/updating a function, you need to enable log delivery.
3. Perform code testing and initiate a function call.
4. In the "Function Management" page, select **running instances**, click **Log In**.
5. Enter the corresponding command line and check the instance's operating status.

# Resource Managed Mode Management

Last updated : 2024-12-02 20:11:42

## Overview of Function Resource Managed Mode

The function resource managed mode determines the resource pool for the SCF runtime. By default, upon enabling the function service, the platform allocates a public cloud resource pool for each region. This resource pool, composed of underlying machines, manifests as a 128 GB function concurrency quota. You can also enhance the concurrency quota of a region or even a namespace by purchasing packages. The platform will automatically allocate matching machines based on the new concurrency limit, ensuring the smooth operation of functions.

To better cater to your needs in various service scenarios, we now support a custom resource managed mode for functions. This allows functions to run on your specified infrastructure, such as public cloud TKE clusters, hybrid clouds, and IDCs.

The platform has now launched the K8s resource managed mode to support the execution of functions in your own TKE clusters, thereby accelerating business development using functions on a unified cloud-native resource base. Gradually, we will iterate to support more cloud-native infrastructures such as hybrid clouds.

## Types of Function Resource Managed Modes

SCF supports two types: the **Default Resource Managed Mode** and the **K8s Resource Managed Mode**.

In the **Default Resource Managed Mode**, functions run in the public cloud resource pools under each region of the function platform. The function platform fully controls the supply and scheduling of underlying machines. You only need to focus on the actual service scale requirements and ensure service operations by adjusting the concurrency quotas of the function.

In the **K8s Resource Managed Mode**, functions run in your specified K8s cluster. You manage the resource supply in the K8s cluster, while the function platform fully controls the request scheduling of the functions, intelligently calling them within the given resource pool to fully utilize resources.

