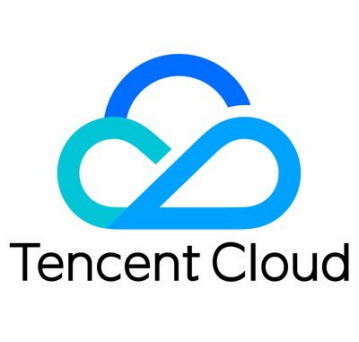
Serverless Cloud Function
Operation Guide



Contact Us

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

Contents

Operation Guide
Quota Management
Quota Limits

Last updated: 2023-09-27 17:13:38

For each user account, the SCF has a certain quota limit.

User Account Quota Limits

Content	Default Quota Limit
Total function code size per region	100GB
Total function concurrency quota per region	128,000 MB(Guangzhou, Shanghai, Beijing, Chengdu, and Hong Kong (China))
	64,000 MB(Mumbai, Singapore, Tokyo, Toronto, Silicon Valley, Frankfurt, Shenzhen Finance, and Shanghai Finance)
Number of namespaces per region	5
Total concurrent function quota per namespace	You can purchase function packages 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 character length of the namespace name + function name should not exceed 118.
Maximum code size (including bound layers) per function (version) before compression	500MB
Maximum number of same-type triggers per function	10
Maximum environment variable size per function	4KB
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: Only one function can be bound to the same event and prefix/suffix rules in a single COS bucket.

Function Runtime Environment Limits

Content	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 `/tmp` directory	512MB
Timeout period	Minimum: 1 second, maximum: 900 seconds
Number of file descriptors	1024
Total processes and threads	1024
Sync request event size	6MB
Sync request response size	6MB
Async request event size	128KB

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: 2023-09-28 09:47:13

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

Content	Solution
The number of namespaces in each region has reached its maximum limit.	You can submit a ticket to increase the quota limit.
The number of functions in the namespace has reached its maximum limit.	Each region can support multiple namespaces, and the function quota of other namespaces can be utilized as a priority. If the number of namespaces and functions in the current region have both reached their maximum limits, you can submit a ticket to increase the quota limit. Cloud Development manages quotas through packages. Please refer to Cloud Development Product Pricing to understand the quota of each package and increase the quota by upgrading the package.
The number of triggers for a single function has reached its maximum limit.	It is suggested to refine the granularity of the function business and solve the problem of the single function trigger limit by binding triggers to multiple functions separately. If the business needs cannot be divided, you can submit a ticket to increase the quota limit.
The total concurrency quota for functions in each region has reached its maximum limit.	Please try adjusting the total concurrency quota in the current region on the function concurrency quota page.
The function initialization timeout period has exceeded the limit.	You can submit a ticket to increase the quota limit.

Function Management
Function Overview

Last updated: 2023-09-27 17:14:32

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

Region

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 specific namespace in a certain region. Each region has a default `default` namespace, and users are allowed to create new namespaces. Once created, the namespace name cannot be modified.

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 types of functions: `event` functions and `Web` functions.

- 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.

Running 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:

- Single-segment format **"[filename]"** is used for Go environment. For example: "main".
- Two-segment format **"[filename].[function name]"** is used for Python, Node.js, and PHP environments. For example: "index.main_handler".

> **Description**
> Please note that FileName does not include the file name extension, and FunctionName is the name of the entry function. Ensure that the file name extension matches the programming language. For example, for Python programming, the file name extension is `.py`, and for Node.js programming, the file name extension is `.js`.

- Three-segment format **"[package].[class]::[method]"** is used for Java environment. For example: "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

Set the specifications corresponding to the resource type, such as different memory configurations for CPU, different card types for GPU, etc. For more details, see Function Computing Power Support.

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, please 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 Roles

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

Through Async Invocation Configuration, you can set the retry policy for asynchronous invocation scenarios. You can also configure a Dead Letter Queue to collect error event information and analyze the cause of failure.

Application performance monitoring

Once enabled, SCF will report the basic runtime of the function to the specified Application Performance Monitoring (APM) system. You can also embed points in the function code for custom reporting, helping you track and monitor the execution of the function. For more details, see Application Performance Monitoring.

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.
- Start/Stop Trigger: Temporarily halts the triggering of cloud functions by event sources through the activation or deactivation of triggers.

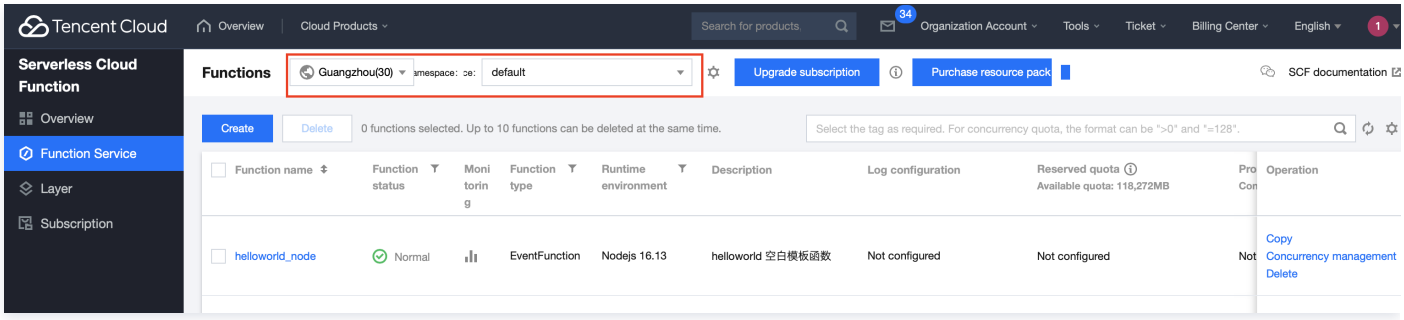
Creating Function

Last updated: 2023-09-27 17:15:14

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

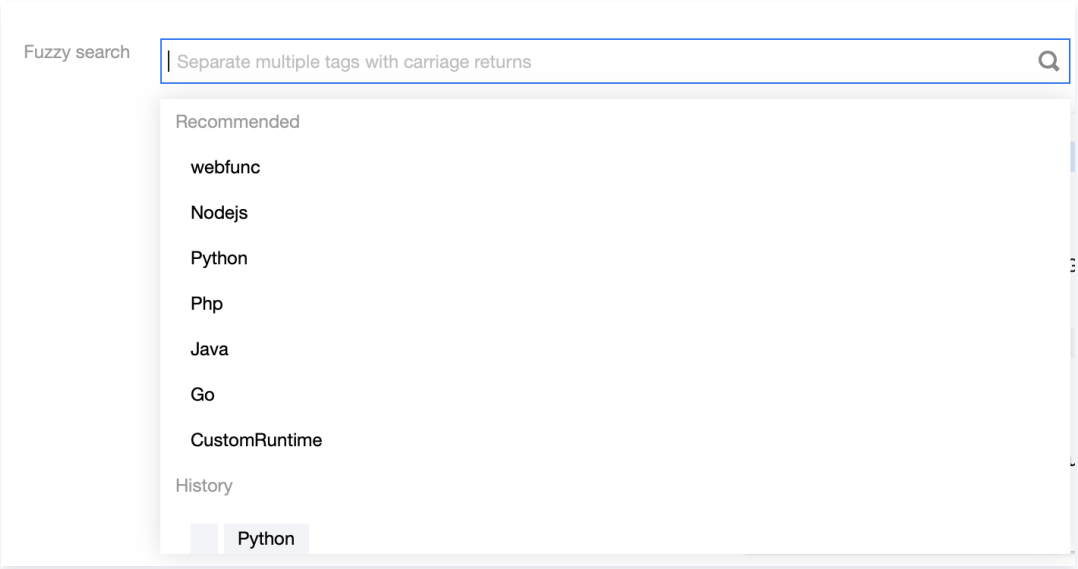
1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
2. At the top of the Function Service page, select the desired region and namespace for function creation, then click **Create** to initiate the function creation process. As shown in the figure below:



3. On the "Create Function" page, you can choose the method of creating a function based on your actual needs.
   - **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

1. Add tag queries to the "Fuzzy Search" in the template. As shown below:



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**: By default, the cloud function uses UTC time. You can modify this by configuring the TZ environment

variable. After you select a time zone, the corresponding TZ environment variable for that time zone will be automatically added.

Create from scratch

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**: By default, the cloud function uses UTC time. You can modify this by configuring the TZ environment variable. After you select a time zone, the corresponding TZ environment variable for that time zone will be automatically added.

Use TCR image

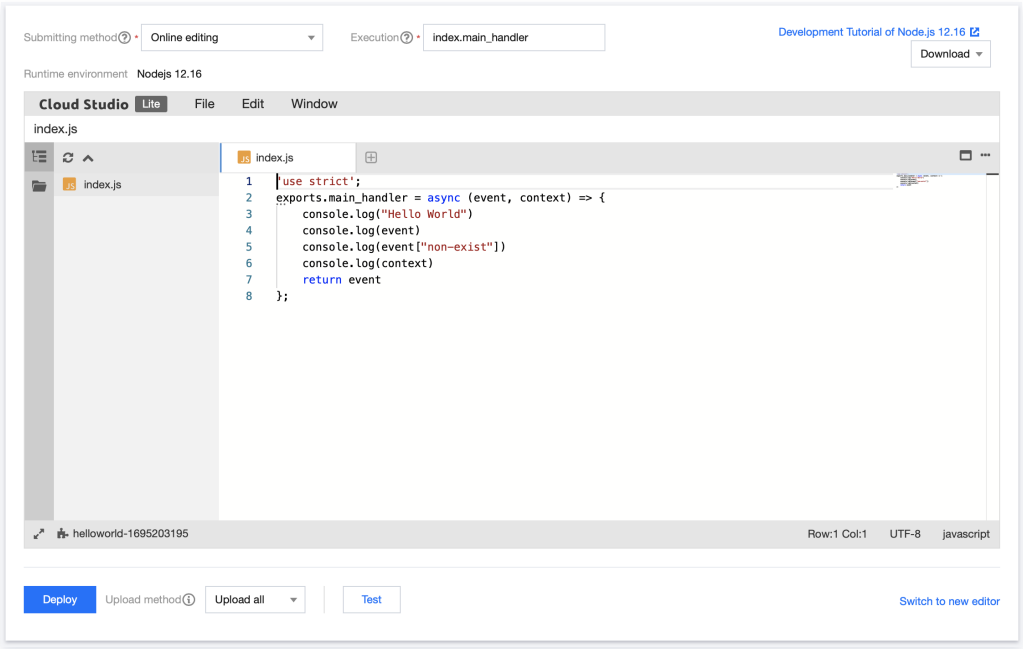
Enter the basic information of the function.

- **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**: By default, the cloud function uses UTC time. You can modify this by configuring the TZ environment variable. After you select a time zone, the corresponding TZ environment variable for that time zone will be automatically added.

5. Configure the function code.

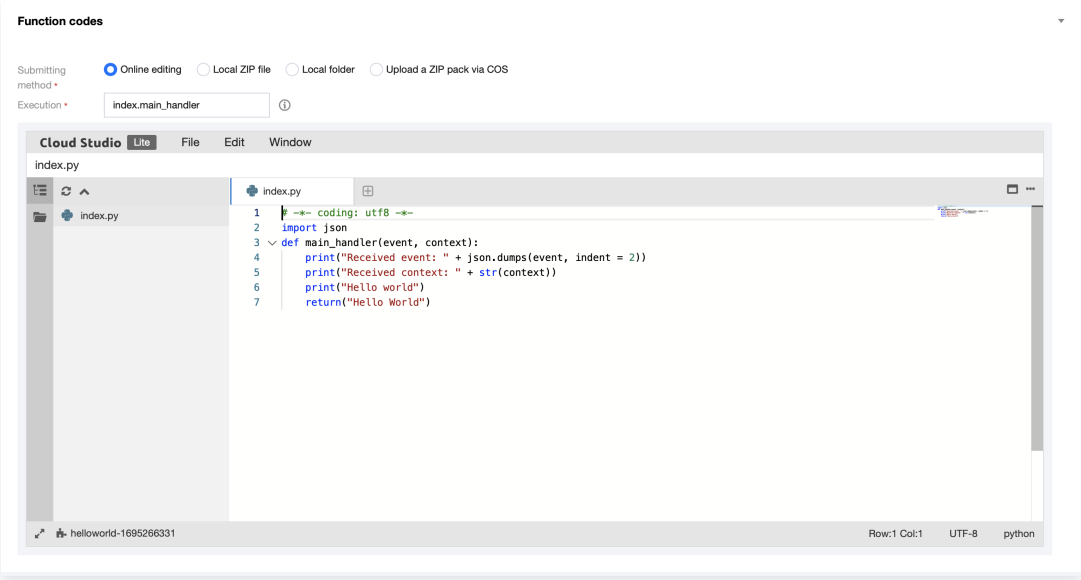
Create from template

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



Create from scratch

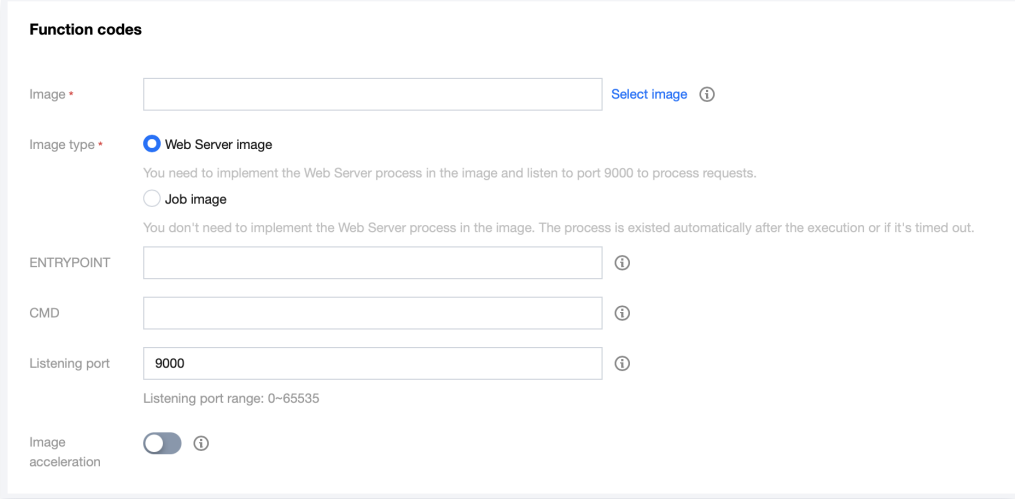
Select the method for submitting function code and the execution method, as shown below:



- **Submission Method**: Supports online editing, local upload of zip packages, local upload of folders, and uploading zip packages via COS.
  - 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.

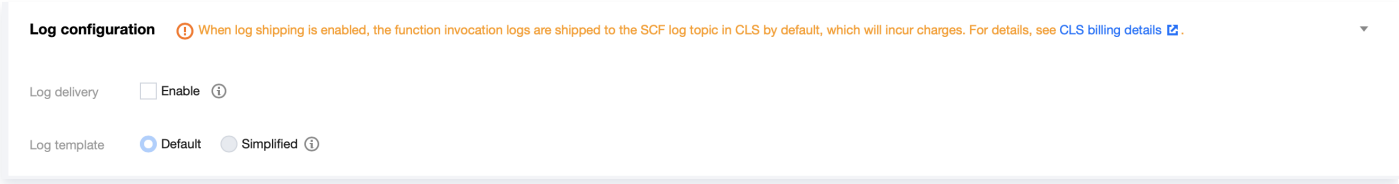
Use TCR image

Enter the relevant image information, as shown below:



- **Image**: Select an image already built in the image repository in the current region.
- **ENTRYPOINT**: (Optional) Startup command for the container. If left blank, the Entrypoint in the Dockerfile is used. Enter a valid command, such as `python`.
- **CMD**: (Optional) Startup parameters for the container. If left blank, the CMD in the Dockerfile is used. Separate each parameter with a space, for example, `-u app.py`.
- **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 the log configuration, choose whether to enable log shipping, as shown below:



Log shipping is disabled by default. When enabled, function running logs can be shipped in real-time to the specified location. For more details, see Log Shipping Configuration.

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

7. In the advanced settings, you can configure the function environment, permissions, layers, network, etc., according to your actual needs. For more details, see Function-related Configuration.
8. In the trigger configuration, choose whether to create a trigger. If you choose "Custom Creation", see Trigger Overview for more details.
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.
- Use VS Code to create a function. For more information, see Creating Functions with VS Code Plugin.

Updating Function

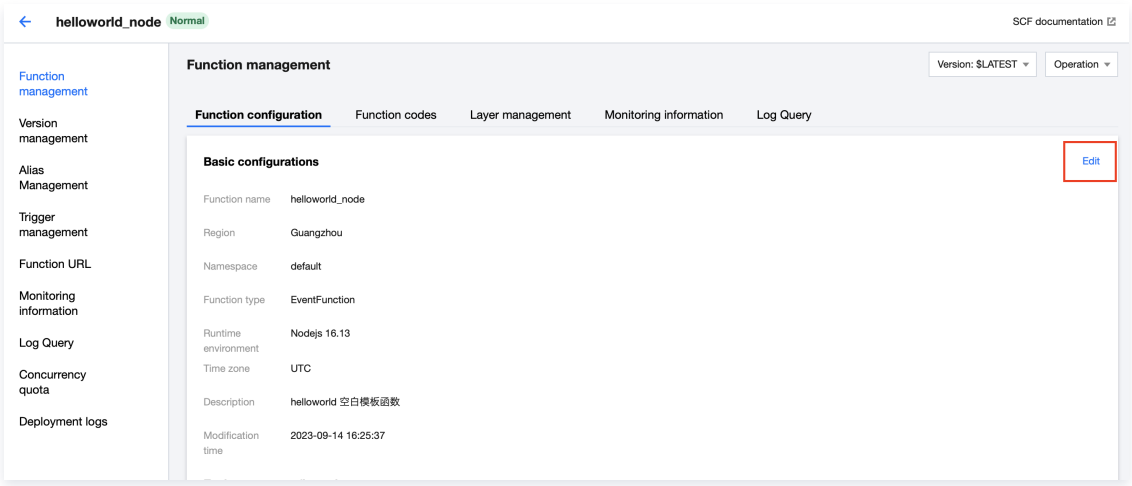
Last updated: 2023-09-27 17:15:31

This document provides guidance on how to update function configurations and code via the console and command line tools.

Function Configuration Update

Updating function configuration in the console

1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
2. At the top of the main interface, select the region and namespace where the function is located, click on the function name in the list, and proceed to the function details page.
3. Navigate to the function configuration page and click **Edit** in the upper right corner to enter edit mode, as shown below:



4. You can modify the basic configuration, environment configuration, permission configuration, log configuration, and network configuration of the function according to your needs. For more details, see Function-related Configuration.
5. After making the modifications, click **Save** to store the updated configuration. If you wish to cancel the operation, click **Cancel** to discard the changes.

Serverless Cloud Framework Updating Function Configuration

1. To modify the function configuration, directly modify the `serverless.yml` configuration file under the function root directory as shown below:

```
# serverless.yml
component: scf # Name of the imported component, which is required. The tencent-scf component is used in this example
name: scfdemo # Name of the instance created by this component, which is required


inputs:
  name: scfFunctionName
  src: ./src
runtime: Nodejs10.15 # The runtime environment of the cloud function. In addition to Nodejs10.15, other available options include: Python2.7, Python3.6, Nodejs6.10, Nodejs8.9, Nodejs12.16, PHP5, PHP7, Golang1, Java8.
  region: ap-guangzhou
  handler: index.main_handler
  events:
 - apigw:
     name: serverless_api
     parameters:
       protocols:
         - http
         - https
       serviceName:
```

```
    description: The service of Serverless Cloud Framework
    environment: release
    endpoints:
      - path: /index
        method: GET
```

2. After making the modifications, deploy the function by executing the `scf deploy` command using the Serverless Cloud Framework.

Updating function code

Updating function code in the console

1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
2. At the top of the main interface, select the region and namespace where the function is located, click on the function name in the list, and proceed to the function details page.
3. Switch to the function code page and select the submission method to edit the function code in the following ways:
   - 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.
4. After making changes, click **Save** to store the modified configuration. If you wish to cancel, click **Cancel** to discard the changes.

Updating function code using Serverless Cloud Framework

After modifying the function code locally, execute the `scf deploy` command using Serverless Cloud Framework to deploy the function and complete the code update.

> **Note**
> Serverless Cloud Framework's development mode supports synchronous updates of functions. For more details, please refer to Development Mode and Cloud Debugging.

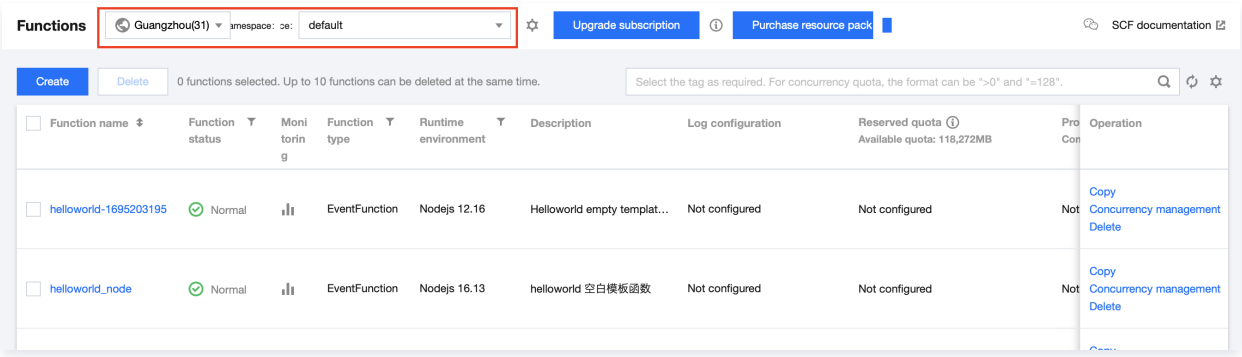
Querying Function

Last updated: 2023-09-27 20:43:30

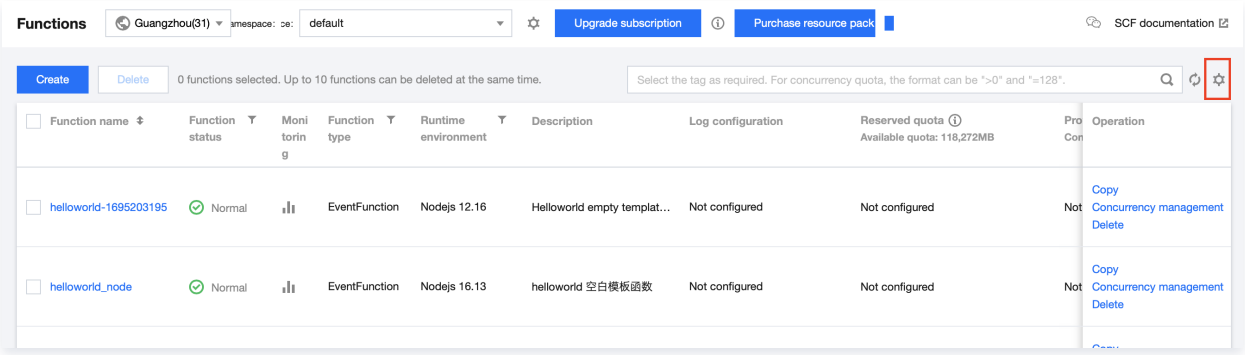
A function can be queried in the console or on Serverless Cloud Framework CLI.

Viewing a Function in the Console

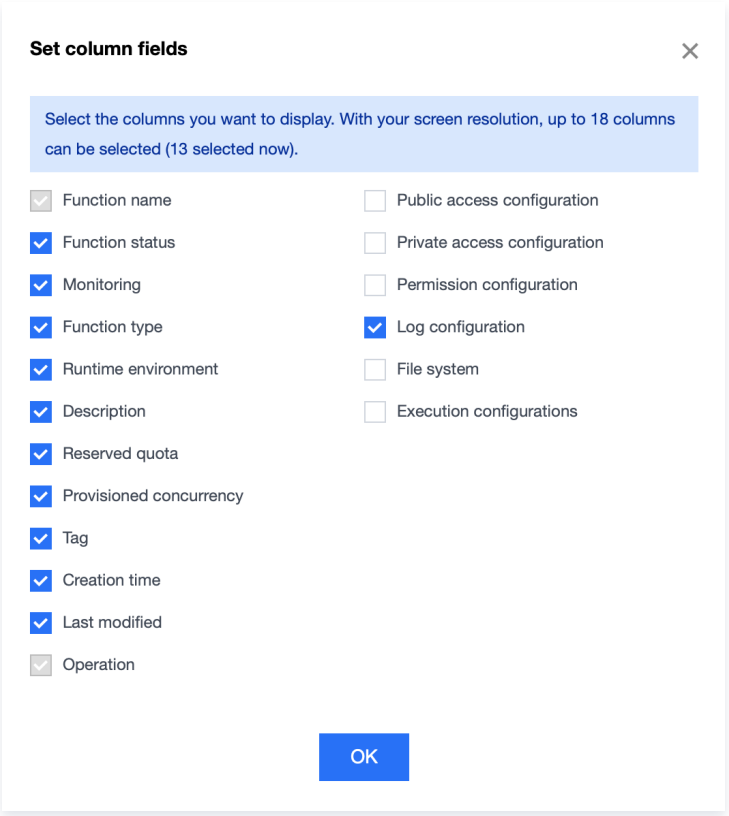
1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
2. At the top of the "Functions" page, select the desired region and namespace where the function is located. Through the function list, you can view all functions within the specified region and namespace, as shown in the figure below:



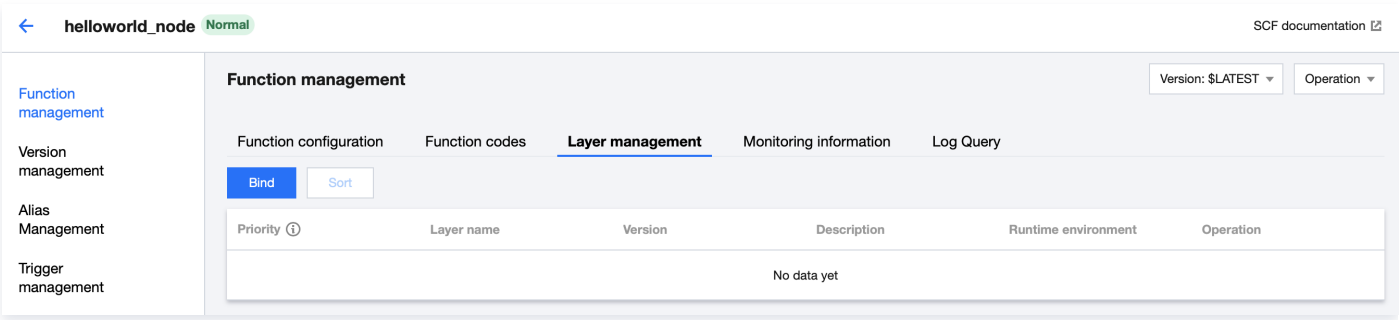
3. The function list includes the function name, monitoring, function type, runtime environment, log configuration, creation time, etc. You can customize the list fields according to your needs. Click on the right side of the function list ⚙, as shown in the figure below:



In the pop-up window, check the list details you want to display and click **OK**, as shown in the figure below:



4. Click on the function name to enter its details page, as shown in the figure below:



The function details page includes the following content:

- **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 aliases to invoke bound functions. For more information, see Alias Management.
- **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

Description

> Before using the Serverless Cloud Framework tool, please refer to Installing Serverless Cloud Framework to complete the installation.

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

Debugging Function

Last updated: 2023-09-27 20:46:01

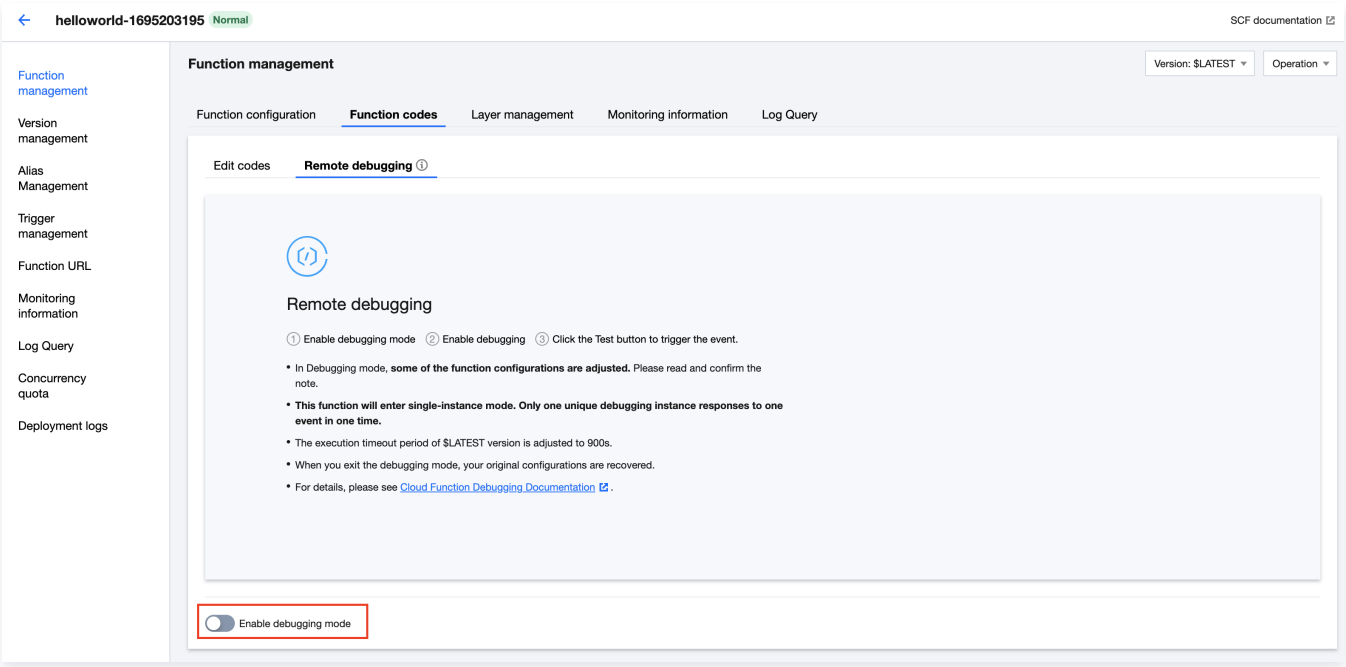
The SCF console now supports the Online Debugging feature, enabling you to troubleshoot and pinpoint issues directly through the console.

> **Note**
> Currently, the Online Debugging feature is exclusively compatible with the Chrome browser and only supports the Node.js 10.15 and Node.js 12.16 programming languages.

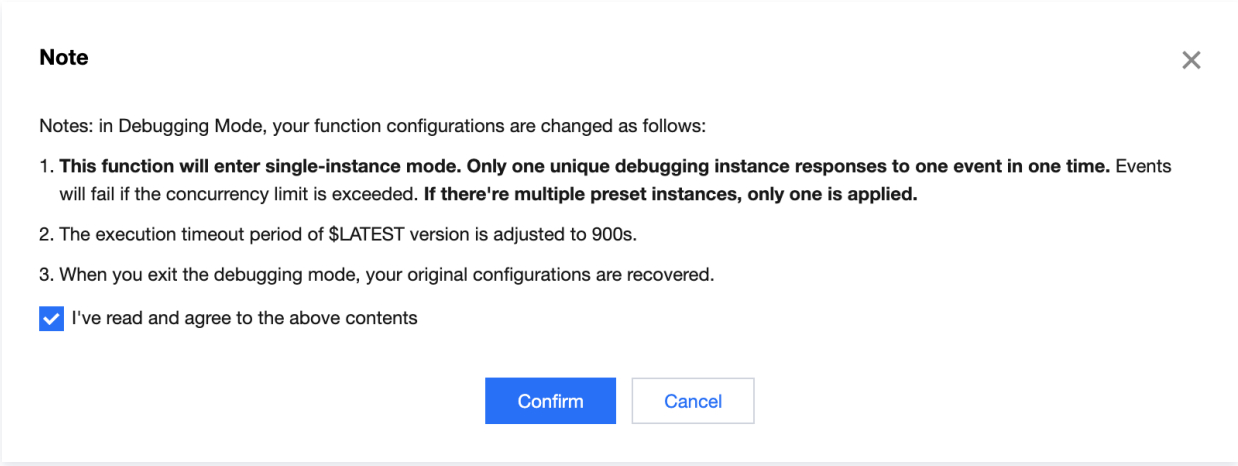
Enable debugging mode

> **Note**
> Before utilizing the Online Debugging feature, you must manually enable the debugging mode for the function. **Enabling debugging mode will alter certain function configurations**, which will be restored upon disabling the debugging mode. This may impact your operations, so please ensure you understand the following:
> - This function will transition into single-instance mode, where all versions of the function can only respond to one event at a time. Any concurrent events exceeding this limit will result in a failed call.
> - The execution timeout period is adjusted to 900 seconds. The execution timeout period cannot be set during the debugging phase.
> - Multiple pre-set instances will be reduced to a single instance.
> - Enabling debugging mode will reduce the function's performance. This function will transition into single-instance mode, where all versions of the function can only respond to one event at a time. Any concurrent events exceeding this limit will result in a failed call.

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 you wish to enable debugging mode for. Then, click on the name of the function you wish to debug to enter its details page.
3. On the **Function management** page, select **Function codes** > **Remote debugging**, and click **Enable debugging mode**. As shown in the figure below:



4. In the pop-up window, click **Confirm** to complete the activation of debugging mode. As shown in the figure below:

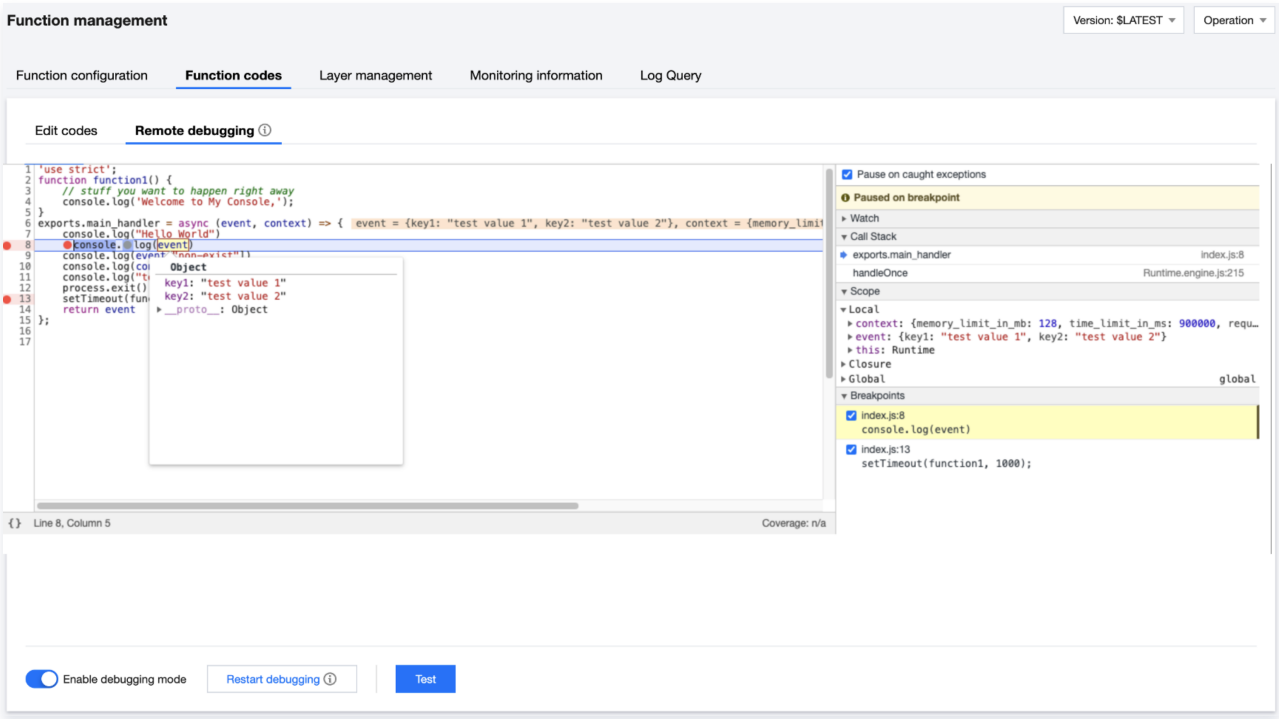


Debugging Steps

1. After Enabling Debugging Mode, the function will automatically initiate debugging upon update.

> **Note**
> If debugging mode is already enabled, you will need to manually select **Start Debugging** when you re-enter the debugging interface.

2. Once the loading is complete, the entry file will be automatically displayed. To open any file you need, you can use the shortcut Cmd + P (Mac) or Ctrl + P (Windows).
3. You can set breakpoints as needed and click **Test** to trigger a test based on the test template. As shown in the figure below:



> **Explanation**
> For more information about debugging tools, refer to Chrome DevTools.

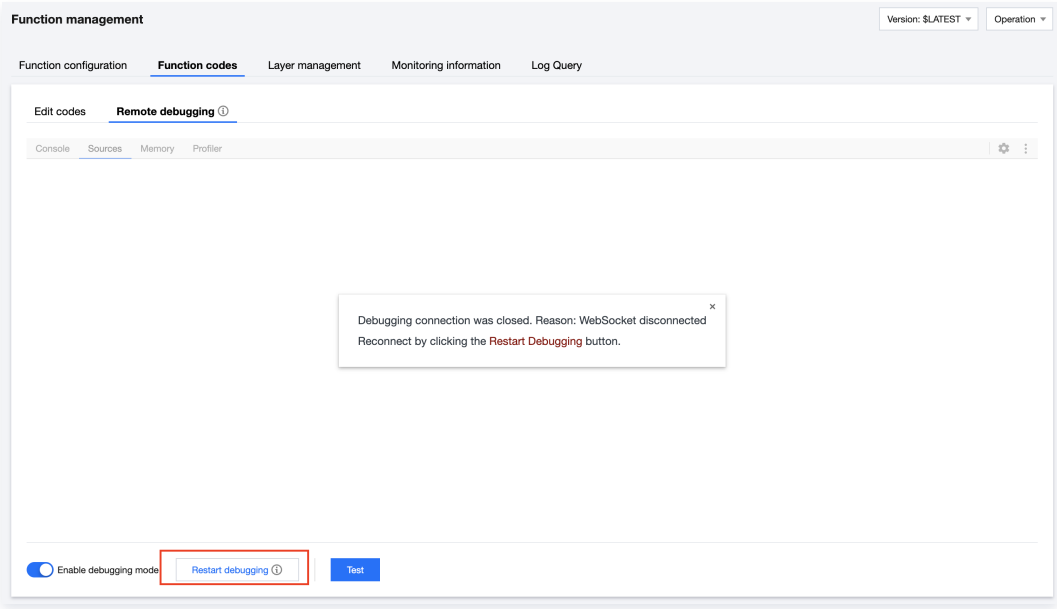
Exit Debug Mode

1. In the **Function management** page, select **Function codes** > **Remote debugging**.
2. By turning off the **Enable debugging mode** button, you can exit the debug mode and the function configurations will be restored.

> **Note**
> Modifications made to the code on the debugging page will not be synchronized to the cloud. If you need to save the modified code, please save it and use the online code editing feature.

Troubleshooting

- Due to network issues or code anomalies, the inspector might disconnect. If you encounter a situation like the one shown below, you need to click on **Restart debugging** to reconnect.



- If your function runs normally but encounters an Out Of Memory error in debug mode, you need to increase the memory configuration of the function. This will resolve the issue of insufficient memory caused by the increased memory requirements of the function when debug mode is enabled.

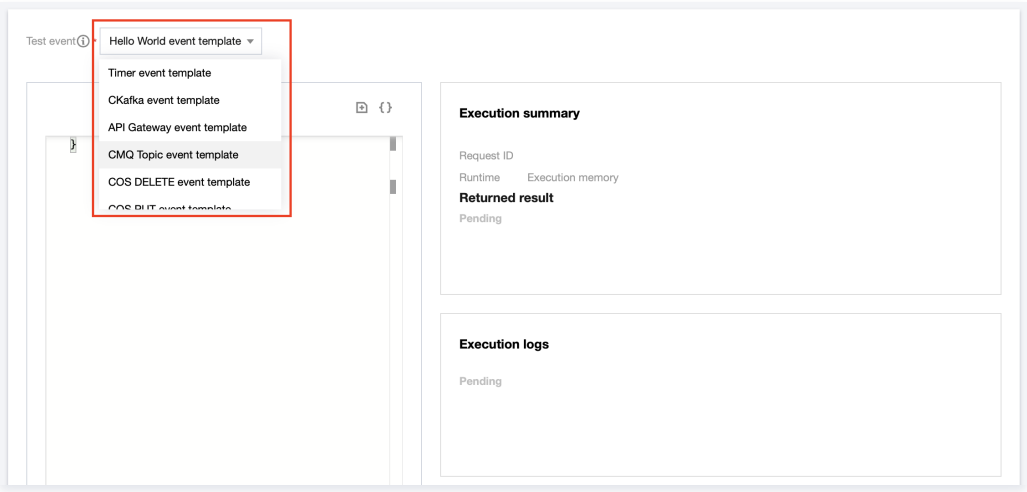
Testing Function

Last updated: 2023-09-27 17:17:24

The testing feature of Serverless Cloud Function (SCF) allows you to initiate function calls directly through the console, simulating trigger events sent by triggers, and displaying the execution status, return content, and running logs of the SCF. In the function details page of the console, you can enter the function code subpage and click **Test** to test run the function. The following video will introduce you to the function testing:
Watch video

Procedure

1. Log in to the SCF console and select **Function Service** 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 desired test template in the editor, as shown below:



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

Preset Testing Event Templates

Throughout the product iteration process, default test event templates will be continuously added. These templates are used to simulate the events and content passed to the cloud function when the corresponding trigger initiates the function's execution, which is reflected in the function as an event parameter. The test event templates must be in JSON format. The currently included default test event templates and their descriptions are as follows:

- Hello World Event Template: A simple, customizable event template. When triggering the function through the cloud API, you can input custom event content.
- COS Upload/Delete File Event Template: This simulates the events generated and passed when a cloud function is triggered upon file upload or deletion in a Bucket, following the binding of a COS object storage trigger.
- CMQ Topic Event Template: This simulates the events generated and passed when a cloud function is triggered upon receiving a message in the message queue, following the binding of a CMQ message queue topic subscription.
- API Gateway Event Template: This simulates the events generated and passed when a cloud function is triggered upon the arrival of an API request at the API Gateway, following the binding of the API Gateway to the cloud function.

> **Note:**
> There is a call timeout limit in the function console testing scenario. Calls with a timeout of less than 60 seconds are synchronous, while those exceeding 60 seconds are asynchronous.

Custom Testing Event Template

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

Use limits

The following use restrictions apply to custom testing event templates:

- 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.
- Each custom testing template can contain up to 64 KB of data.

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**.

Deploying Function

Last updated: 2023-09-27 20:47:56

Deploy via the Console

A deployment package is a zip file containing all the code and dependencies that the SCF platform runs. It must be specified when creating a function. Users can create a deployment package in their local environment and upload it to the SCF platform, or write code directly on the SCF console, which will then create and upload the deployment package for you. Please determine whether you can use the console to create a deployment package based on the following conditions:

- For simple scenarios: If your custom code only requires the use of standard libraries and SDK libraries provided by Tencent Cloud, such as COS and SCF, and there is only one code file, then you can use the inline editor in the SCF console. The console will automatically compress the code and related configuration information into a runnable deployment package.
- For advanced scenarios: If the code you write requires other resources (such as using graphic libraries for image processing, using web frameworks for web programming, using database connection libraries to execute database commands, etc.), you need to first create a function deployment package in your local environment, and then upload the deployment package using the console.

Packaging Requirements

ZIP Format

The code package submitted directly to the SCF platform, or submitted via COS and then imported into SCF, must be in ZIP format. Tools for compression or decompression can be used, such as the 7-Zip tool on the Windows platform, or the zip command-line tool on the Linux platform.

Packaging Method

When packaging, you need to package the files, not the entire code directory. After packaging, the entry-point function file should be located in the root directory of the package.

- When packaging on Windows, you can enter the function code directory, select all files, right-click, and choose "Compress to zip package" to generate the deployment package. When browsing the zip package using tools like 7-Zip, the package should directly contain the entry program and other libraries.
- When packaging on Linux, you can enter the function code directory and specify all files in the code directory as the source files when calling the zip command. This will generate the deployment package, for example, `zip /home/scf_code.zip * -r`.

Deployment Package Example

The following illustrates the process of creating a Python deployment package in a local environment.

> **Note**
> - Typically, the dependency libraries installed locally run well on the SCF platform. However, in a few cases, the installed binary files may cause compatibility issues. If this happens, please try to contact us.
> - In the example for the Python development language, the pip tool will be used locally to install libraries and dependencies. Please ensure that Python and pip are already installed on your local system.