## K8s Resource Managed Mode

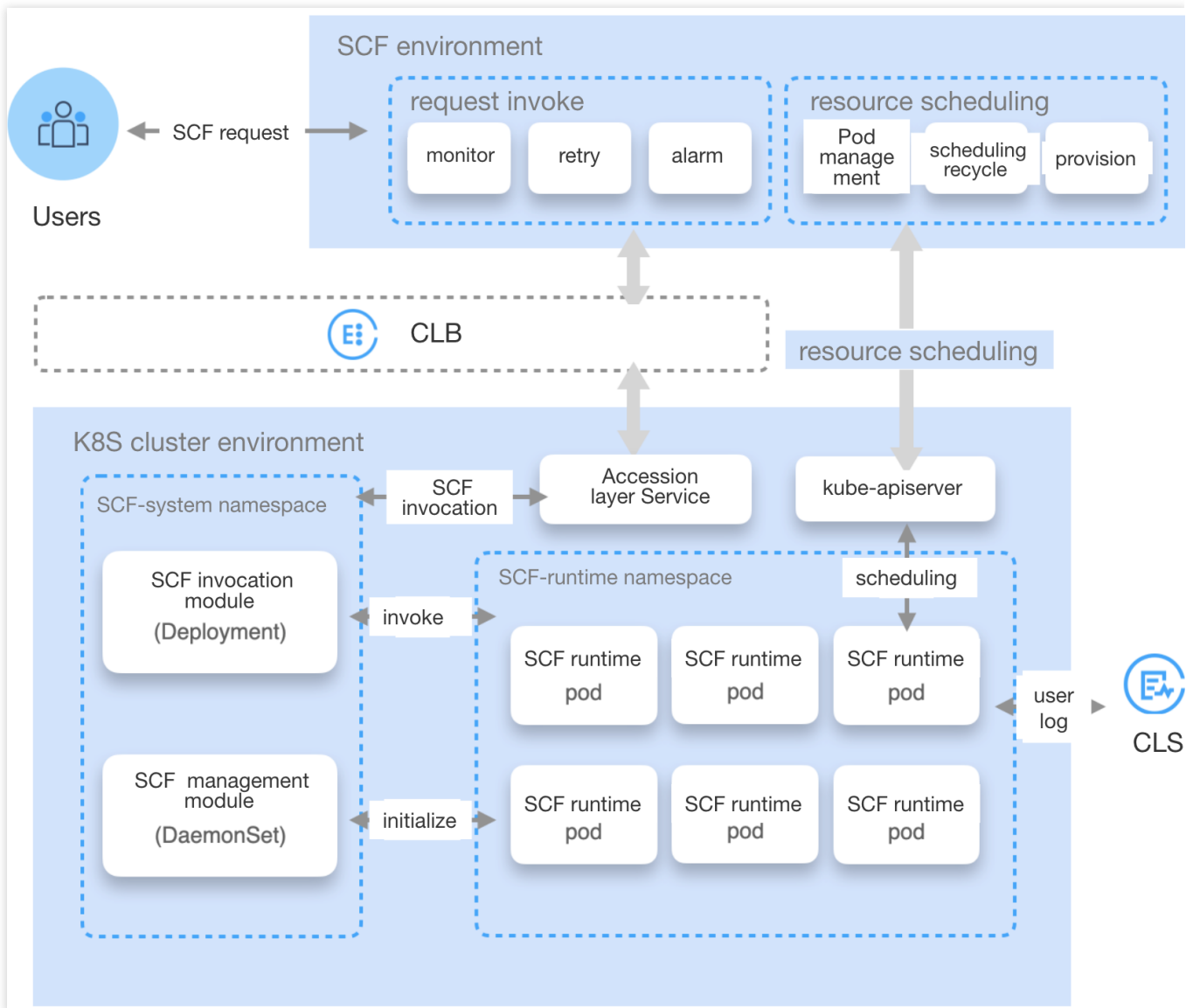
### Overview

In the K8s resource managed mode, you can select a TKE cluster as the computing resource pool for the function. All function request calling will be scheduled to this resource pool, and no fees will be generated on the function side. Currently, TKE cluster native nodes and ordinary nodes are supported, while super nodes are not.

In terms of usage, you only need to configure the resource managed mode as K8s in the function namespace and bind a TKE cluster to enable this mode. All function requests in this namespace will be scheduled to the bound TKE cluster.

## Operating Principle

The scheduling principle of functions under the K8s resource managed mode is illustrated in the following diagram:



### TKE Cluster Initialization

Once you have specified a TKE cluster and function runtime namespace for the function namespace, the platform will automatically create an scf-system namespace in the cluster, along with daemonset for deploying and managing function code metadata, request forwarding Pod, private CLB service components, and so on.

### Function Request Scheduling Lifecycle

The function request sent by the user will first be directed to the request calling entry in the function environment. The function scheduling management module will then analyze whether there are idle function runtime Pod resources available for execution in the TKE cluster:



1. If there are no idle resources, a scheduling request will be issued to the TKE cluster via the resource scheduling module to prepare function runtime Pod resources. Once the Pod is ready, the function management module in the TKE cluster will prepare the function code and other metadata, and finally report the newly added runtime resources to the scheduling management module in the function environment.
2. If there are idle resources, the request will be forwarded to the access layer Service in the TKE cluster via the private CLB. Then, the function calling module in the cluster will send the request to the function runtime Pod, entering the function execution phase. During the function execution, logs will be reported in real time to the user's CLS log system. After the function execution ends, the execution results, monitoring metrics, and other information will be sent back to the request calling module in the function environment.

## Advantages

Compared to the default resource managed mode, the K8s resource managed mode offers the following advantages: Functions can run in your specified K8s cluster, offering greater flexibility and control, and enabling better cost management and stronger infrastructure resource management.