Creating a Python Deployment Package in Linux

1. Create a directory:

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

2. Save all Python source files (.py files) for this function in this directory.
3. Use pip to install all dependencies into this directory:

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

For instance, the following command will install the Pillow library in the 'my-first-scf' directory:

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

4. In the 'my-first-scf' directory, compress all contents. Pay special attention to compress the contents within the directory, not the directory itself:

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

> **Note**
> - For libraries that require compilation, to maintain consistency with the SCF runtime environment, it is recommended to perform the packaging process under CentOS 7.
> - If there are requirements for other software, compilation environments, or development libraries during the installation or compilation process, please resolve the compilation and installation issues according to the installation prompts.

Creating a Python Deployment Package on Windows

It is recommended that you package the dependencies and code that have been successfully run in the Linux environment into a zip package as the function's execution code. For specific operations, please refer to Code Practice – Retrieve Images from COS and Create Thumbnails.
For Windows systems, you can also use the `pip install <module-name> -t <code-store-path>` command to install Python libraries. However, for packages that require compilation or come with static/dynamic libraries, since the libraries compiled under Windows cannot be invoked and run in the SCF runtime environment (CentOS 7), the library installation under Windows is only suitable for libraries implemented purely in Python.

Deploying via Serverless Cloud Framework Command Line

> **Note**
> Before using the Serverless Cloud Framework tool, please complete the installation through Installing Serverless Cloud Framework.

You can deploy functions by executing the `scf deploy` command through the Serverless Cloud Framework.

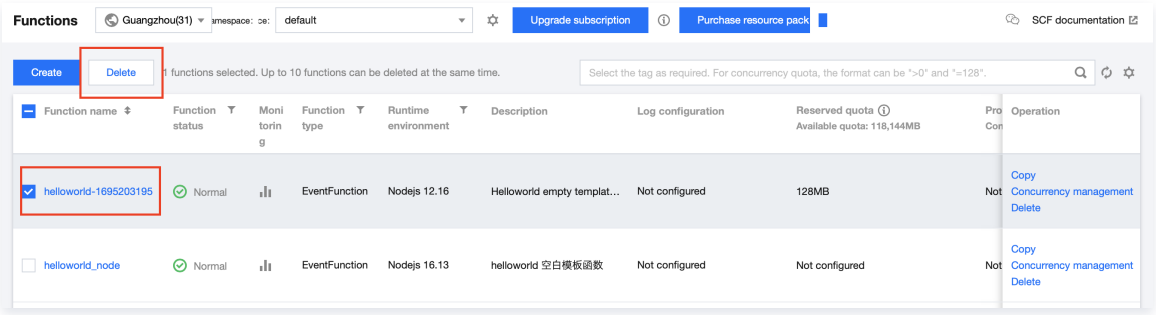
Deleting Function

Last updated: 2023-09-27 20:49:10

Function deletion can be accomplished through either the console or the Serverless Cloud Framework command line.

Deleting Functions via the Console

1. Log in to the Serverless console and select Function Service from the left sidebar.
2. On the "Functions" page, select the region and namespace to view all functions within the specified region.
3. In the function list, check the functions you want to delete and then click **Delete**, as shown in the figure below:



4. After confirming the information in the "Delete Function" pop-up window, click **OK** to delete the function.

Deleting Functions via Serverless Cloud Framework

> **Note**
> Before starting, install the Serverless Cloud Framework CLI tool first as instructed in Installation.

You can delete the deployed project by executing the `scf remove` command via Serverless Cloud Framework.

Copying Function

Last updated: 2023-09-27 17:18:42

Operational Overview

You can replicate functions across regions and namespaces through the Serverless Cloud Function Console. When replicating a function, you can choose to copy only the function code or both the function code and configuration. For functions with a high degree of code repetition, we can quickly create functions through the copy feature, modify the code, and swiftly implement multiple cloud functions with subtle differences.

Feature Overview

Source and Target of Replication

Function Type	Description	Usage Constraints
Source Function	The replicated cloud function becomes the source function.	You can select and replicate a cloud function from any region or namespace. By default, the content of the $LATEST version of the source function is replicated.
Target Function	The function to which it is copied is referred to as the target cloud function.	The target function for replication can be selected from any region, any namespace, and can be custom-named. Within the selected region and namespace, if a function with the same name exists, the replication operation will overwrite the identically named function. The target function for replication only generates or updates the $LATEST version.

Description

The `$LATEST` version is intended for development and testing, facilitating further code development and debugging.

Replication Method

The content that can be replicated in a cloud function includes the function code and its configuration:

- Function Code: This includes the function's code package, runtime environment, and execution method.
- Function Configuration: This encompasses the function's memory, timeout, description, environment variables, network, logs, and other configuration details, **excluding trigger configurations**.

When replicating a function, you can choose from the following two replication methods:

Replication Method	Description	Usage Constraints
Copy Code Only	Only the source function's code is copied to the target function's code.	If the target function exists, the original configuration is used; otherwise, the default configuration is applied. If the target function exists, its runtime environment must be identical to that of the source function.
Copy Code and Configuration	Both the source function's code and configuration are copied to the target function.	If the target function exists, its runtime environment must be identical to that of the source function.

Note

- If the source function and the target function are in different regions, the network and log configurations in the function configuration cannot be copied to the target function during the code and configuration copy process.
- Due to the absence of identical objects across regions, configuration items with regional attributes cannot be copied. If additional configurations are required, you can manually edit the cloud function to modify the necessary configurations after the copy process is completed.

Procedure

1. Log in to the SLS console.
2. In the left sidebar, select Function Service to navigate to the Function Service Management page.
3. At the top of the **Function Service**, select the region where the function you wish to update is located to view all functions in that region.
4. In the function list, select the row of the source function you wish to copy, then click **Copy** in the **Operation** column.
5. In the pop-up **Function Copy** window, fill in the following information:
   - Region: The region to which the target function belongs.
   - Namespace: The namespace to which the target function belongs.
   - Function Name: The name of the target function.
   - Content to Copy: By ticking **Function Configuration**, you can choose to **Copy Function Code Only** or **Copy Both Function Code and Configuration**.
   - Overwrite Target Function: If this option is selected, it will overwrite the function with the same name in the target region.
   - Description: Enter the optional description for the target function.
6. Click **Submit** to complete the copy.
   If the target function already exists, please confirm again in the warning window, or cancel and modify the function name again.

HTTP-Triggered Function Management Function Overview

Last updated: 2024-06-14 14:57:09

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

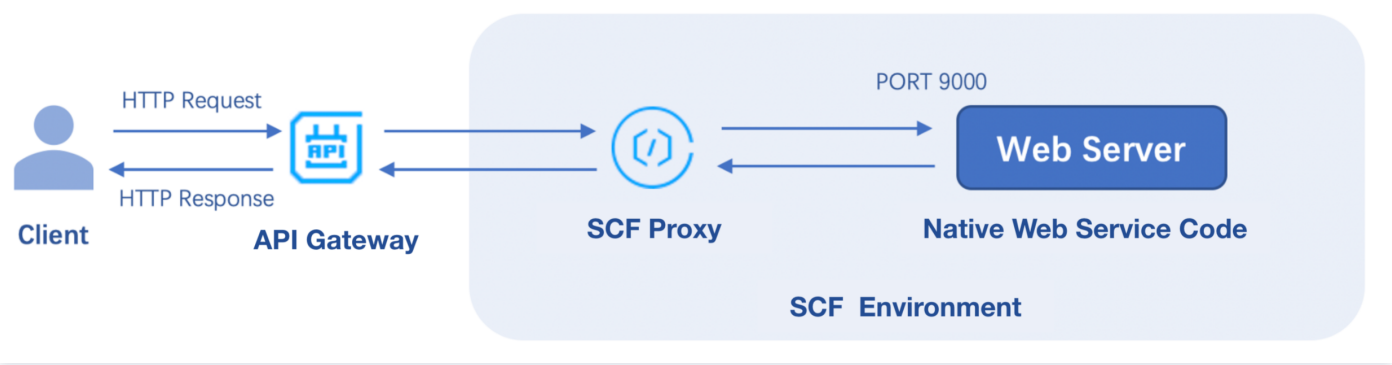
Features and Strengths

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 native 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 in the console.

Operating Principle

The operating principle of the HTTP-triggered Function is illustrated as follows:



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.

Use 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.
- The following restrictions apply to `Response headers`:
  - 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 to the specified port `9000` and address `0.0.0.0`.
- 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	`Appid` of function owner
X-Scf-Uin	`Uin` of function owner
X-Scf-Session-Token	Temporary `SESSION TOKEN`, this field will be available once the function execution role is enabled.
X-Scf-Secret-Id	Temporary `SECRET ID`, this field will be available once the function execution role is enabled.
X-Scf-Secret-Key	Temporary `SECRET KEY`, this field will be available once the function execution role is enabled.

Creating And Testing Function

Last updated: 2023-09-27 17:55:28

Operational Overview

This document describes how to quickly create and use a HTTP-triggered Function in the SCF console.

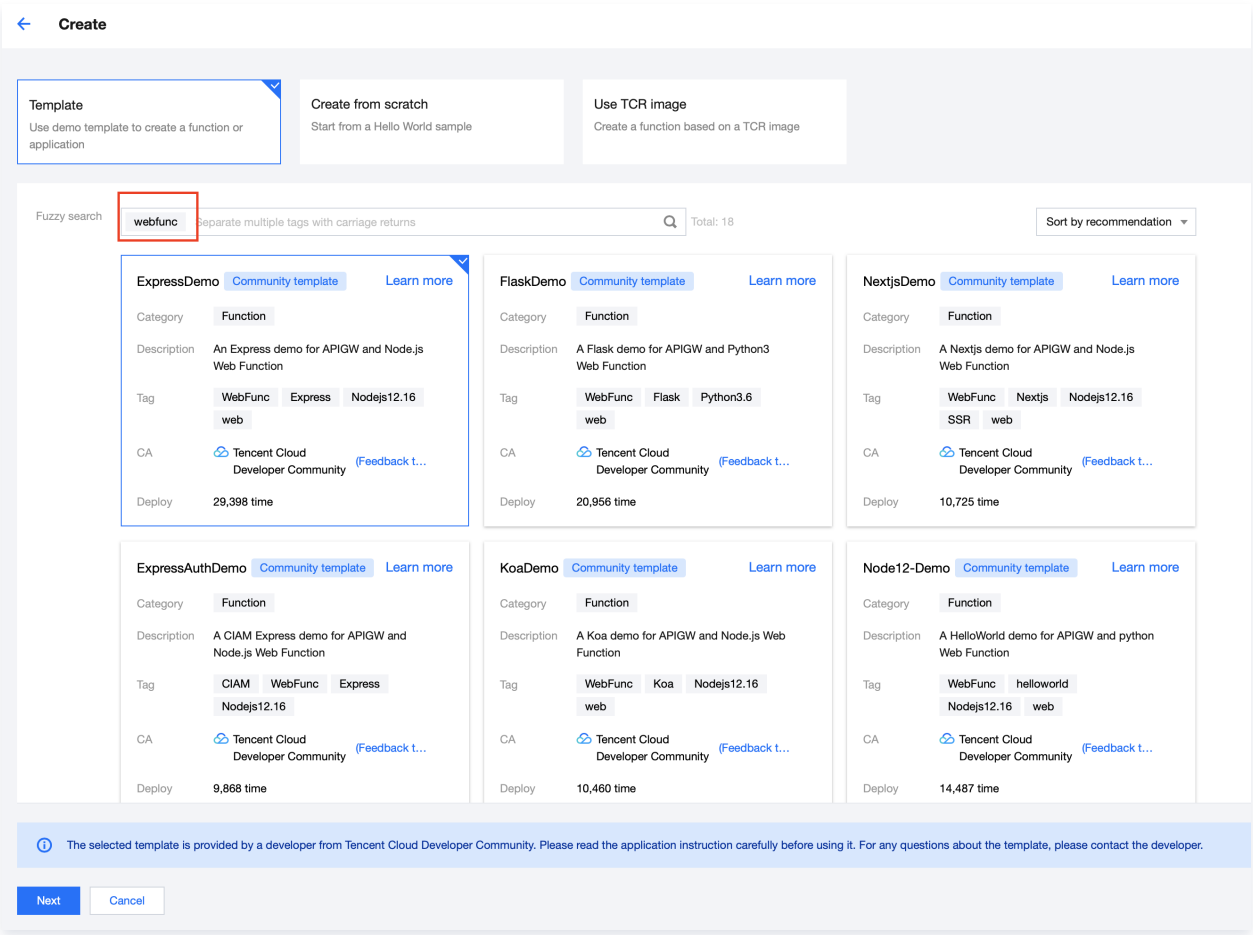
Prerequisites

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

Procedure

Creating a Function via Template

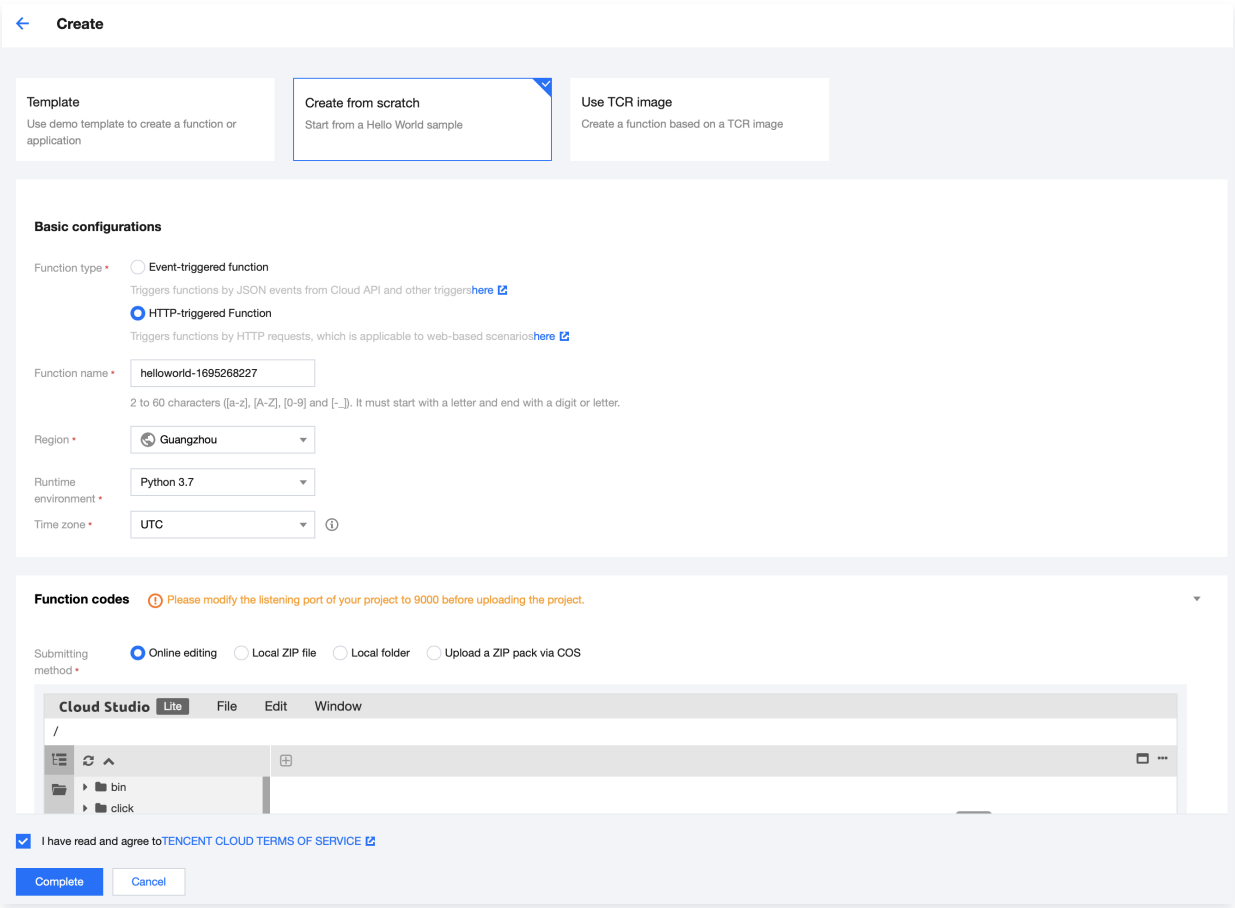
1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Choose to create a new function using a **template**. Filter for `WebFunc` in the search box to view all HTTP-triggered Function templates. Select the template you wish to use and click **Next**. As shown in the figure below:



4. On the **Configuration** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete** to create the function.
   Once the function is created, you can view the basic information of the HTTP-triggered Function on the **Function Management** page, and access it via the access path URL generated by the API gateway.

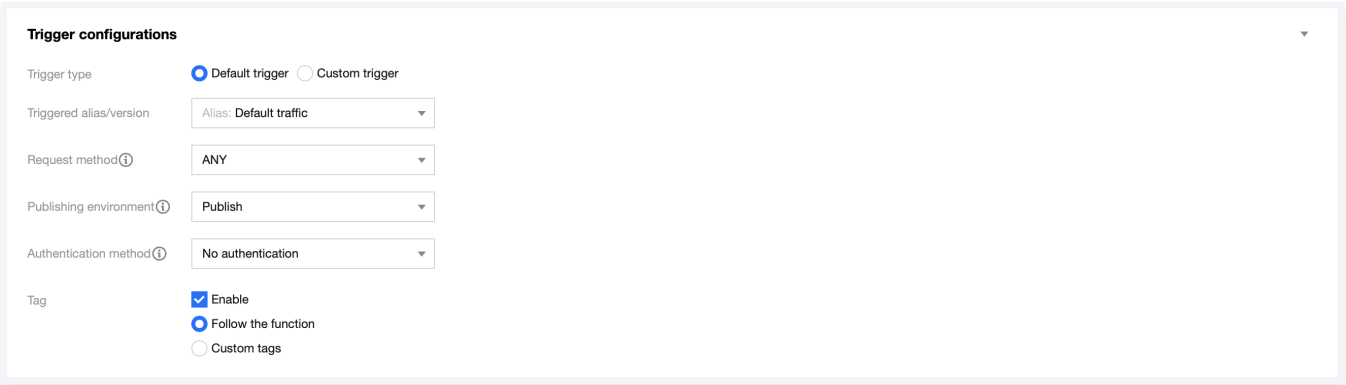
Creating custom function

1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
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. Choose to create a new function **from scratch** and fill in the basic function configuration, as shown in the figure below:



- **Function Type**: Select "HTTP-triggered Function".
- **Function name**: Enter the name of your function.
- **Region**: enter your function deployment region.
- **Runtime Environment**: Here, using Nodejs framework as an example, select "Nodejs 12.16".

4. In **Advanced Settings**, view other required configuration items.
   - **Namespace**: the default namespace is used by default. You can select another namespace for deployment.
   - **Start Command**: For HTTP-triggered Functions, you must configure the scf_bootstrap startup file for your project to ensure that the Web Server can start normally in the function environment. You can choose the default framework template provided by SCF, or use a custom template to write your own start command. For more details, refer to Startup File Instructions.
5. In **Trigger Configuration**, triggers currently only support API Gateway triggering and will automatically create triggers according to the default configuration.

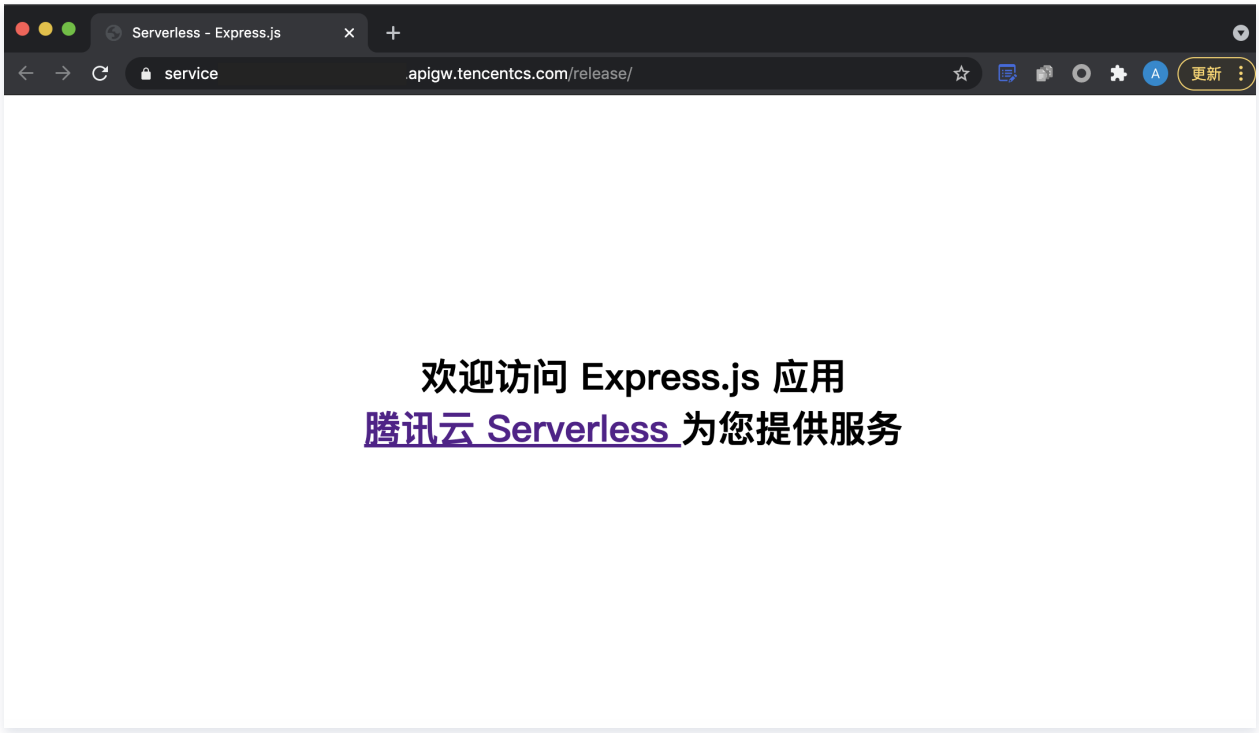


6. Click **Complete** to create the function.
   Once the function is created, you can view the basic information of the HTTP-triggered Function on the **Function Management** page, and access it via the access path URL generated by the API gateway.