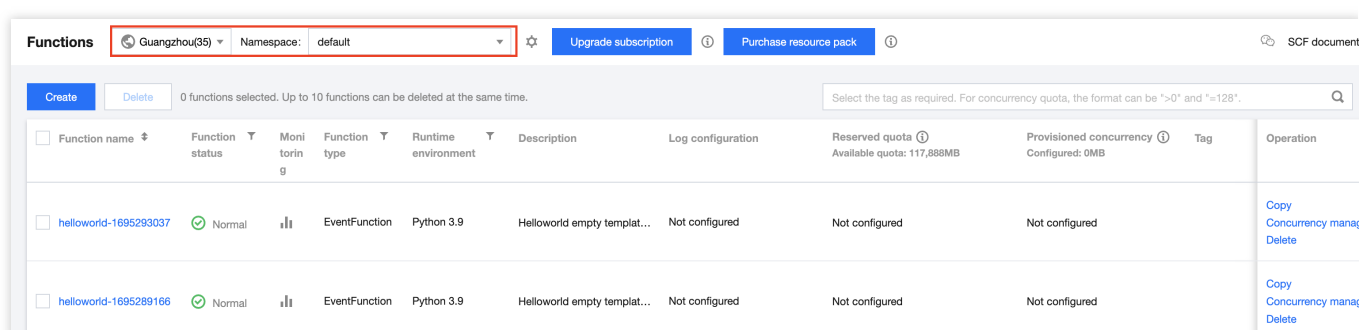
The proactive scheduling mechanism of the function can significantly enhance the resource utilization of your K8s cluster. It not only improves R&D efficiency through the function development experience, but also reduces the resource waste, truly achieving the cost reduction and efficiency enhancement.

After functions are integrated with the K8s ecosystem, a full-stack cloud-native research and development system and service governance mechanism can be achieved. This brings advanced development experiences to service developers and higher availability guarantees for online services.

## Directions

### Creating Function Namespace and Binding it to the TKE Cluster

1. Log in to the [Serverless Console](#) and click **Functions** in the left sidebar.
2. At the top of the Function Service page, select the region where you wish to create the function, and click on the ⚙️ next to the namespace to enter Namespace Management, as shown in the figure below:



3. In the **Namespace** pop-up window, click **Create a Namespace** to navigate to the namespace creation page, as shown in the figure below:

### Create namespace

Namespace

Please enter the namespace

Description

Enter the namespace description

Resource hosting mode

K8s

SCF now supports a custom resource hosting model. With the K8s resource hosting model, functions can be executed in a user's K8s cluster, allowing users to use functions on a unified cloud-native resource base.  
In K8s resource hosting mode, you need to select a TKE (K8s) cluster as the compute resource pool for SCF. All SCF invocations are scheduled to this pool. No fees are incurred on the SCF side. Note that super nodes cannot be added to this pool. See "Resource Hosting Mode".

TKE cluster

Select a cluster

Create TKE cluster

Create SCF add-ons (such as daemonset, intranet clb) in the scf namespace of the specified cluster

Cluster namespaces

Select a cluster first.

This will create a function runtime Pod in the specified namespace.

Subnet for SCF

Select a cluster

Select a cluster first.

SCF takes an IP in this subnet to create a private CLB, which is used to receive SCF requests and forward them to the TKE cluster.

### Advanced settings

Function directory

/var/lib/scf

The temp storage location for function codes, layer codes and function execution logs

Support service port

38000

SCF support service listens to this port for function schedule linkage.

NodeSelector

☒ Disable

☐ Custom scheduling rules

The function can be dispatched to the node that meets the expected Label according to the scheduling rules.[Guide for setting workload scheduling rules](#)

Create

Cancel

4. In **Resource Managed Mode**, select K8s. If this is your first time, a pop-up window for TKE role authorization will appear. Follow the instructions to complete the authorization before proceeding to the next step.

5. In **TKE Cluster**, select the TKE cluster and the namespace under that cluster. The platform will create function service support components such as daemonset and private CLB under the scf-system namespace of the specified cluster, and will create function runtime Pods under the specified namespace. **Please ensure that there are nodes**

in the selected TKE cluster, and that the node types are normal and native nodes, to ensure that the initialization process is completed smoothly.

6. In **Function VPC Subnet**, specify the subnet. The platform will consume an IP address under this subnet to create a private CLB as the function request entry, enabling function request forwarding to the TKE cluster. **Please note that the subnet does not support the 9.x.x.x IP range.**

7. In addition to the basic configuration items above, you can also configure the following items as needed:

**Function Directory:** Specify a path on a TKE cluster node for temporary storage of function codes, layer codes, and logs generated during the function execution process.

**Support Service Port:** Specify an available port number. The function support service will listen on this port to implement the function scheduling link.

**NodeSelector:** Function instances can be scheduled to nodes with expected labels based on the scheduling rules. For more details, see the following section [Setting the Scheduling Policy for Function Instances in the TKE Cluster](#).

**Taints and Tolerations Scheduling:** Function instances can be scheduled to nodes with expected taints based on the scheduling rules. For more details, see the following section [Setting the Scheduling Policy for Function Instances in the TKE Cluster](#).

8. Click **Create**, and then in the pop-up confirmation window, click **Continue**. This will initiate the function support component initialization process in the TKE cluster, which will take approximately 20 seconds. Upon completion, you will see the status updated to Normal in the **Namespace Management** page, as shown below:

Namespace management						×
Instance ID...	Hosting ...	Resource pool	Status	Descript...	Operation	
default	Default	Default resource pool	Creating			
aa	Default	Default resource pool	Normal	bb	<a href="#">Manage</a> <a href="#">Delete</a>	
<a href="#">Add namespace</a>						
<a href="#">Close</a>						

9. Switch to the created function namespace to creat and use functions.

## Setting the Scheduling Policy for Function Instances in the TKE Cluster

NodeSelector

☐ Disable
 ☒ Custom scheduling rules

The function can be dispatched to the node that meets the expected Label according to the scheduling rules.[Guide for setting workload scheduling rules](#)

Up to 63 characters, including lowercase letters, numbers, slash ("/") and hyphens ("-"). It can not begin with slash, and support prefix. For more information, please see [Resource Details](#).

Label key can contain only letters, numbers, and delimiters letters ("-", "\_", "."). It must start with letters or numbers, and end with letters or numbers.

[Add](#)

Blemish tolerance scheduling

☐ Disable
 ☒ Enable

[Add](#)

Tag name	Condition	Tag value	Effect	Duration (Second)
----------	-----------	-----------	--------	-------------------

By setting the NodeSelector and taints and tolerations scheduling policies, you can specify the scheduling of function instances within the TKE cluster. This allows for more effective utilization of resources within the cluster. For more details, see [Container Service - Proper Resource Allocation](#). The following application scenarios exist:

Run function instances on specified nodes.

Run function instances on nodes within a specific scope (the scope can be attributes such as availability zones, and machine types).

## Prerequisites

The scheduling rule is set in the advanced settings of the workload, and the Kubernetes version of the cluster is 1.7 or higher.

To ensure that your Pods can be scheduled successfully, please make sure that the node has resources available for container scheduling after the scheduling rule is set.

When using the custom scheduling feature, it is necessary to set corresponding labels or taints for the nodes. For more details, please refer to [Setting a Node Label](#) and [Setting a Node Taint](#).

## Setting the Scheduling Rule

### NodeSelector

Custom scheduling rules can be used to match node labels and schedule function instances to specified nodes. If an affinity condition is met during scheduling, it is scheduled to the corresponding node. If no node satisfies the condition, the scheduling fails.

For more details, please see [K8s Node Affinity](#).

### Taints and Tolerations Scheduling

Through custom scheduling rules, node taints can be tolerated, allowing function instances to be scheduled onto specified nodes.

For more details, see [K8s Taints and Tolerations](#).

# Near-Offline Resource Hosting Model

Last updated : 2025-06-17 16:36:30

## Overview

The Near Offline Resource Hosting Model is a low-cost serverless computing solution provided by Serverless Cloud Function (SCF). It intelligently schedules a pool of bidding instance resources across global multiple regions, significantly reducing computing costs while maintaining the full features of serverless. This mode supports functions to run in the near offline cluster on the platform, and with the aid of a dynamic scheduling algorithm, automatically assigns requests to the optimal resource nodes, achieving cost-effective computing services.