Cloud test

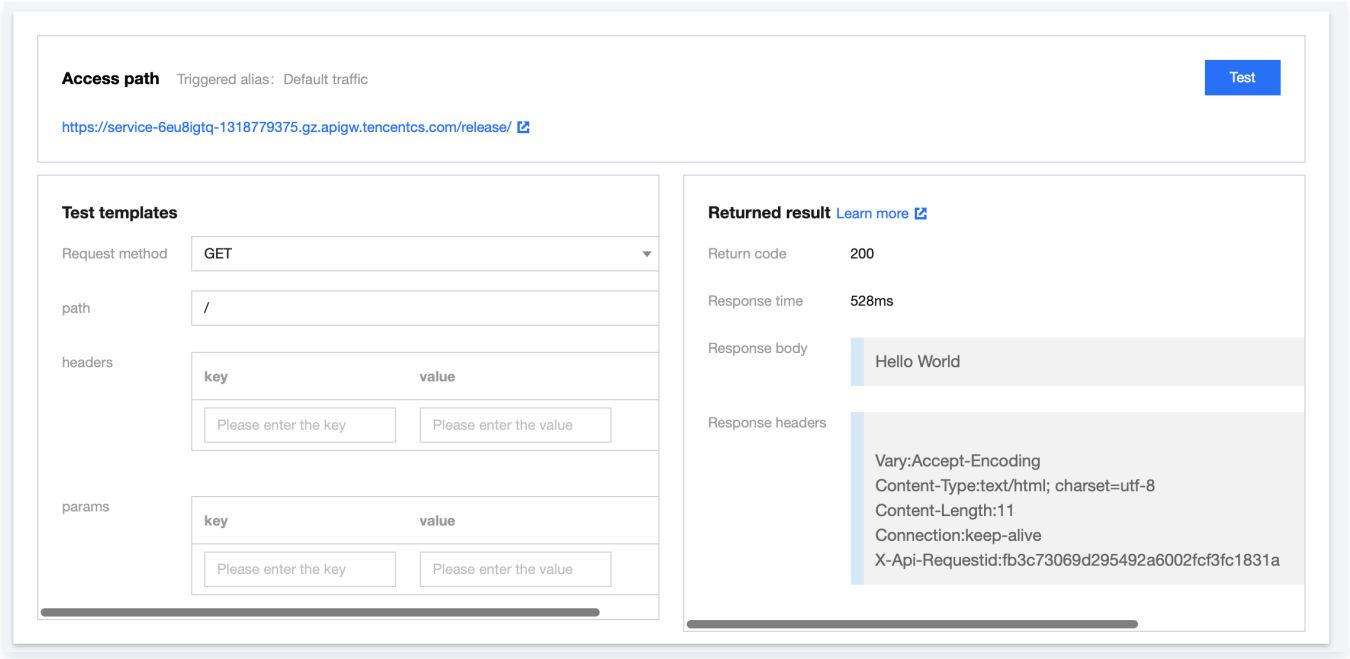
Method 1

You can open the access path URL in your browser. If it can be accessed normally, it indicates that the function has been created successfully. As shown in the following figure:



Method 2

On the function code page, you can assemble specific HTTP requests for testing through the testing capability. You can determine whether the function has been successfully deployed by examining the HTTP response results.



> **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.

Method 3

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

Viewing logs

For HTTP-triggered 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 Errors and Solutions

Common errors can be categorized into two types: User Errors and System 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 has executed successfully. If you see this return code, but the return message does not match your expectations, please verify the correctness of your code logic.

4xx Status Code

Status	Return Message	Description

Code		
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 has exited normally. Please verify the correctness of your startup file. For more details, refer to the Troubleshooting Guide.
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 time has exceeded the timeout configuration. Check whether there are time-consuming operations in the business code, or adjust the execution timeout period on the function configuration 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 `9000`.
499	kRequestCanceled	The user manually interrupts the request.

5xx Status Code

Status Code	Return Message	Description
500	InternalError	Internal error. Please try again later. If the problem persists, contact us for assistance.

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:

> **Note**
> This command is for reference only. Please modify it with your own image environment.

```
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: 2023-09-27 17:55:55

Within the standard language image environment built into the function, you need to create an executable file named scf_bootstrap to launch the Web Server. This file, along with your code files, should be packaged and deployed together to complete the creation of the HTTP-triggered Function. During actual request processing, your Web Server listens to the specified `9000` port to receive HTTP requests. These requests are then forwarded to the backend service for logical processing and response to the user.

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, please select the `/tmp` path; otherwise, the service will exit exceptionally 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.
- Be able to run in the SCF system environment (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.

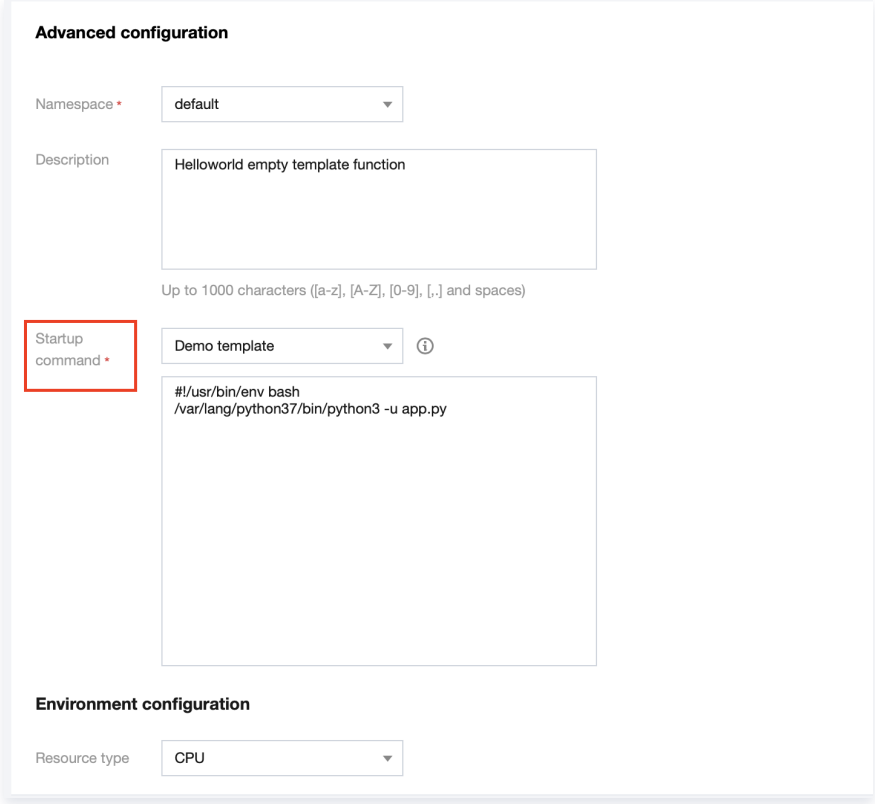
Creation method

Local package upload

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.

Quick creation in console

You can create an HTTP-triggered function in the Serverless Console. During the Create Function process, you can edit your bootstrap file under **Advanced Configuration > Startup Command**. SCF provides common startup templates for popular web frameworks, which you can modify according to your needs. As shown in the figure below:

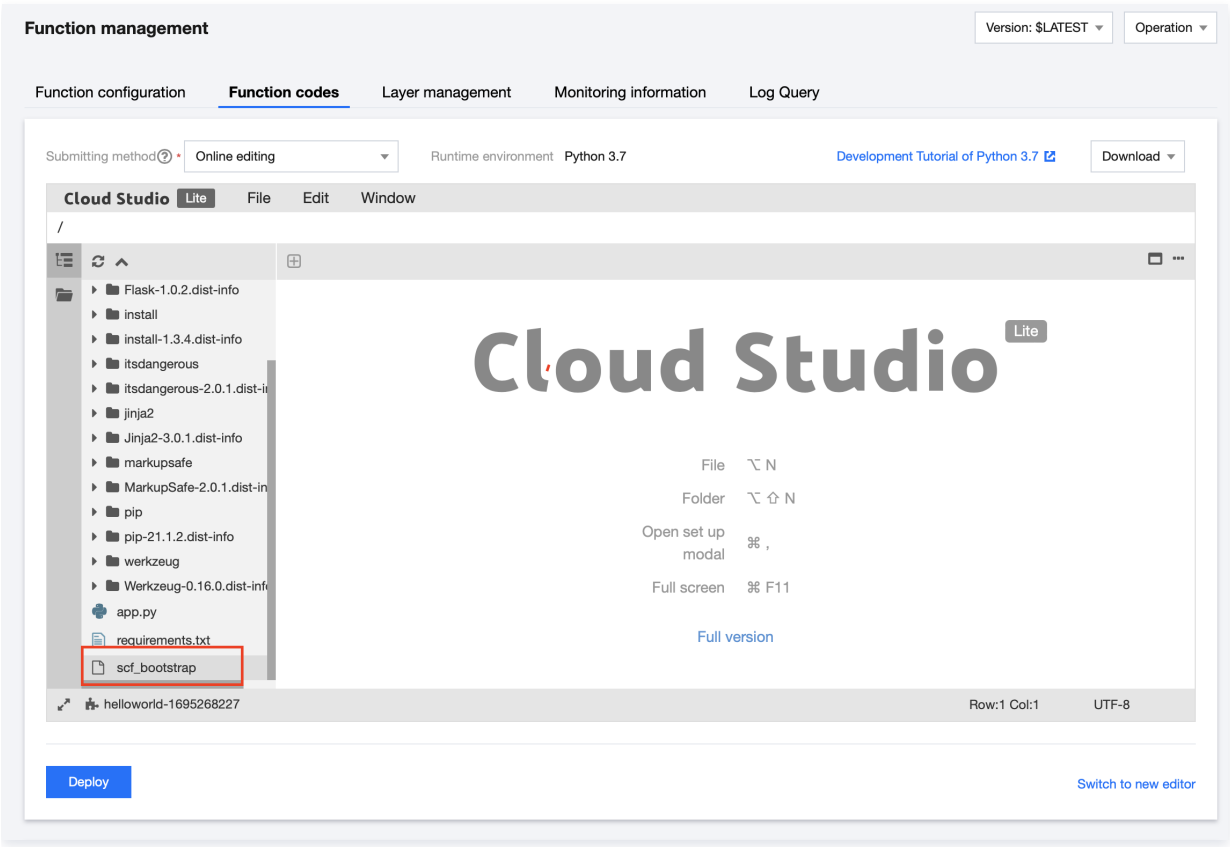


Once the function is created, the console will automatically package and deploy your code along with the `scf_bootstrap` file.

Note

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.

Upon successful deployment, you can view and edit the `scf_bootstrap` file in the code editor, as shown in the figure below:



Troubleshooting common errors

The `scf_bootstrap` executable file serves as the container's startup command. It is crucial to ensure that the container can start and execute the code logic properly. Therefore, please make sure your startup command is written correctly. If you encounter an error code `405`, it is usually due to the executable file not running correctly. Please ensure that your bootstrap file is written correctly.

Absolute Paths in Standard Languages Environments

Language Version	Absolute Path
Node.js 16.13	`/var/lang/node16/bin/node`
Node.js 14.18	`/var/lang/node14/bin/node`
Node.js 12.16	`/var/lang/node12/bin/node`
Node.js 10.15	`/var/lang/node10/bin/node`
Python 3.7	`/var/lang/python37/bin/python3`
Python 3.6	`/var/lang/python3/bin/python3`
Python 2.7	`/var/lang/python2/bin/python`
PHP 8.0	`/var/lang/php80/bin/php`
PHP 7.4	`/var/lang/php74/bin/php`
PHP 7.2	`/var/lang/php7/bin/php`
PHP 5.6	`/var/lang/php5/bin/php`
JAVA 11	`/var/lang/java11/bin/java`

JAVA 8	/var/lang/java8/bin/java

Common Web Server Bootstrap Command Templates

Nodejs

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

Python

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

PHP

```
#!/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: 2023-09-27 17:56:10

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:

Features	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	No charges are levied for the number of gateway calls.	Billed according to the standard API Gateway billing plan
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.

For detailed billing methods, see HTTP-triggered Function Billing Instructions.

Trigger Overview

Characteristics of HTTP API Gateway triggers:

- **Transparent HTTP Request**
  Upon receiving an HTTP request, if the API on the gateway is configured to dock with a cloud function, the function will be triggered to run. At this point, the API Gateway will directly forward the HTTP request without converting it into an event type format. The relevant information of the HTTP request includes the specific service and API rules that received the request, the actual path of the request, the method, and the content of the request's path, header, and query.
- **Synchronous Invocation**
  The API Gateway invokes the function synchronously, waiting for the function to return before the timeout period configured in the API Gateway expires. For more details on invocation types, please refer to Invocation Types.

Trigger Configuration

Basic gateways can only be bound through the Cloud Function console using the default creation method.
API Gateway triggers can be configured either in the Cloud Function Console or in the API Gateway Console. For more details on trigger configuration, please refer to API Gateway Trigger Configuration.

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 do not support user-defined configurations:
> - Connection Field
> - Custom fields beginning with X-SCF-

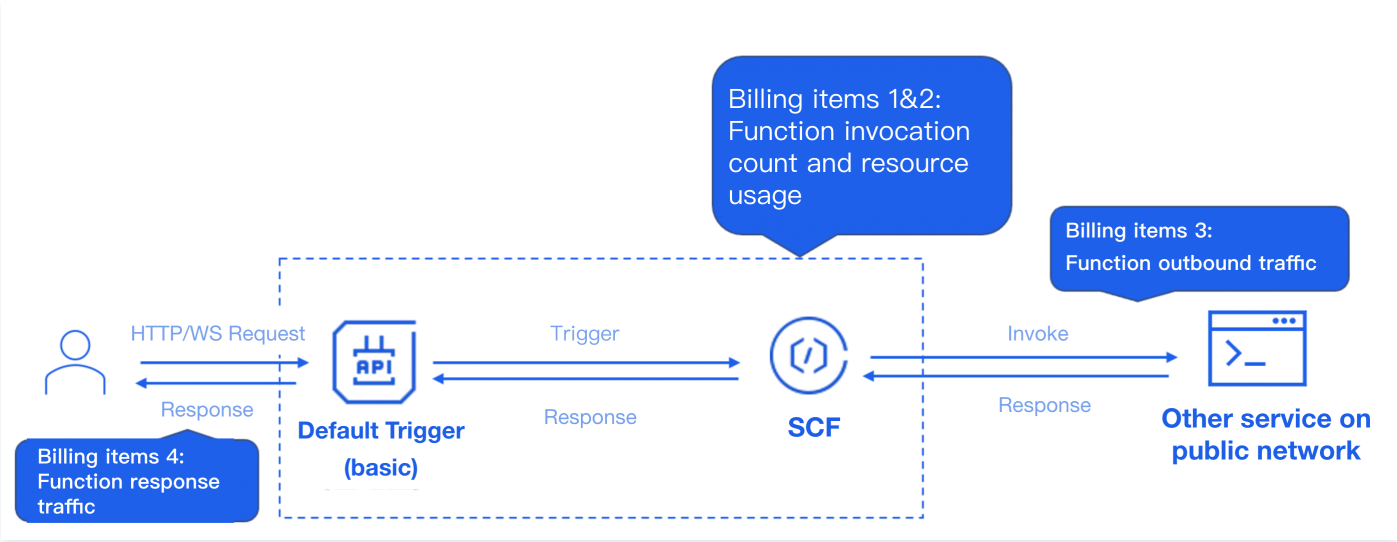
HTTP-Triggered Function Billing

Last updated: 2023-09-27 17:56:32

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

Default Creation

By choosing "Default Creation", the cloud function will automatically create a basic API Gateway service trigger for you. This type of trigger only provides you with a URL access link and is invisible in the API Gateway console. In this scenario, the billing scheme for HTTP-triggered functions is as follows:



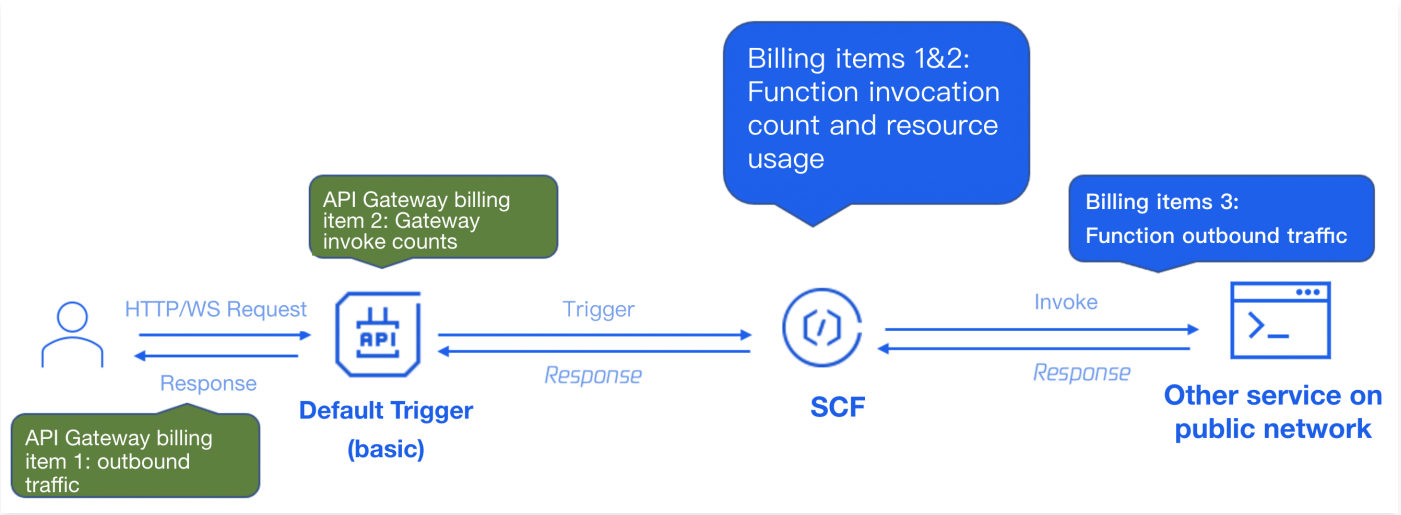
- **Trigger side**: Invocation is not billed, and outbound traffic is billed on the function side.
- **Function side**: In addition to the standard billing items, a new outbound response traffic billing item is added.

> **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

Select "Custom Creation". You need to select the trigger type in the function console and bind the created related services. In this scenario, the billing scheme is the same as the existing billing method. The function and trigger are billed according to their respective billing standards. Taking the standard API Gateway trigger as an example, the HTTP-triggered Function billing scheme

is as follows:



- **Trigger side**: Costs are calculated according to the product's own billing standards.
- **Function side**: Costs are calculated based on standard billing items (number of calls, resource usage, outbound traffic). Response traffic is not included in the function side calculation.

Trigger Capability Comparison

Features	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 Gateway console	Invisible	Visible
Advanced API Gateway capabilities (such as plugin and dedicated instance)	Not supported	Supported
Billing method	No charges for the number of gateway calls	Billed according to the standard API Gateway billing plan
Type conversion	Can be upgraded to a standard API Gateway, after which all gateway capabilities can be utilized and billed according to the standard API Gateway billing plan.	Irreversible, a standard gateway cannot be reverted back to the basic gateway created by default.
Backend timeout period	Fixed at 15s	Configurable

Deploying Web Function On Command Line

Last updated: 2023-09-27 17:57:02

Operational Overview

HTTP-triggered Function is a new function capability in SCF. Compared with event 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. For more information, please see Function Overview.
The Serverless Cloud Framework (SCF) now supports the deployment of HTTP-Triggered Functions. With the SCF component, you can quickly create and deploy HTTP-triggered Functions.

Procedure

1. Execute the following command to initialize the Serverless HTTP-triggered Function template.

```
scf init scf-nodejs
```

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 inspect the configuration details.
You merely need to introduce a new `type` parameter within the `yml` file, designate the function type, and you can accomplish the deployment of a HTTP-Triggered Function.

> **Note**
> - For HTTP-triggered 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 your local code, you can specify the entry function by setting the `entryFile` parameter in the `yml` file. The component will generate a default `scf_bootstrap` bootstrap file based on the runtime language and complete the deployment. After deployment, you need to modify the content of the `scf_bootstrap` file in the Cloud Function Console according to the actual situation of your project.

The sample yml is as follows:

```
component: scf
name: http
inputs:
src:
  src: ./
  exclude:
    - .env
# Specify the SCF type as Web type
type:        web
name: web-function
region: ap-guangzhou
runtime:      Nodejs12.16
# For Node.js, automatic dependency installation can be enabled.
installDependency: true
events:
  - apigw:
      parameters:
```

```
    protocols:
      - http
      - https
  environment: release
    endpoints:
      - path: /
    method:       ANY
```

4. Execute the `scf deploy` command in the root directory to complete the service deployment. The example is as follows:

```
$ scf deploy
serverless-cloud-framework
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-xxxxxx
   serviceName: serverless
   subDomain:   service-xxxxxx.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-xxxxxx
     internalDomain:
     url:          https://service-xxxx.cd.apigw.tencentcs.com/release/
18s › http › executed successfully
```

Relevant Commands

Viewing access log

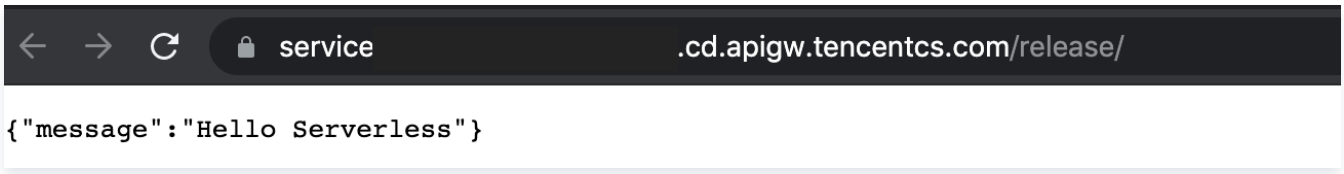
Similar to event-based functions, you can directly view the latest 10 log entries of the deployed function using the scf log command. The example is as follows:

```
$ scf log
serverless-cloud-framework
Action: "log" - Stage: "dev" - App: "http" - Name: "http"
-
  requestId: xxxxx
  retryNum:    0
  startTime:   1624262955432
  memoryUsage: 0.00
  duration:    0
```

```
  message:
    """
-
  requestId: xxxxx
  retryNum:   0
  startTime:  1624262955432
  memoryUsage: 0.00
  duration:   0
  message:
    """
```

Testing service

- Solution 1: Directly open the output path URL in the browser. If it can be accessed normally, it indicates that the function has been created successfully. As shown in the figure below:



- Solution 2: You can use other HTTP testing tools, such as CURL, POSTMAN, etc., to test the HTTP-triggered Function you have successfully created. The following example tests using the CURL tool:

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

Deleting service

Run the following command to remove your deployed cloud resources.

```
scf remove
```

Web framework migration

Serverless Cloud Framework also offers a dedicated HTTP component for Web framework deployment, enabling quick implementation of features such as Web framework deployment, layer creation, static resource separation, and CDN acceleration. For usage, please refer to Deploying Frameworks via Command Line.

WebSocket Protocol Support

Last updated: 2023-09-27 20:57:45

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.
Simultaneously, the API Gateway trigger must be set to support the "WS or WSS" frontend protocol, with the backend being the currently specified HTTP-triggered function that supports WebSocket. The ws path provided by the API Gateway can be used for client connections.

Establishing WebSocket connection

The WebSocket client initiates a WebSocket connection using the WS connection provided by the API Gateway trigger. The API Gateway and the cloud function platform will transparently pass the connection to the service process in the runtime environment. The negotiation and communication process of establishing the connection are all handled by the server-side 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 a connection termination or closes the connection. The termination status codes are 1000 and 1010 (sent by the client), and 1011 (sent by the server).	The function executes normally and the operation status is successful.	200
The client or server initiates a connection termination or closes the connection, with termination status codes other than 1000, 1010, and 1011.	The function terminates abnormally, resulting in a failed operation status.	439 (Server-side closure) 456 (Client-side closure)
In the absence of any message being sent upstream or downstream on the WS connection, reaching the configured idle timeout, the connection is severed by the function platform.	The function terminates abnormally, resulting in a failed operation status.	455

Upon continuous usage after the connection is established, once the function runtime reaches its maximum duration, the connection is severed by the function platform.	The function terminates unexpectedly, resulting in a failure status.	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.

Use limits

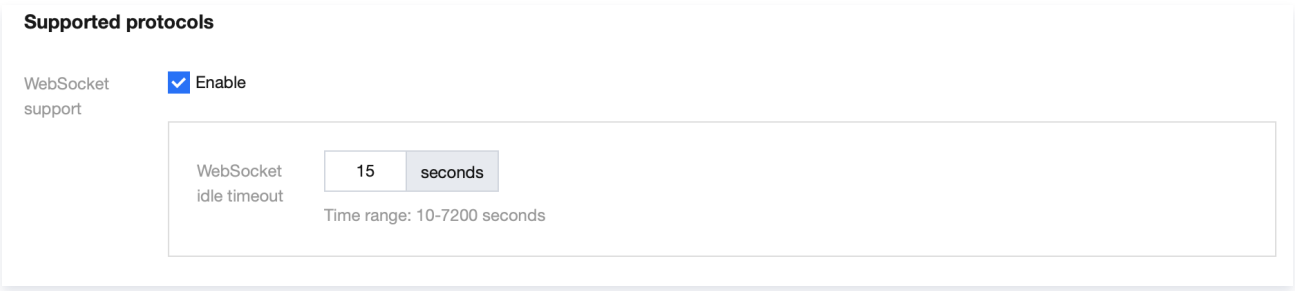
Use of WebSocket has the following limits:

- 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 size of a single request or response packet: 256KB. To increase the quota limit, please contact us.
- Single connection request size limit: 128KB/s. To increase the quota limit, please contact us.
- Single connection request QPS limit: 10. To increase the quota limit, please contact us.

Procedure

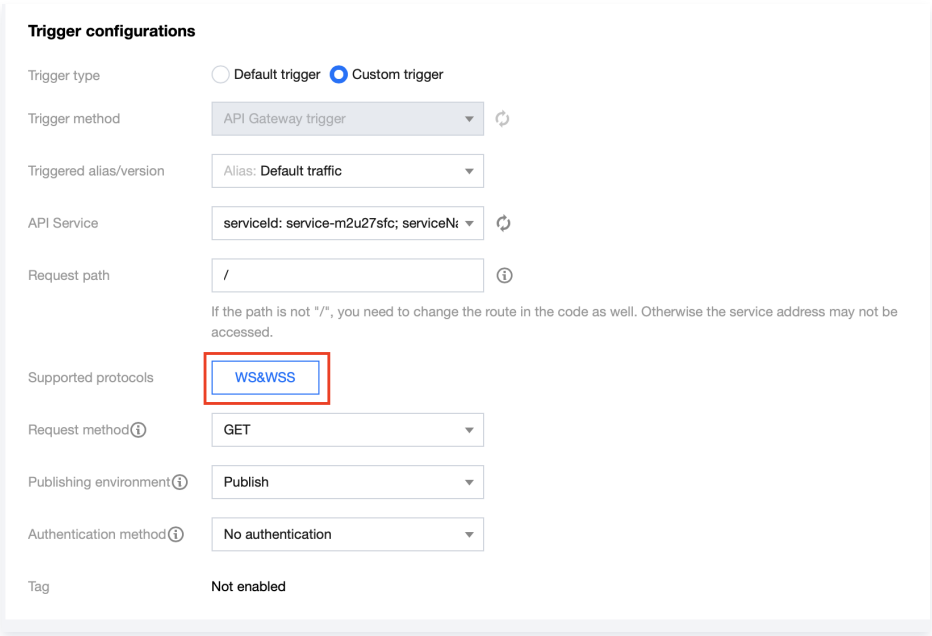
Creating a Function

1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
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. Choose to create a new function **from scratch**, and select **HTTP-triggered Function** as the function type.
4. In **Advanced Settings**, view the supported protocols. By checking **WebSocket support** and configuring the WebSocket Idle Timeout, you can complete the WebSocket protocol support. As shown below:



5. After enabling WebSocket support, in **Trigger configurations**, the protocol support of the API Gateway will also automatically switch to WS&WSS support. The link address provided by the created API Gateway will also be a WebSocket address. As shown

below:



Description
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:

- Python Example: Implementing a WebSocket server using the websockets library.
- Nodejs Example: Implementing a WebSocket server using the ws library.

SSE Protocol Support

Last updated: 2024-03-25 15:42:31

SSE (Server-sent Events) serves as a lightweight alternative to WebSocket, providing a unidirectional, server-to-client (browser) streaming message push protocol. It is commonly used in scenarios such as AI-generated dialogues. HTTP-triggered Functions currently support establishing connections between the client and the server running the function via the SSE protocol.

Protocol Activation Method

The SSE protocol is supported by default, eliminating the need for any configuration in the console.

SSE Connection Lifecycle

Under the SSE support of HTTP-triggered Functions, the lifecycle of an SSE connection is equivalent to a single function call request. Each function instance corresponds to one connection, meaning that at any given moment, a single instance only handles one SSE connection. When more connection requests are initiated, a corresponding number of instances will be launched to handle them.

Procedure

Creating a Function

1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
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. On the **Create Function** page, choose to create a new function **from scratch**, and select **HTTP-triggered Function** as the function type.
4. Using Python 3.7 as the runtime environment for this example, select **Online Editing** in **Function Code**, and copy and paste the following app.py sample code into the function code:

```
import json
import time

from flask import Flask, Response, stream_with_context

app = Flask(__name__)

@app.route('/stream')
def stream_data():
    msg = ['SSE','empowering','GPT','applications','!','Happy','chatting','!']
    # You can use yield to return content piece by piece
    def generate_response_data():
        for i,word in enumerate(msg):
            json_data = json.dumps(
                {'id': i, 'content': word})
            yield f"data:{json_data}\n\n"
            time.sleep(1)
    return Response(stream_with_context(generate_response_data()), mimetype="text/event-stream")

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9000)
```

5. Click **Done**. After the function is created, it can be viewed in the function list.

Testing Function

You can initiate an SSE connection using the curl tool in your local terminal. Here is an example command:

```
curl -v -H 'Accept:text/event-stream' {API Gateway address exposed by the function}/stream
```

The returned content is as follows:

```
> GET /release/stream HTTP/1.1
> Host: XXXXXXXXXXXXXX.XX.apigw.tencentcs.com
> User-Agent: curl/8.0.1
> Accept: */*
> 'Accept:text/event-stream'
>
< HTTP/1.1 200 OK
< Content-Type: text/event-stream; charset=utf-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< X-Api-RequestId: 22ad36c38536ee65bd07c44cb5311e1d
< Vary: Accept-Encoding
<
data:{"id": 0, "content": "SSE"}

data:{"id": 1, "content": "empowering"}

data:{"id": 2, "content": "GPT"}

data:{"id": 3, "content": "applications"}

data:{"id": 4, "content": "!"}

data:{"id": 5, "content": "Happy"}

data:{"id": 6, "content": "chatting"}

data:{"id": 7, "content": "!"}

* Connection #0 to host XXXXXXXXXXXXXXXXXX.XX.apigw.tencentcs.com left intact
```

HTTP-Triggered Function Request Concurrency Management

Last updated: 2023-09-27 21:00:39

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 iowait 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.
Web functions currently support the Enable Multiple Concurrent Requests configuration, which you can enable and configure according to your business needs. Multiple concurrent requests support two modes: **Custom Static Concurrency** and **Intelligent Dynamic Concurrency**.

- **Custom Static Concurrency**
  When enabled, if there are multiple requests at the same time, requests up to the specified concurrency value will be scheduled to execute within the same function instance. As concurrency increases, the CPU, memory, and other resources consumed by the function instance will also increase. It is recommended to set this value reasonably in conjunction with stress testing to avoid abnormal function execution. The currently supported concurrency range is 2 to 100 concurrent requests.
- **Intelligent Dynamic Concurrency**
  When enabled, it intelligently schedules more requests to run within the same function instance, provided the function instance load allows. This feature will be launched in the future. Stay tuned.

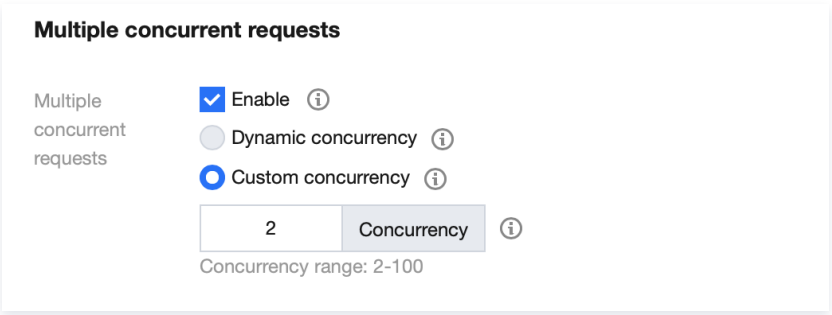
Strengths of Request Concurrency

- 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.

Instructions

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 HTTP-Triggered function that needs to be configured.
3. On the "Function Management" page, select **Function Configuration**.
4. On the "Function Configuration" page, click "Edit" to enter the editing mode.
5. In "Multiple concurrent requests", check "Enable" to activate the multiple concurrent request mode. In the input box under the pop-up "Custom concurrency", enter the required concurrency value. See the following figure:

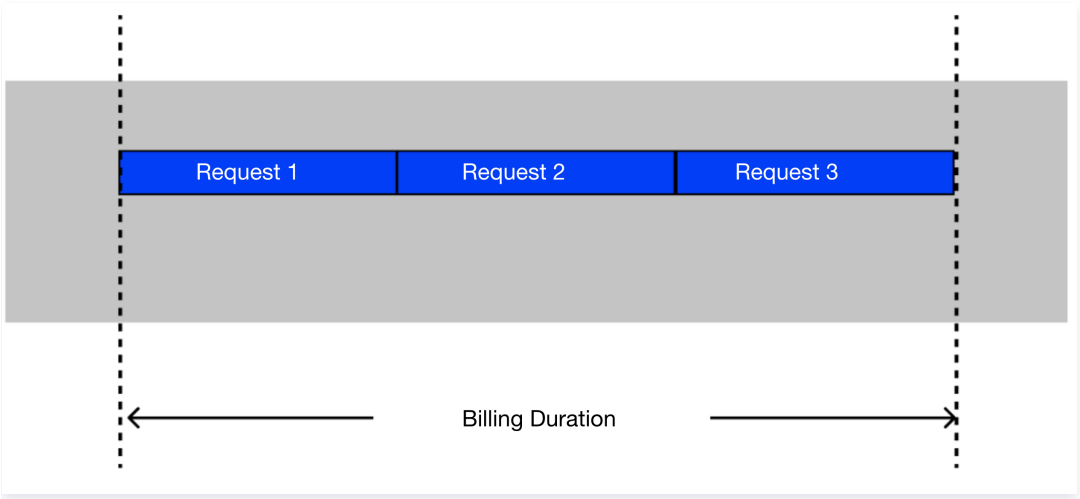


6. Click **Save**.

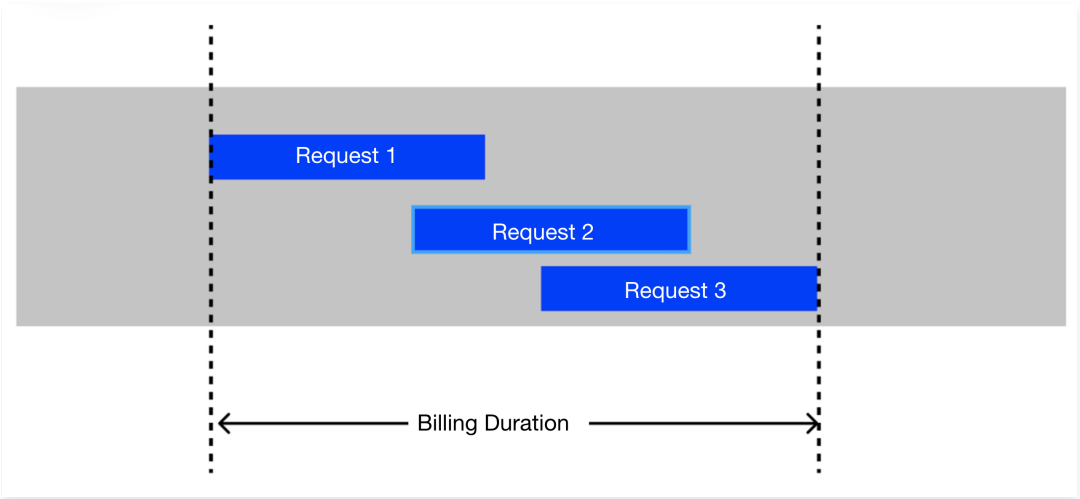
Supports and Limits

Billing

When multiple concurrent requests are not enabled, a single function instance will only process one request at a time. The next request will only be processed after the first one is completed. The billing duration of memory time is the sum of the execution duration of each request, as shown in the following figure:



Upon enabling multiple concurrent requests, a single function instance will process multiple concurrent requests at once. If a second request comes in before the first one ends, there will be a period where both requests are being processed simultaneously. During this overlapping period, the time is only calculated once. As illustrated in the following figure:



Other billable items remain unchanged. For more information, see Billing Overview.

Logs

Upon enabling multiple concurrent requests, due to the simultaneous processing of multiple concurrent requests, the logs generated by each request may not correspond one-to-one with the RequestID during stream reporting. In this case, the logger should be correctly set in the code to print the RequestID into the logs to resolve this issue. The RequestID is obtained from the `X-Scf-Request-Id` field in the common request headers received from the Web function (for some frameworks, it is `x-scf-request-id`).

NodeJS Sample Code

```
let WebSocketServer = require('ws').Server;
let wss = new WebSocketServer({ port: 9000 });

wss.on('connection', function connection(ws) {

  let requestID = ws.upgradeReq.headers['x-scf-request-id'];
  console.log('requestID: %s', requestID);

  ws.on('message', function incoming(message) {
    console.log('requestID: %s', requestID);
    console.log('received: %s', message);
  });

});
```

Overrun error

OOM

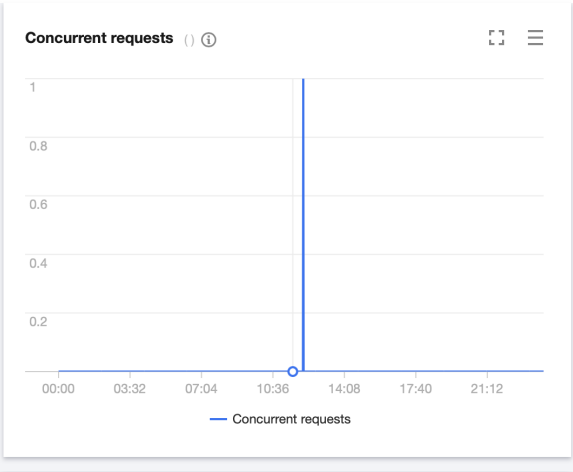
Multiple concurrent requests increase the probability of OOM. If OOM occurs, the instance will restart. In this case, the `abort` error (error code: 434 MemoryLimitReached) will be reported for multiple ongoing requests in the instance. Before setting the maximum number of concurrent requests, you need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

Timed out

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 enabling multiple concurrent requests, the "Concurrent Requests" panel will appear on the monitoring page, where you can intuitively view the concurrent request situation within a specified time period.



Monitoring FAQs

When concurrent requests are made to a function that has not been called for a while, the number of concurrent instances displayed in the "Concurrent Instances and Provisioned Concurrency" panel may exceed 1, even though the number of concurrent requests has not exceeded the set concurrency value. This is because the function instances recycle resources after a period of

inactivity, leading to a cold start when a request is made. To ensure timely response to incoming requests, multiple function instances are launched concurrently until the first instance can accept requests normally. For regular HTTP requests, after a while, new requests will be processed on a few function instances, and the remaining instances will gradually go offline after the request processing is completed, restoring the normal number of concurrent instances in the monitoring. For WebSocket connections, the number of concurrent instances will remain at the initial launch number until the connection is disconnected. Configuring dynamic provisioning to avoid cold starts can reduce the likelihood of such issues.

Log Management
Log Search Guide

Last updated: 2023-09-28 15:55:23

Operational 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 two log query methods for functions created after January 29, 2021 and the migrated functions:

- **Console Search**: The Serverless Console incorporates the CLS search and analysis page, supporting keyword search. You can use query syntax to combine keywords for search on the advanced log search page of the SCF console.
- **API Search**: You can query function call logs by invoking the Search Logs interface of the CLS log service.

> **Attention**
> - For the log structure written by SCF into CLS, please refer to Log Structure Description.
> - If your function was created before January 29, 2021 and has not yet been migrated, and you need to use more log analysis features, please refer to Log Shipping Configuration (Old) to deliver function call logs to the CLS log service.

Query in console

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

1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
2. From the "Function Service" list page, select the function name for which you need to search logs, and enter the function details page.
3. Select **Log Query** > **Advanced Search**. In the "Advanced Search" page, you can search using keywords or combine keywords using query syntax. **For detailed syntax rules, please refer to** Log Search Syntax and Rules.
4. After configuring the search content, click **Search Analysis** on the right to query the search results.

Query via API

The following describes how to get and enter relevant SCF parameters for the CLS log search API.

TopicId

`TopicId` is the CLS log topic ID where the function logs are delivered. The steps to obtain `TopicId` are as follows:

1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
2. From the "Function Service" list page, select the function name for which you need to search logs, and enter the function details page.
3. View "Log configuration" on the "Function Configuration" page, as shown below:

4. Click the link corresponding to **Log Topic** to navigate to the log service console, where you can obtain the log topic ID, as shown below:

Description
You can also obtain `TopicId` by calling the Get Function Details API to fetch `ClsTopicId`.

Query

Attention
- For the description of each field in the returned results, please refer to Log Structure Description.
- SCF invocation logs are reported in real time. If a request exists but its returned value is empty, it may be because that the function execution has not been completed yet, so the log has not been reported to CLS. Please search again after the function is executed.

Getting all invocation logs of a function

Take the function `hello-scf` in the `default` namespace as an example:

```
SCF_Namespace:"default" AND SCF_FunctionName:"hello-scf"
```

Getting the invocation logs of a request (RequestId)

Take the request `09c346d3-8417-49c5-8569-xxxxxxxxxxxx` as an example:

```
SCF_RequestId:"09c346d3-8417-49c5-8569-xxxxxxxxxxxx"
```

Getting the execution result of a request (RequestId)

The execution result includes the request start time, execution result, execution duration, memory usage, and log level. Take the request `09c346d3-8417-49c5-8569-xxxxxxxxxxxx` as an example:

```
SCF_RequestId:"09c346d3-8417-49c5-8569-xxxxxxxxxxxx" AND SCF_Type:Platform AND SCF_Message:Report*
```

Getting the returned value of a request (RequestId)

Take the request `09c346d3-8417-49c5-8569-xxxxxxxxxxxx` as an example:

```
SCF_RequestId:"09c346d3-8417-49c5-8569-xxxxxxxxxxxx" AND SCF_Type:Platform AND SCF_Message:Response*
```

Attention
- After SCF logs are connected to CLS, returned data will be retained for functions, which will be written to the `SCF_Message` field in the format of `Response RequestId:xxx RetMsg:xxx` in CLS.
- The value of `SCF_Message` is limited to 8 KB in length, and excessive parts will be truncated.

Getting the list of requests for a function over a period of time

For instance, with the function name `hello-scf`, in the namespace `default`, querying the alias `$DEFAULT`, and version 1:

- Get the list of failed requests due to internal errors:

```
SCF_Type:Platform AND SCF_Message:Report* | select SCF_RequestId as requestId, SCF_RetryNum as retryNum,SCF_StartTime as startTime where SCF_FunctionName='hello-scf' and SCF_Namespace='default' and SCF_Qualifier='1' and SCF_Alias:'$DEFAULT' and SCF_StatusCode = 500 order by startTime desc
```

- Get the list of requests that took more than 3 seconds to execute:

```
SCF_Type:Platform AND SCF_Message:Report* | select SCF_RequestId as requestId, SCF_RetryNum as retryNum,SCF_StartTime as startTime where SCF_FunctionName='hello-scf' and SCF_Namespace='default' and SCF_Qualifier='1' and SCF_Alias:'$DEFAULT' and SCF_Duration>3000 order by startTime desc
```

Log Structure Description

Last updated: 2023-09-27 21:01:55

SCF logs are written into CLS by log. Each request is logged in multiple logs, and each log is in a fixed `key:value` format.

Key-Value Format of Single Log

Default format

In the default format, each log consists of 14 fixed `key:value` pairs as shown below:

Field	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 running duration (in milliseconds).
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	Function running Status Code. 202 indicates that the request is in progress.
SCF_RetryNum	long	Number of retries.

Simplified format

In the simplified format, there are fewer key-value pairs than in the default format, where only the fields required in log query scenarios are retained. In this format, logs are divided into user logs and platform logs with different specific formats as shown below:

User log

A user log is a user's standard output in the code, which the platform will capture and report to CLS.

Field	Field type	Description
SCF_FunctionName	text	Function name.
SCF_Namespace	text	Function namespace.
SCF_LogTime	long	Log generation time.
SCF_RequestId	text	Request ID
SCF_Alias	text	Alias.
SCF_Qualifier	text	Version.
SCF_Message	text	Log content.

SCF_Type	text	Log type. Platform: platform log, Custom: user log.
SCF_RetryNum	long	Number of retries.

Platform log

A platform log is the log printed at the start or end of each request. It is used to mark the start and end of a request and record its execution. Platform logs cannot be canceled.
Platform logs consist of two types of key-value pairs. The key-value pairs for `report` logs, which record the execution of requests, are shown in Table 1. The key-value pairs for other platform logs (such as START, END, ERROR, Response) are shown in Table 2.