### Notes:

The resource pool has a risk of passive release. It is recommended for near-offline task scenes with high fault tolerance. A retry mechanism needs to be considered to ensure business continuity.

The concurrency limit of the function can be adjusted by contacting the function side to enhance the overall concurrency capability.

## Application Scenarios

The new model launched by the function platform is suitable for nearline scenarios (such as Kafka task queues) and offline scenarios (such as dial testing and stress testing). Tasks are submitted through the function platform's trigger. The platform manages the queue management, resource supply, and result reporting of task execution throughout the process. It ensures high availability and achieves an overall cost lower than that of bidding instances.

### Nearline Scenarios

Generally, data is processed quasi-real-time based on the message queue. It can respond in seconds or minute-level, without requiring immediate returned results, for example, media transcoding, data processing, model inference, message notifications.

### Offline Scenarios

High computing throughput, bulk initiation, no requirement for immediate processing, with a certain flexible expectation for processing time. High resource usage rate during operation. For example, stress tests/dial tests, data archiving, automation (RPA), other scheduled tasks.

## Billing Mode

When the namespace type is "Nearline Resource Pool Type", it supports [pay-as-you-go \(postpaid\)](#). The original resource usage fee under this namespace will be split into VCPU usage fee and memory usage fee. Other billing items and the hosting of the default resource pool remain unchanged.

## Product Pricing

The cost of SCF consists of the following components. Pricing is as follows:

VCPU usage fee: 0.00000124 USD/VCPU\*second (0.0124 USD/10,000 VCPU\*seconds)

Usage fees for memory: 0.00000012 USD/GB\*s (0.0012 USD/10,000 GB\*s)

API call fee: 0.002 USD per 10,000 calls.

public network outbound traffic fee: Different pricing in each region. In the Chinese mainland, it is 0.12 USD/GB.

Idle provisioned concurrency fee: 0.00000847 USD/GBs (0.0847 USD/10,000 GBs). Please refer to [billing detail](#) and [billing example](#).

Web functions and event functions share the same pricing. For web functions using the default trigger, additional response traffic for web functions will be incurred. For details on web function billing, see [Web Function Billing Description](#).

## Features and Advantages

Compared with the default resource hosting mode, the Near Offline Resource Hosting Model has following strengths:

**Extremely low cost expenditure:** By leveraging the bidding instance resource pool across global multiple regions, computing costs can be reduced by over 90%. It supports the time-sharing strategy, automatically utilizes the tidal characteristics of resources during idle periods such as nighttime, retains the pay-as-you-go mode, and avoids resource waste due to idleness.

**Complete Serverless experience:** Zero modification for business development; APIs and development methods fully compatible with the default resource pool. **Retained maintenance-free features:** Basic capabilities such as automatic monitoring, logs, and alarms are fully inherited. **Intelligent auto scaling:** Supports scaling out in seconds and can handle tens of thousands of concurrent requests in a single region.

**Intelligent resource scheduling:** Dynamic routing algorithm: Automatically select the best resource node (based on cost/delay/stability); **Hybrid deployment support:** Can be used in conjunction with the standard resource pool in proportion; **Automatic compensation for interruptions:** Built-in dual guarantees mechanism of request retry and dead letter queue.

## Operation Steps

### Create and Bind a Function Namespace to Nearline/Offline Resource Pools

1. Log in to [Serverless Console](#), click **Function Service** in the left sidebar.
2. On the Function Service page, select the expected region for function creation at the top, click the ⚙ icon on the right of the namespace, and enter namespace management. As shown below:
3. In the **namespace** pop-up window, click **add namespace** to enter the namespace creation page. As shown below:
4. In the **Resource Hosting Mode** options, select Near Offline and complete the creation.

---

5. Switch to the created function namespace to create a function and start using it.