Table 1

Field	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 running duration (in milliseconds).
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	Function execution status code.
SCF_RetryNum	long	Number of retries.

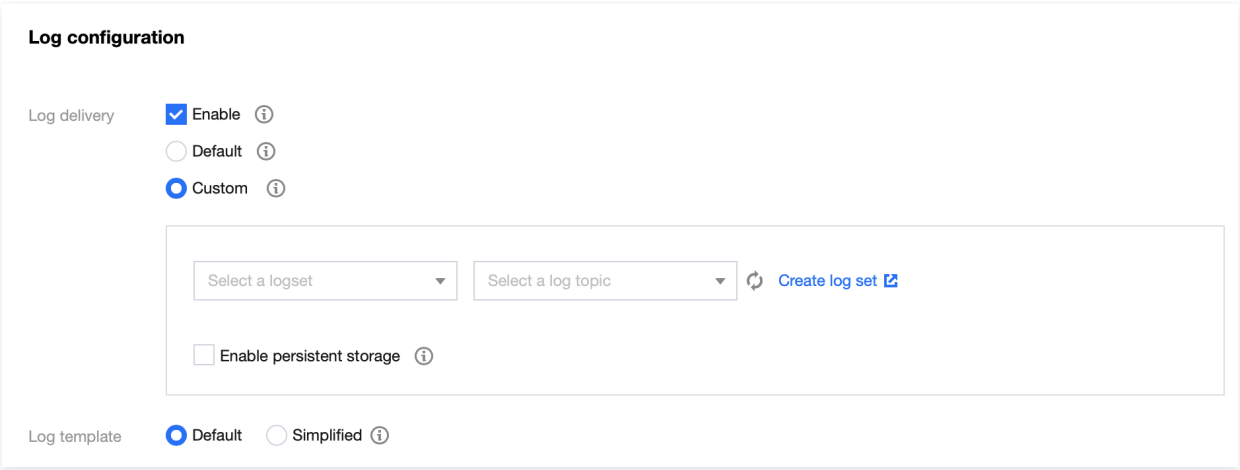
Table 2

Field	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_Alias	text	Alias.
SCF_Qualifier	text	Version.

SCF_Message	text	Log content.
SCF_Type	text	Log type. Platform: platform log, Custom: user log.
SCF_RetryNum	long	Number of retries.

Log format selection and switch

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 of the function. Click on the name of the function for which you wish to switch the log format, to enter its details page.
3. On the "Function Management" page, select **Function Configuration** > **Log Configuration** > **Log Format** to choose and switch the log format. As shown in the figure below:



Note

- The log configuration is at the function level, and the updated configuration will take effect immediately for both the `$LATEST` version and published versions.
- The Get Function Execution Logs interface does not currently support querying logs in the simplified format. Please use the CLS Log Search Interface. If you have already used the API to process log key values, please read carefully about the field adjustments under different log formats before proceeding.
- Compared with the default format, the simplified format can reduce the fees incurred by logs, but information such as execution duration and memory during function execution will no longer be displayed in real time.
- The simplified format is only supported for non-image-based event functions.

Log Structure of Single Request

There are two log structures for one single SCF request: invocation log and provisioning log.

Invocation log structure

SCF invocation logs use platform logs to mark the request start and end, request error message, function return, and request execution, and user logs are encapsulated between the start and end of the request. The log structure is as follows (the table only shows the `SCF_Message` field as an example):

SCF_Message	Log Type	Description
START RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx	Platform log	Request start.

init log	User log	The log content printed by the user during function initialization. The container will execute the initialization logic only in case of cold start, and no initialization logs will be output in non-cold start scenarios.
Init Report RequestId: 09c346d3-8417-49c5-8569-xxxxxxxxxxxx Coldstart: 236ms (PullCode: 70ms InitRuntime: 8ms InitFunction: 158ms) Memory: 640MB MemUsage: 57.86MB	Platform log	Initialization execution log, where Coldstart represents the total time consumed during the initialization phase. PullCode represents the time consumed to pull user function and layer codes or to pull image during the initialization phase. InitRuntime represents the platform time consumed during the initialization phase. InitFunction represents the time consumed by user code execution during the initialization phase. Memory represents the configured function memory, and MemUsage represents the memory used during the initialization phase. The container will execute the initialization logic only in case of cold start, and no initialization logs will be output in non-cold start scenarios.
invoke log	User log	The log content printed by the user during function invocation.
ERROR RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Result:xxx	Platform log	Function error cause. There will be no ERROR logs when the function is executed properly.
Response RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx RetMsg:"Hello World"	Platform log	The function return is recorded in `RetMsg`.
END RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx	Platform log	Request end.
Report RequestId:09c346d3-8417-49c5-8569-c55033b17f51 Duration:1ms Memory:128MB MemUsage:29.734375MB	Platform log	Function invocation execution log. `Duration` is the function execution duration, `Memory` is the configured function memory size, and `MemUsage` is the execution memory size during function execution.

Below is the sample code for printing one line of initialization log and one line of invocation log of a web function in the `Python` runtime environment:

```
from flask import Flask
app = Flask(__name__)
print("init log")

@app.route('/')
def hello_world():
  return 'Hello World'

if __name__ == '__main__':
  app.run(host='0.0.0.0',port=9000)
```

The output log structure is as follows (only the content of the `SCF_Message` field is displayed):

```
START RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx
 * Serving Flask app "app" (lazy loading)
 * Environment: production
 WARNING: Do not use the development server in a production environment.
 Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:9000/ (Press CTRL+C to quit)
init log
Init Report RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Coldstart: 640ms (PullCode: 119ms InitRuntime: 2ms
InitFunction: 519ms) Memory: 640MB MemUsage: 5.21MB
Hello world
Response RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx RetMsg:"Hello World"
END RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx
Report RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Duration:1ms Memory:128MB MemUsage:29.734375MB
```

Provisioning log structure

SCF provisioning logs start by user log printing and end by provisioning marking in platform log. The log structure is as follows (the table only shows the `SCF_Message` field as an example):

SCF_Message	LogType	Description
provision log	User log	The log content printed by the user during function initialization, which will be recorded in a log in instance provisioning scenarios.
ERROR RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Result:xxx	Platform log	Cause of an instance provisioning failure. There will be no ERROR logs if instance provisioning succeeds.
Provisioned Report RequestId: c6af0fb4-1c07-4a92-8307-xxxxxxxxxxxx Coldstart: 640ms (PullCode: 119ms InitRuntime: 2ms InitFunction: 519ms) Memory: 640MB MemUsage: 5.14MB	Platform l	Provisioning instance execution log, where Coldstart represents the total time consumed by the provisioning instance. PullCode represents the time consumed in pulling user function and layer codes or pulling image during the provisioning process. InitRuntime represents the platform time consumed during the provisioning process. InitFunction represents the user code execution time consumed during the provisioning process. Memory represents the configured function memory, and MemUsage represents the memory used during the provisioning stage. This log is only output in provisioning scenarios.

	o g	

Below is the sample code for printing one line of initialization log and one line of invocation log of a web function in the `Python` runtime environment:

```
from flask import Flask
app = Flask(__name__)
print("init log")

@app.route('/')
def hello_world():
  return 'Hello World'

if __name__ == '__main__':
  app.run(host='0.0.0.0',port=9000)
```

The output log structure in instance provisioning scenarios is as follows (only the content of the `SCF_Message` field is displayed):

```
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:9000/ (Press CTRL+C to quit)
init log
Previsioned Report RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Coldstart: 640ms (PullCode: 119ms InitRuntime:
2ms InitFunction: 519ms) Memory: 640MB MemUsage: 5.21MB
```

Log Delivery Configuration

Last updated: 2023-09-27 21:06:18

> **Description**
> 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, please 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.

Use Limits

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.

For other limits, see CLS Specification Description. Pay attention to whether the CLS configuration can meet your business needs. Exceeding the limits may cause log write failures.

Procedure

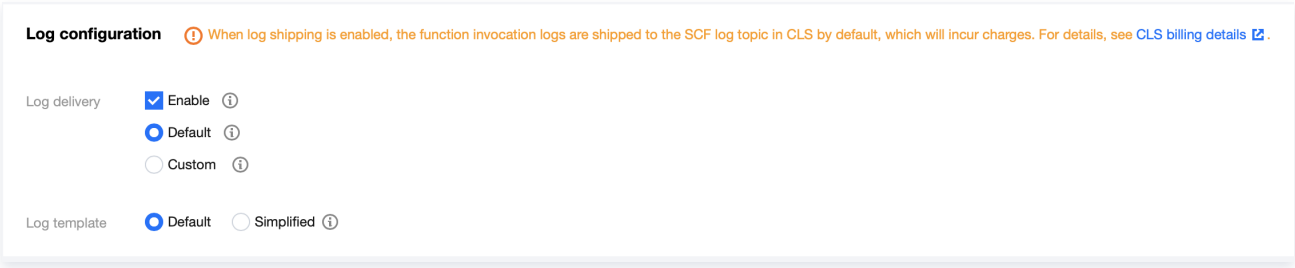
Default delivery

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.

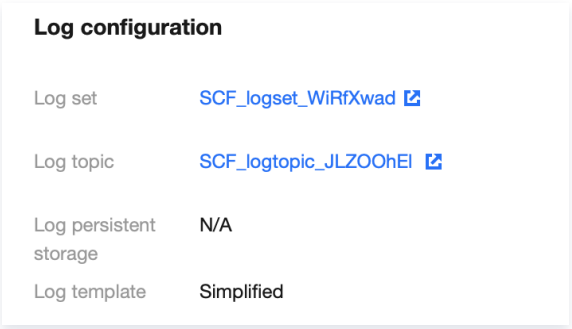
> **Description**
> - CLS is billed separately, and the SCF-specific log topic will consume the free tier of CLS. For more information, please see Billing Overview.
> - To ensure the normal display of logs in the SCF console, the SCF-specific log topic does not support modification of index configurations. If you need to customize the log index configuration, please refer to the following section on customizing the delivery configuration of function log topics.

Configuring CLS

1. Log in to the Serverless console and select Function Service from 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:



4. Click **Finish** to create the function and complete the default log delivery. You can view the log configuration in **Function Management** > **Function Configuration**, as shown below:



Viewing and managing logs

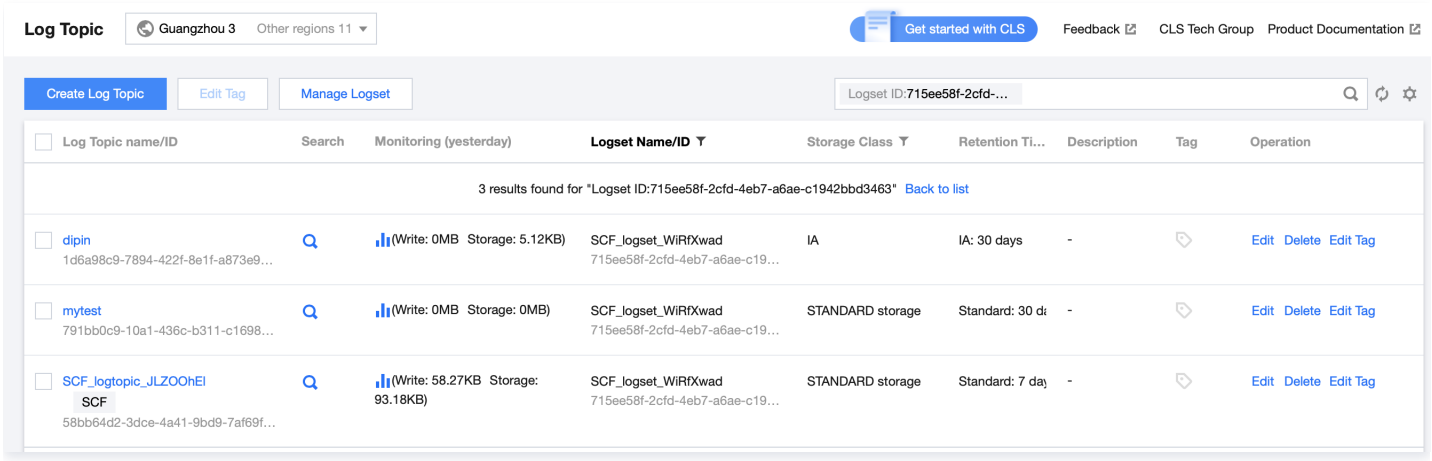
You can click on the logset ID in "Log Configuration" to go to the Log Service Console to view and manage logs. The SCF dedicated logset is marked with `SCF` in the Log Service Console. If you need to persistently store, deliver or consume logs, or monitor and alert log content, you can complete the configuration in the Log Service 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 Log Service Console and Create Log Topic. This document uses the creation of the `SCF-test` log topic in Guangzhou as an example. As shown in the figure below:

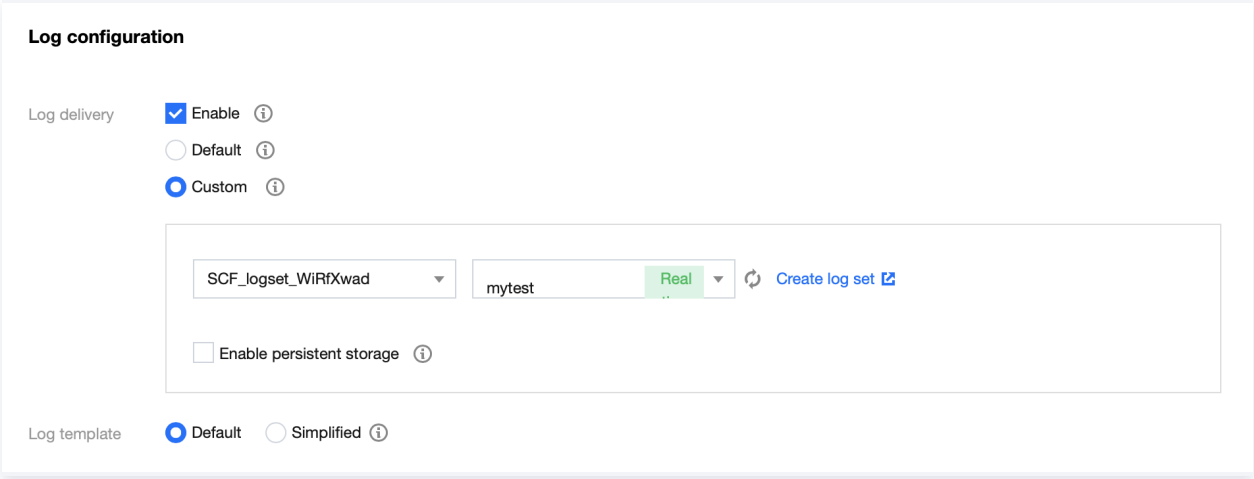


> **Note**
> For the logset region, please select the region where the SCF service is located. Cross-region log push is not supported currently.

Configuring CLS

1. Log in to the Serverless console and select Function Service from 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 choose the log topic that has been created for this function, this document uses `SCF-test` as an example. As shown in the figure below:

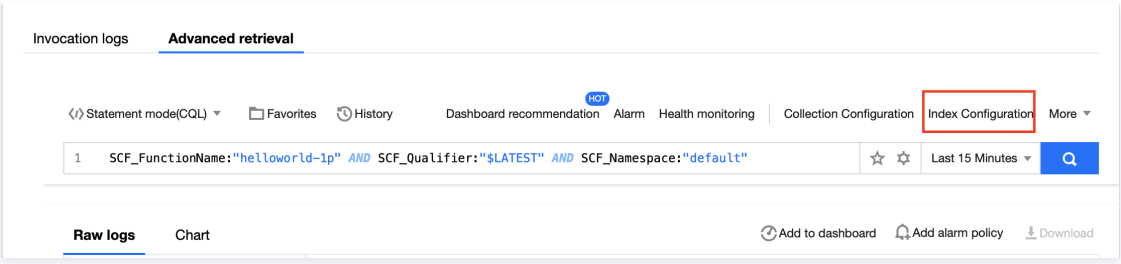


4. Click **Confirm**.

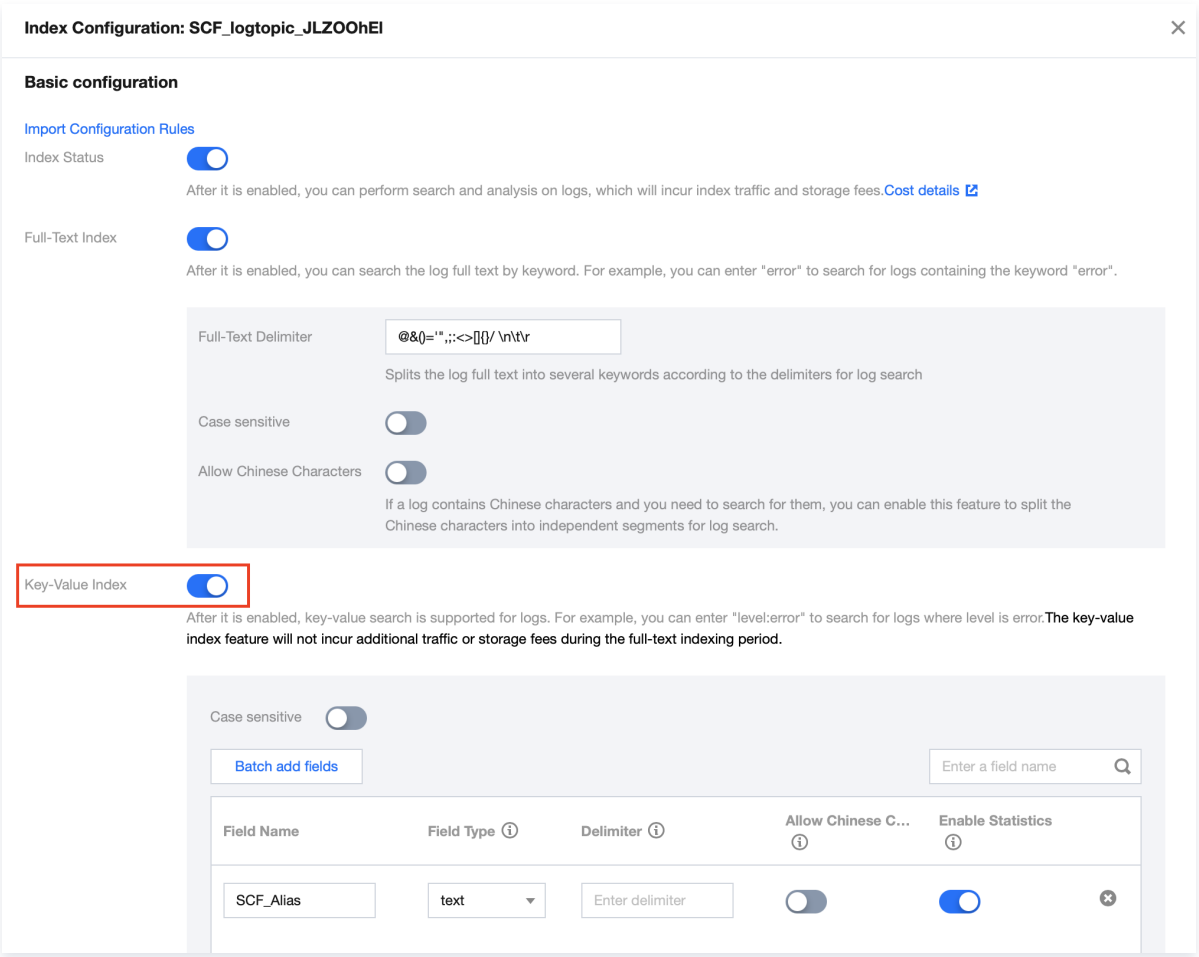
Index configuration

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 exceptional and logs cannot be viewed properly, please configure the index in the following steps:

1. Log in to the Serverless console and select Function Service from 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. In the "Log Query" tab, select "Index Configuration" under "Advanced retrieval". As shown in the figure below:



4. In the "Index Configuration" page, enable "Index Status" and "Key-Value Index". As shown in the figure below:



This configuration method is only effective for scenarios where there are function call logs in the log topic. If there are no function call logs in the log topic, please refer to the following table to manually configure **Key-Value Index**.

Field	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	Function execution status code.
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**.

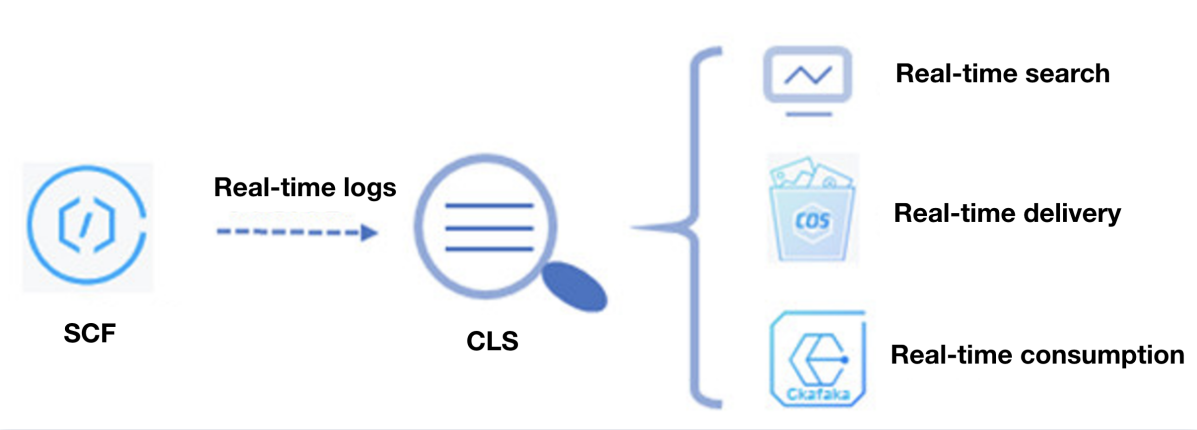
Log Delivery Configuration (Legacy)

Last updated: 2023-09-27 21:09:22

> **Description**
> Starting from January 29, 2021, SCF has been fully integrated with Tencent Cloud Log Service CLS. Function call logs created after this date will be delivered to CLS by default and support real-time output. If your function was created before January 29, 2021, and you need to perform log retrieval and delivery, please refer to this document to use this feature.

Overview

When using SCF for function computing, a large number of function operation logs will be generated. If you need to persistently store, deliver, or consume logs, and monitor and alert on log content, you can deliver logs to the Tencent Cloud Log Service (CLS) platform. As shown in the following figure:



Prerequisites

Before using the SCF real-time log service, you need to activate CLS first.

> **Description**
> For understanding the related limitations of the log service, please refer to the Specifications. Exceeding these limitations may result in log loss.

Procedure

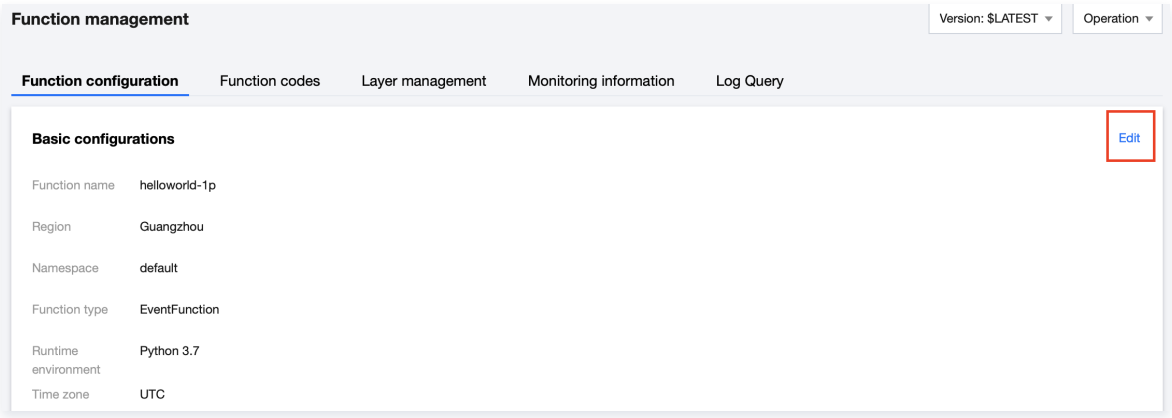
Creating log topics

Log in to the Log Service Console and Create a Log Topic. This document uses the creation of the `SCF-test` log topic in Guangzhou as an example.

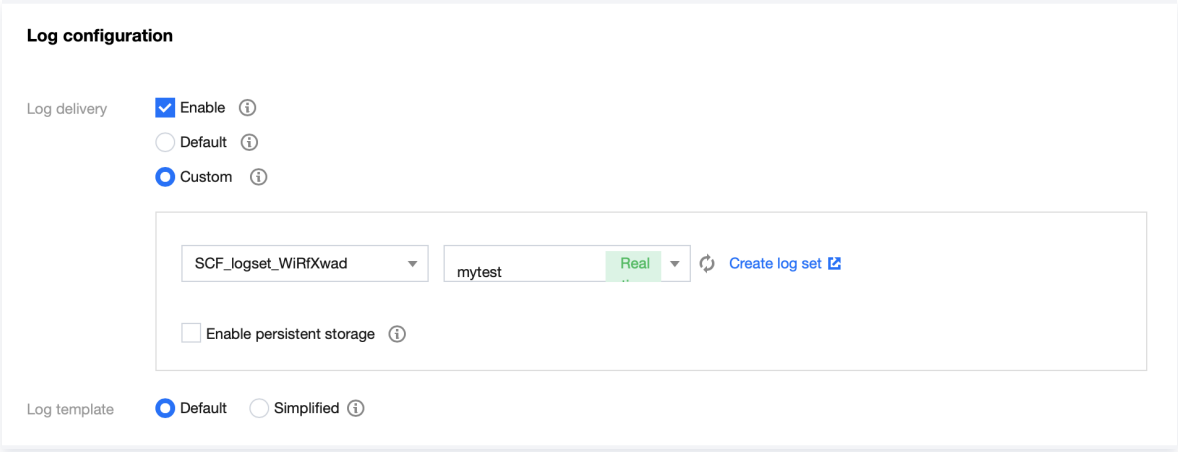
> **Attention**
> For the logset region, please select the region where the SCF service is located. Cross-region log push is not supported currently.

Configuring CLS

1. Log in to the Serverless console and select Function Service from 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 upper right corner. As shown below:



4. In "Log Delivery", check "Enable" and select the logset and log topic that have been created for this function. In this example, we use `SCF-test`. As shown below:

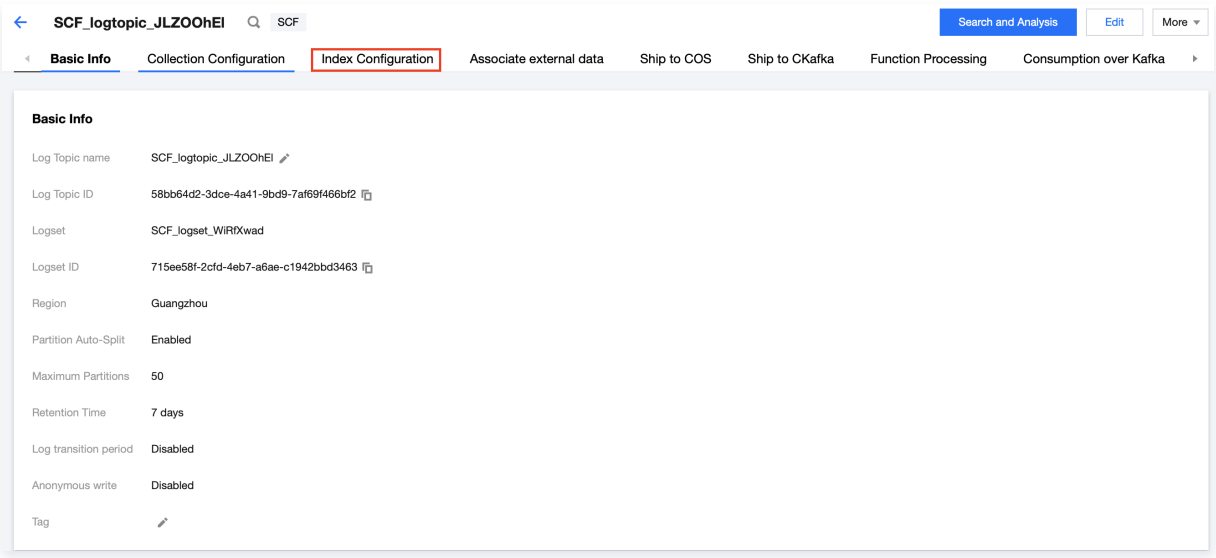


5. Click **Save** to successfully connect to the log service platform.

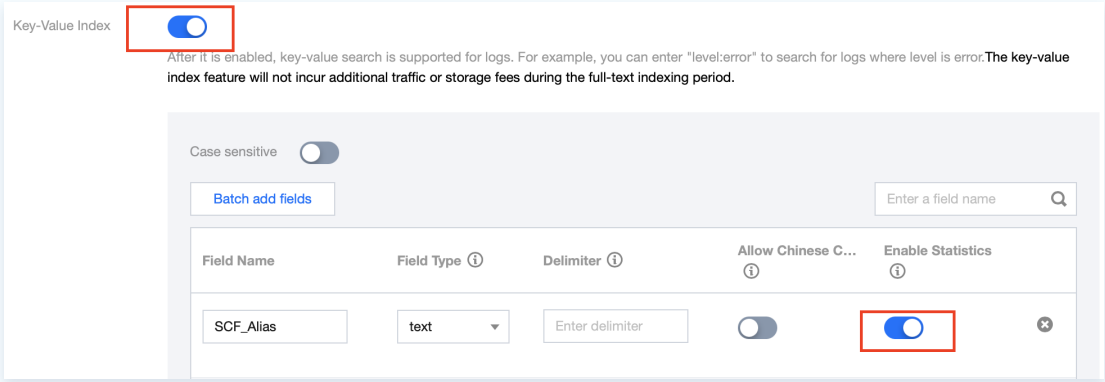
Enabling index

Log retrieval depends on the index configuration of the log topic. After the function is associated with the log topic, SCF automatically configures the index for the log topic. If an index anomaly still causes log retrieval to fail, please refer to this step to adjust the index configuration.

1. Log in to the log service console and select Log Topic from the left navigation bar.
2. Click on the ID of the created log topic to enter the "Basic Information" page.
3. Select **Manage** on the right side of the row where the log topic is located to enter the "Basic Information" page of the log topic.
4. On the "Basic Information" page of the log topic, click **Index Configuration**. As shown below:



5. Click **Edit** in the upper right corner, turn on "Key-Value Index" and add "Field Name" and "Field Type" according to the table below.

> **Description**
> For functions configured with log service, to ensure the display effect of logs in the SCF console, please enable the "Open Statistics" capability for fields in the **Key-Value Index** configuration. As shown below:
>
> 
>

Field	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	Function execution status code.
SCF_RetryNum	long	Number of retries.

For more information on other features such as real-time log search, delivery, and consumption, please see the CLS documentation. These features can be used in the CLS console directly.

Concurrence Management
Concurrency Overview

Last updated: 2023-09-28 14:53:07

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, please 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 time = Requests per second × Average time per request
You can view the average time per request in the "Execution Time" section of the monitoring information.
For instance, if a service has a QPS of 2000 and the average time per request is 0.02s, the concurrency at any given moment is 2000qps × 0.02s = 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 the SCF platform as needed; therefore, you cannot assume a certain retention duration when writing the function business code.

Concurrency expansion

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 a 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

The concurrent instances of the SCF platform need to go through an initialization process during elastic scale-out, including the initialization of the runtime environment and business code. You can use the **provisioned concurrency** feature to pre-configure concurrent instances. **The SCF platform will start launching concurrent instances as soon as you configure them, and it will not actively reclaim provisioned instances, ensuring as many concurrent instances as possible.** If a concurrent instance encounters errors such as memory leaks in the code, the SCF platform will replace it with a new instance. For more information, see Provisioned Concurrency.

Concurrency service level

Limit on Concurrency Scale-out

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.

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

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 this function**, and the sum of the concurrency quotas for all versions should be less than or equal to the reserved quota.
- After the concurrency quota is allocated to this function, it will be **exclusive to** this function and will no longer be provided to other functions.

Concurrency monitoring

When a concurrent instance of a function is processing actual requests, it will be marked as running concurrent instance. In SCF monitoring information, you can query the running concurrent instances of a function, a specific function version, or an alias. As there are certain intervals in running instance information collection, if a function's execution time is very short and its number of concurrent instances is high, the current monitoring data may not be completely accurate.

Use case

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. If a function generates a surge of business invocations, it can make full use of the unused quota to ensure that the surge will not cause 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: 2023-09-28 15:59:49

The SCF platform provides concurrency management capabilities at the function granularity to allow you to flexibly control the concurrency of different functions.

Concurrency Management

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

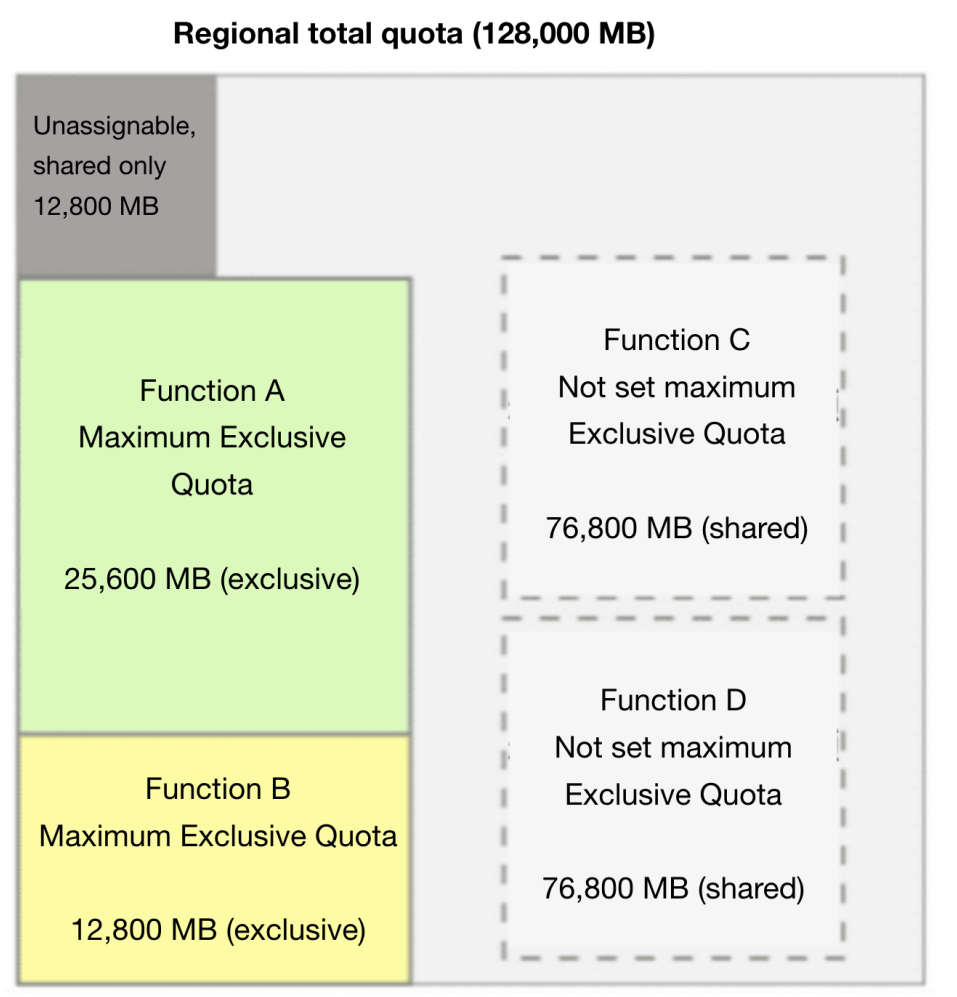
```
Account-Level concurrency quota
 |- Function-Level reserved quota
```

> **Note**
> Provisioned concurrency is not included in the concurrency management capabilities; instead, 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, please 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.
You can utilize the Reserved Concurrency Quota of a function to allocate the region-level concurrency to a specific function, thereby managing the function's concurrency. To prevent functions without a set reserved concurrency quota from being unable to be invoked after all account-level quotas have been allocated, the SCF platform reserves 12,800MB of the account-level

concurrency quota as unassignable and exclusively for functions without configured reservations. As shown in the following figure:



Reserved quota

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

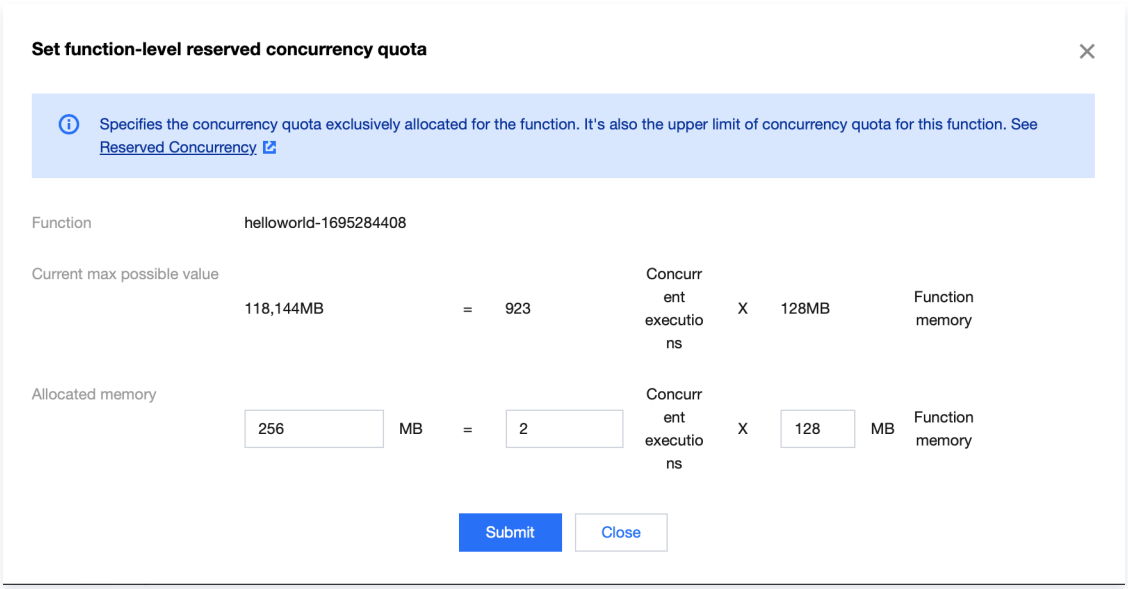
- **The reserved quota is the upper limit of the concurrency quota of this function.** The sum of the concurrency quotas of all versions is less than or equal to the reserved quota.
- After the concurrency quota is allocated to this function, it will be **exclusive to** this function and will no longer be provided to 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.
Setting the function reserved quota will occupy the concurrency quota at the region level. If the unoccupied quota at the region level (region-level quota - reserved quotas allocated to other functions - 12,800 MB) is insufficient, it cannot be set.

Configures the reserved quota

Follow the steps below to set the desired reserved quota for the function.

1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
2. From the "Function Service" list page, select the function name that needs to be configured and proceed to the "Function Management" page.
3. Select **Concurrency Quota** on the left, and in "Reserved Quota", click **Set** in the upper right corner.
4. In the pop-up "Set function-level reserved concurrency quota" window, configure the desired reserved quota and click **Submit**. As shown below:



After setting, you can view the configuration status in the "Reserved Quota" page of the "Concurrency Management" page.

Deleting maximum dedicated quota

When you no longer plan to use the maximum reserved quota, you can delete it. After deleting the maximum reserved quota, the function will share the concurrency limit of the account dimension with other functions.

> **Note**
> Deleting the reserved quota and setting the it to 0 are different configurations.
> - Deleting the maximum reserved quota: The function does not have an exclusive quota and uses the shared quota in the region. The upper limit is determined by the usage of the shared quota.
> - Reserved quota set to 0: The function has an exclusive quota of 0, the function's concurrency limit is 0, the function cannot run, and it stops responding to trigger events.

1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
2. From the "Function Service" list page, select the function name that needs to be configured and proceed to the "Function Management" page.
3. Select **Concurrency Quota** on the left, and in "Maximum Reserved Quota", click **Delete** on the right of the page.
4. In the pop-up window "Delete Function's Maximum Reserved Quota", click **Confirm**.

Provisioned Concurrency

Last updated: 2023-09-28 16:02:34

Provisioned concurrency can start concurrent instances in advance according to the configuration. SCF will not repossess these instances; instead, it will ensure as much as possible that a corresponding number of concurrent instances are available to process requests.
You can use this feature to set the quota of provisioned concurrent instances for a specified function version, so as to prepare computing resources in advance and reduce the duration for cold start and initialization of runtime environment and business code.

Overview

Provisioned concurrency addresses the problem of concurrent instance initialization when requests are received at the version level. After you configure provisioned concurrency for a function version, the following effects will be achieved:

1. SCF will **immediately start concurrent instances** until the configured value is reached.
2. SCF will **not repossess provisioned concurrent instances**, and it will guarantee a 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 will not actively reclaim provisioned concurrent instances, but instances may become unavailable due to process termination or memory overflow. Once an instance becomes unavailable, SCF will reclaim it and prepare a new concurrent instance to meet the provisioned concurrency configuration. During this period, the actual number of concurrent instances may temporarily be less than the provisioned concurrency. Unstarted concurrent instances will not be included in the billing scope. You can view the provisioned concurrency startup situation in the "Concurrent Executions and Provisioned Concurrency" chart in the function's monitoring information.
Provisioned concurrency **can only be configured on published versions and cannot be configured on the `$LATEST` version**. The `$LATEST` version is editable, while provisioned concurrency requires concurrent instances to be launched before requests arrive. To ensure business stability and avoid inconsistencies due to code and configuration edits, provisioned concurrency can only be configured on published versions. The code and configuration of published versions cannot be modified, making them suitable for production environments. For more details, please refer to 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.
Take a function version with a configured memory of 128 MB as an example:

Scenario	Average concurrency	Provisioned concurrency	Function reserved quota	Result
Default condition	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 disablement	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	10,240 MB (80	19,200 MB (150	80 concurrent instances don't need to be initialized, and 20 concurrent instances need to be initialized when they are

	instances	concurrent instances	concurrent instances)	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.**

Procedure

Adding provisioned concurrency

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

> ⚠ **Note**
> Provisioned concurrency can be set for a function version only after the version is published.

1. Log in to the Serverless console and click Function Service on the left navigation bar.
2. On the "Function Service" list page, click the target function name to enter the "Function Management" page.
3. Select **Concurrency Quota** on the left, and in the "Provisioned Concurrency" page, click **Add Provisioned Concurrency**.
4. In the pop-up "Add Function Provisioned Concurrency" window, select the desired version and the number of provisioned concurrencies, then click **Submit**.
   After the settings are complete, you can view the configuration status in "Provisioned Concurrency". The SCF backend will take some time to complete the expansion of provisioned concurrency, and the number of prepared concurrent instances and completion status will be displayed in the list.

Updating provisioned concurrency

After the SCF backend adds the provisioned concurrent instances, you can modify the number of concurrent instances as needed.

1. Log in to the Serverless console and click Function Service on the left navigation bar.
2. On the "Function Service" list page, click the target function name to enter the "Function Management" page.
3. Select **Concurrency Quota** on the left, and in the "Provisioned Concurrency" page, click **Settings** on the right of the row of the version to be updated.

4. In the pop-up "Set Function Provisioned Concurrency" window, update the settings and click **Submit**.
   After the settings are complete, the SCF platform will, based on your modifications, increase or decrease the number of concurrent instances within a certain period of time.

Deletes provisioned concurrency

If you no longer use a provisioned concurrency configuration, you can delete it.

1. Log in to the Serverless console and click Function Service on the left navigation bar.
2. On the "Function Service" list page, click the target function name to enter the "Function Management" page.
3. Select **Concurrency Quota** on the left, and in the "Provisioned Concurrency" page, click **Delete** on the right of the row of the target version.
4. In the pop-up "Delete Function Provisioned Concurrency Quota" window, click **Confirm**.
   After the configuration is deleted, the SCF backend will gradually reclaim the concurrent instances.

Relevant Operations

Utilizing Provisioned Concurrency for Traffic Switching

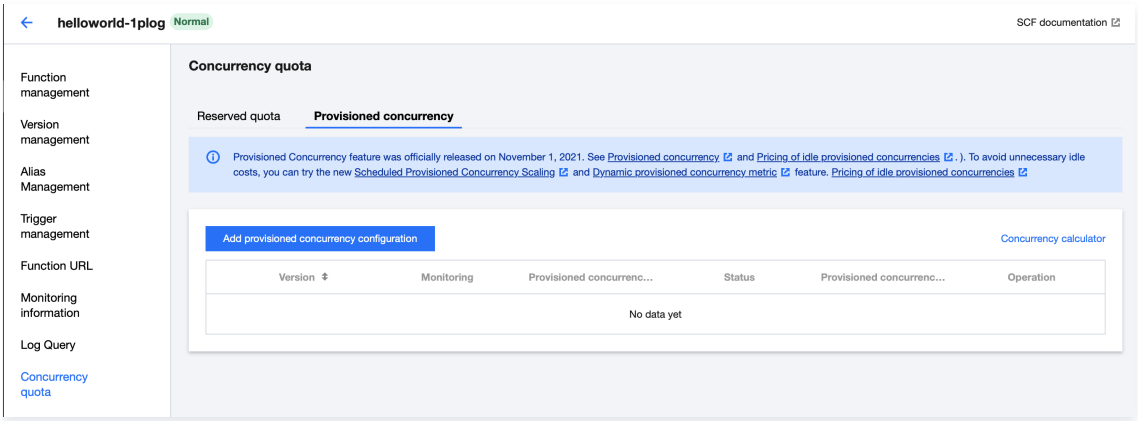
You can set the reserved quota for a function based on the volume of concurrent business requests and configure the provisioned concurrency based on the traffic switch needs as instructed below:

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 concurrency	Provisioned concurrency	Function reserved quota	Result
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.



Scheduled Provisioned Concurrency

Last updated: 2023-09-28 16:02:43

Overview

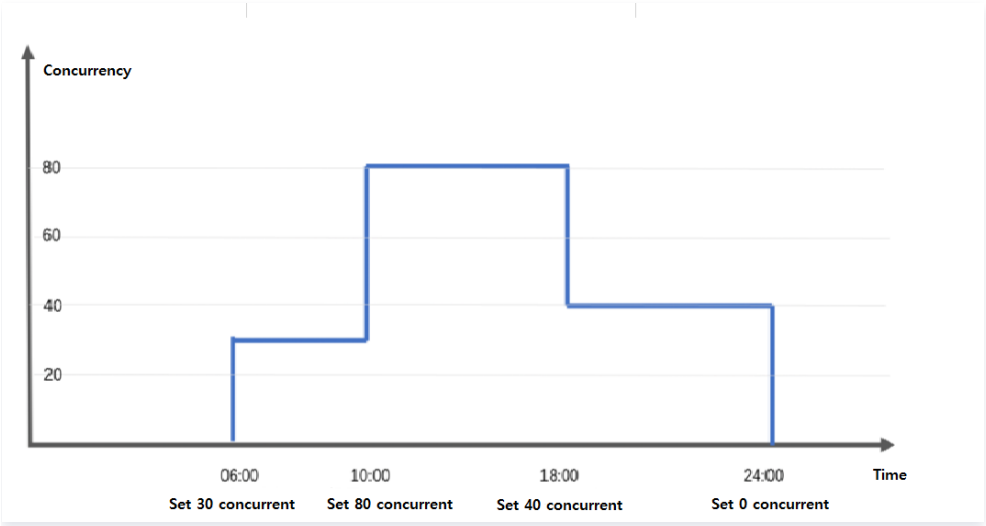
Scheduled provisioned concurrency is an elastic policy for provisioned concurrency. You can reasonably configure provisioned concurrency based on the business conditions and upgrade/downgrade it at specified times to improve the utilization of provisioned concurrent instances and reduce the fees incurred 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.

Applicable Scenario

- 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 scheduled provisioned concurrency is set as the target value for a specific time period. For instance, if you need to set four scheduled tasks according to business requirements, such as setting 30 concurrent instances at 6:00, 80 concurrent instances at 10:00, 40 concurrent instances at 18:00, and 0 concurrent instances at 24:00, the final fluctuation of provisioned concurrency would be as follows:



Description

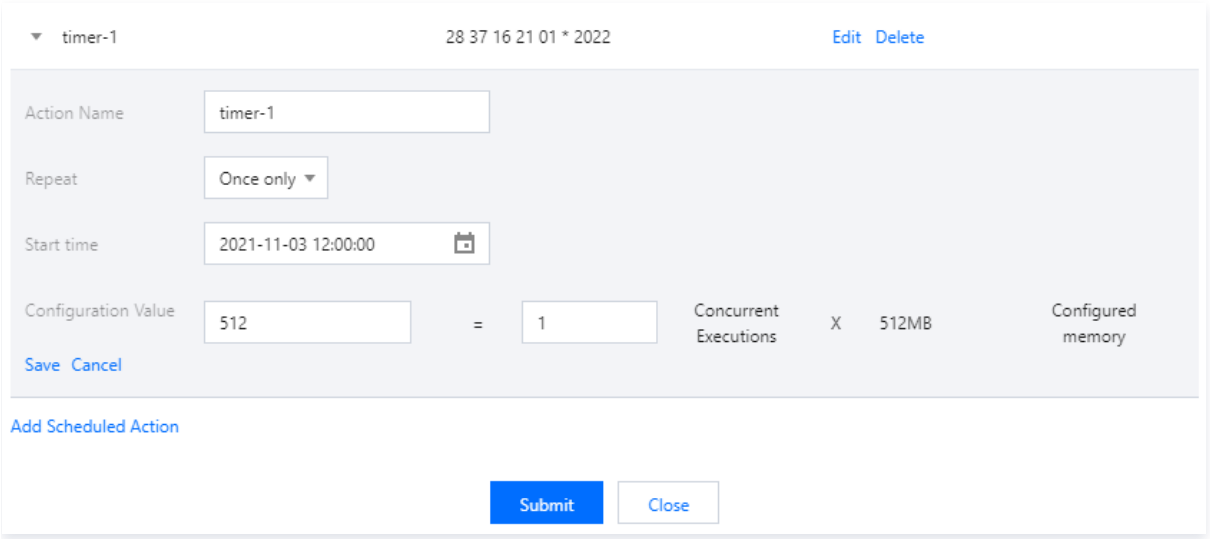
- A cron expression for scheduled provisioned concurrency has seven required fields separated with spaces. For more information, see Timer Trigger Description.
- There is a limit to the number of scheduled provisioned concurrency tasks under one user account on one function version. For more information, 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.

Scenario 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:

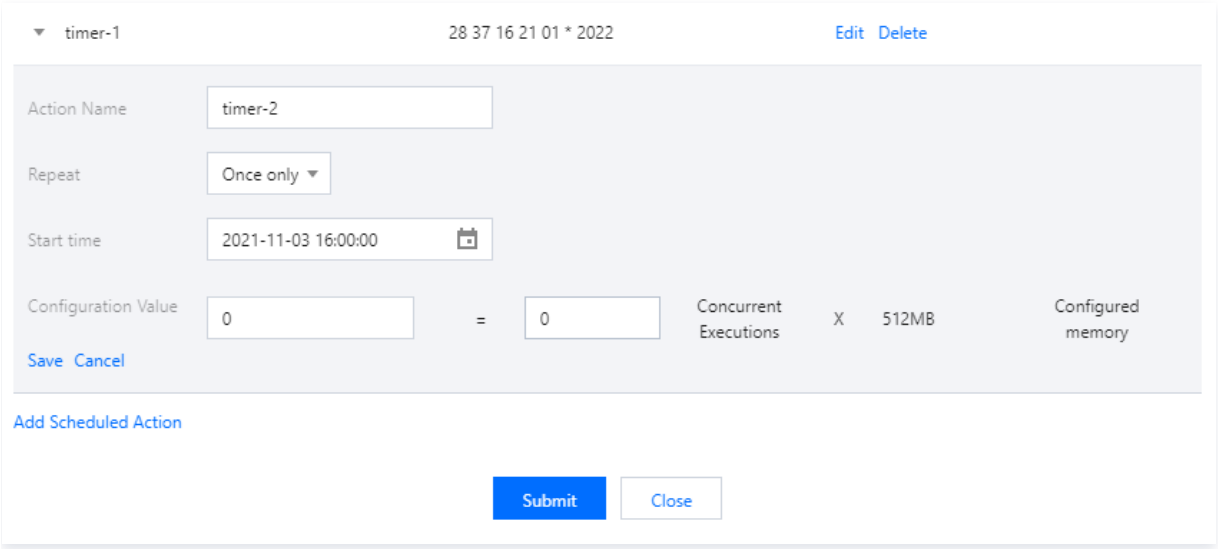
Starting scheduled task

To schedule a provisioned concurrency task, you need to create a new scheduled task, select the start time, and set the provisioned concurrency to the target value. The specific operations are as follows:



Ending scheduled task

To schedule the termination of a provisioned concurrency task, you need to **add an additional scheduled provisioned concurrency task**, select the end time, and change the provisioned setting value to 0. The specific operations are as follows:



Dynamic Provisioned Concurrency Metric

Last updated: 2023-09-28 16:02:50

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.

Applicable Scenario

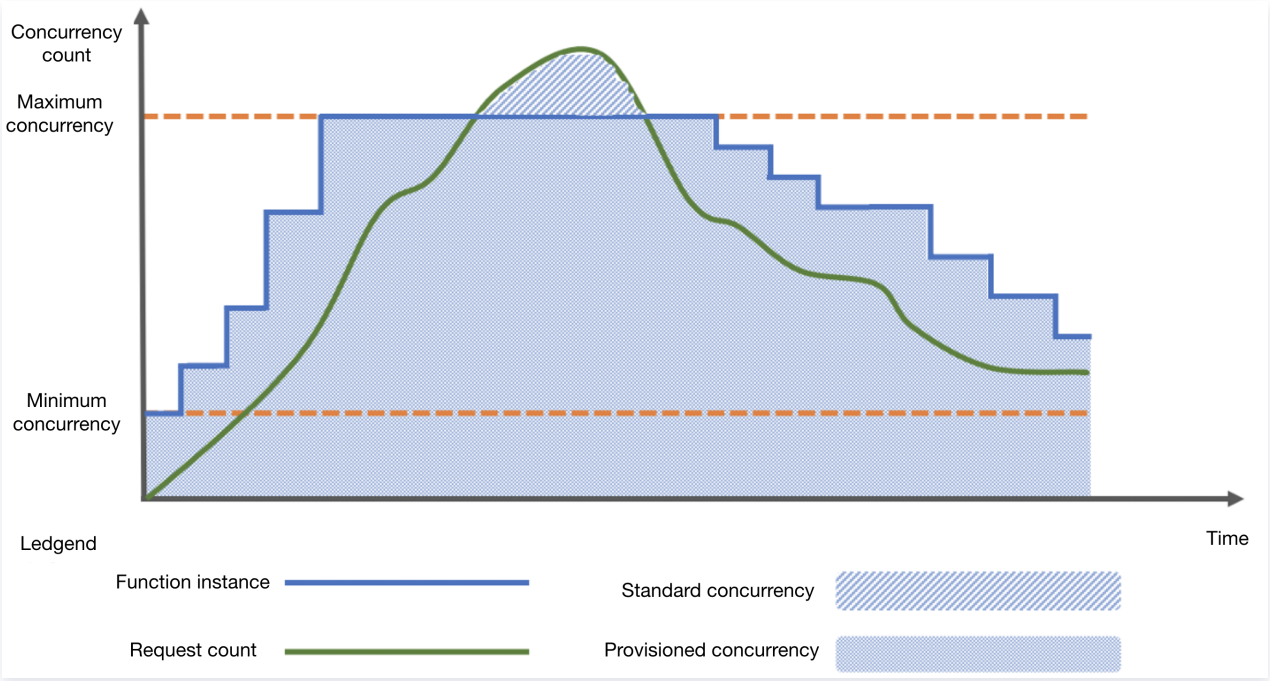
- Businesses that are sensitive to idle provisioned concurrency fees.
- Functions that are sensitive to cold start with unpredictable business traffic peaks.

Implementation Principle

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

- **Scale-out**: As the actual request volume of the business continues to increase, the system begins to scale out when the scale-out threshold is triggered. The scale-out operation stops when the maximum concurrency limit is reached. Requests exceeding this part will be scaled out in a pay-as-you-go mode.
  Scale-out frequency: A scale-out operation is performed every 10 seconds, and there is no window time for scale-out.
- **Scale-in**: As the actual request volume of the business continues to decrease, the system begins to scale in when the scale-in threshold is triggered. The scale-in operation stops when the minimum concurrency limit is reached.
  Scale-in frequency: During scale-in, a relatively conservative scale-in process is implemented through a 10-minute window time. That is, after performing a dynamic scaling operation, no further scale-in operation will be performed within the window time. This can be understood as a cooldown period similar to waiting to use a skill. If no scaling operation has been performed before, a scale-in operation can be performed in 10 seconds.



Provisioned Target Value

The provisioned target value is determined by the current concurrency and the target concurrency utilization metric.

- **Provisioned Target Value** = Current total function instances × Current concurrency utilization ÷ Target concurrency utilization = Current total function instances × (Current concurrency ÷ Current total function instances) ÷ Target concurrency utilization = **Current concurrency ÷ Target concurrency utilization metric**
- 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 Utilization

Concurrency utilization of a function refers to the proportion of the concurrent value of requests currently being responded to by function instances to the total number of function instances. The metric value range is [0,1).

Minimum Concurrency

The minimum concurrency represents the least number of concurrent instances that need to be provisioned for the function, i.e., the lower limit for scale-in.

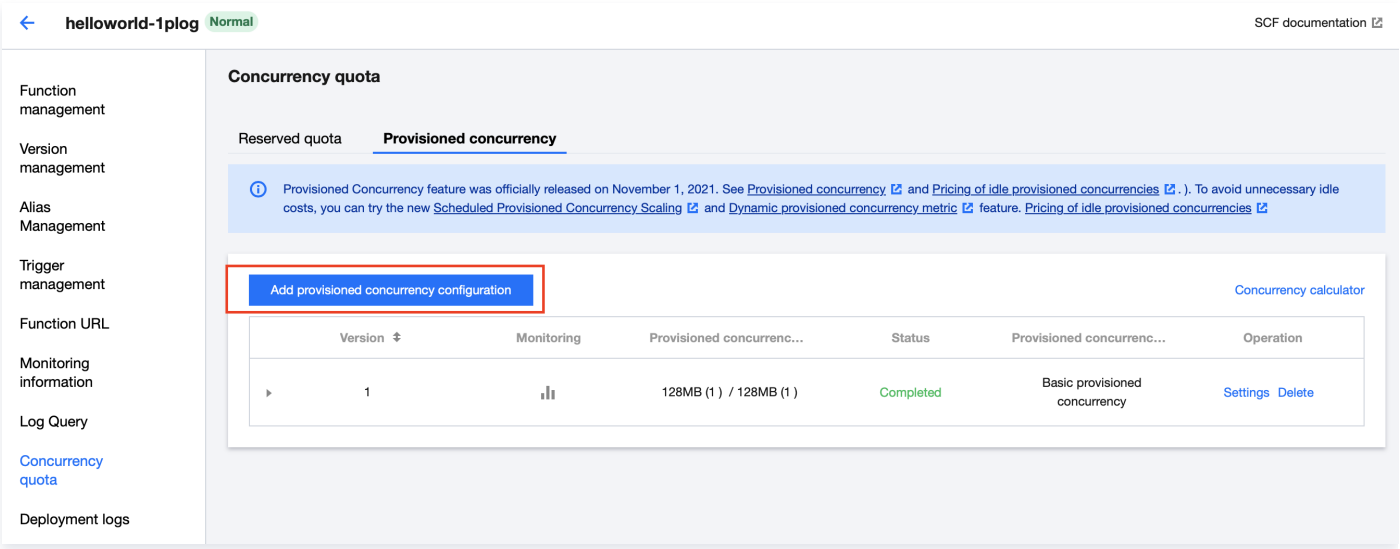
Maximum Concurrency

The maximum concurrency represents the maximum number of concurrent instances that can be provisioned for the function, i.e., the upper limit for scale-out.

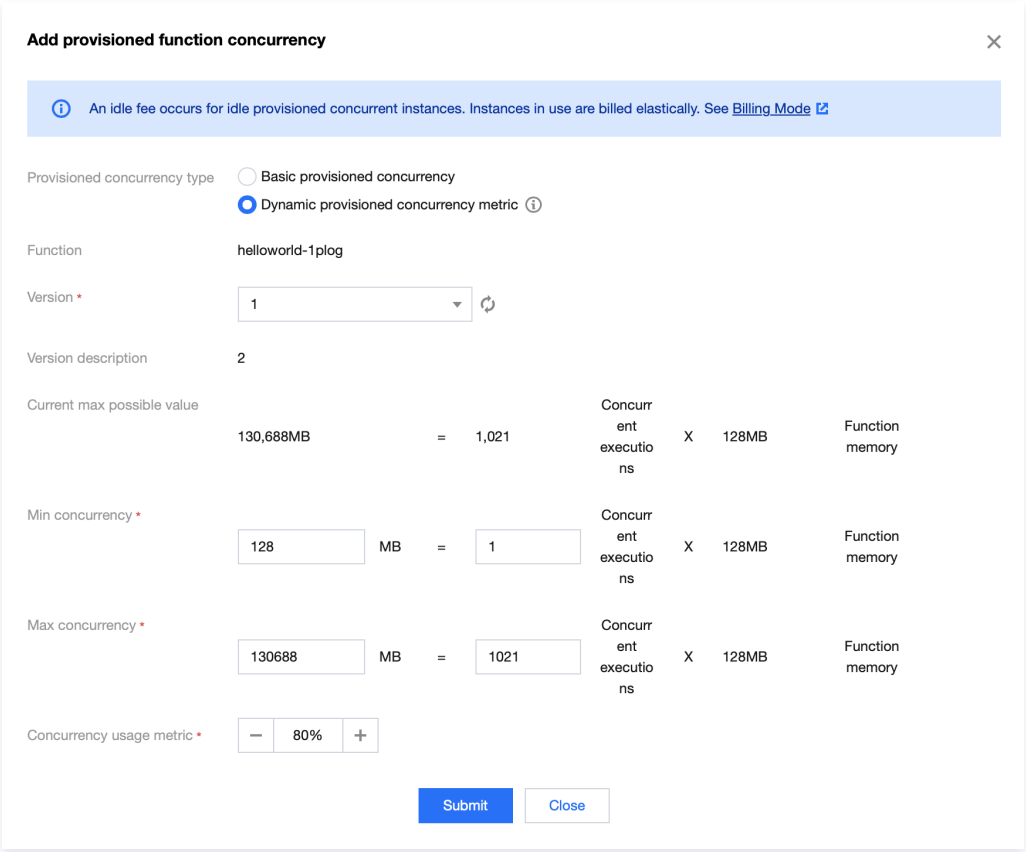
Procedure

Adding dynamic provisioned concurrency metric

1. Log in to the Serverless console and select **Function Service** on the left.
2. From the "Function Service" list page, select the function name that needs to be configured and proceed to the "Function Management" page.
3. Select **Concurrency quota** > **Provisioned concurrency** on the left to enter the "Provisioned Concurrency" page.
4. On the "Provisioned Concurrency" page, click **Add provisioned concurrency configuration** as shown below:



5. In the pop-up "Add provisioned function concurrency" window, select the provisioned type as dynamic metric provisioned and function version. Set the minimum concurrency, maximum concurrency, and target concurrency utilization metric according to the business scenario, then click **Submit** as shown below:

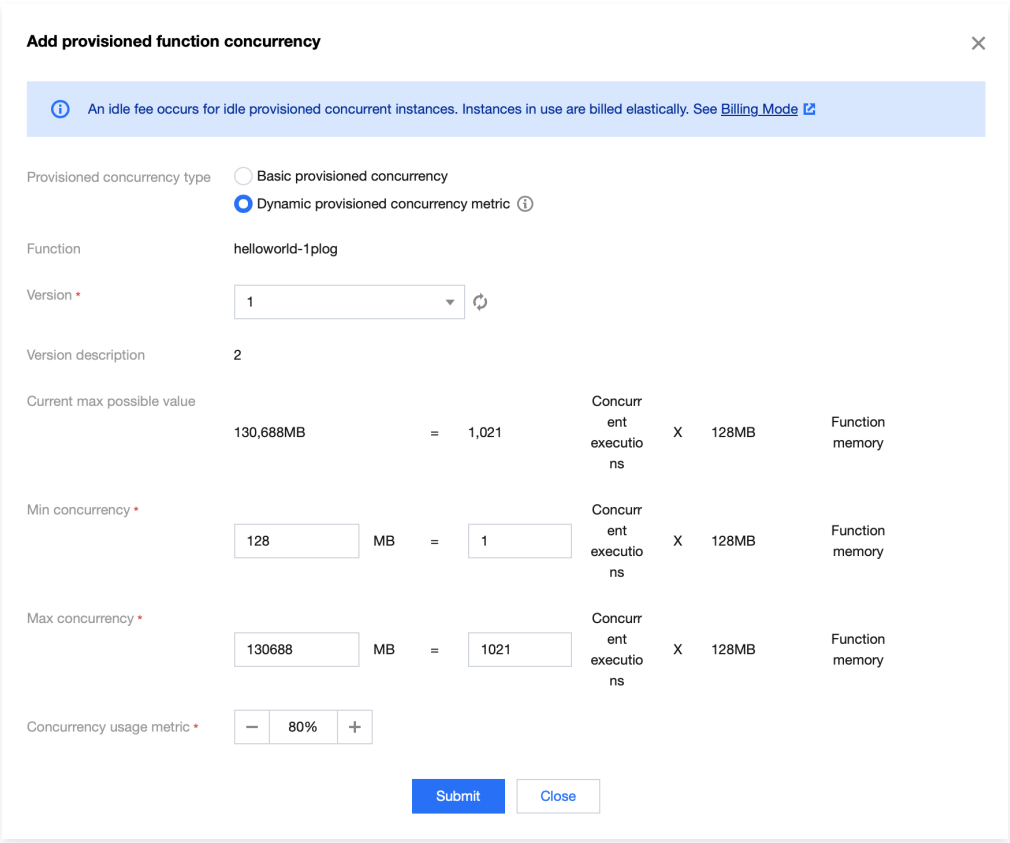


After setting, you can check the configuration status in "Provisioned Concurrency". The cloud function backend will take some time to complete the expansion of the provisioned concurrency, and the number of started and prepared concurrencies and the completion status will be displayed in the list.

Updating dynamic provisioned concurrency metric

When updating the dynamic provisioned concurrency metric, you can modify parameters such as the provisioned type, minimum concurrency, maximum concurrency, and target concurrency utilization metric.

1. Log in to the Serverless console and select **Function Service** on the left.
2. From the "Function Service" list page, select the function that needs concurrency provision updating, and enter the "Function Management" page.
3. Select **Concurrency Quota** > **Provisioned Concurrency** on the left to enter the "Provisioned Concurrency" page.
4. On the "Provisioned Concurrency" page, locate the version to update and click **Settings** on the right.
5. In the pop-up "Add provisioned function concurrency" window, update the settings and click **Submit** as shown below:



> **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 with provisioned concurrency that you want to delete and proceed to the "Function Management" page.
3. Select **Concurrency Quota** > **Provisioned Concurrency** on the left to enter the "Provisioned Concurrency" page.
4. On the "Provisioned Concurrency" page, select **Delete** on the right of the row where the version to be adjusted is located. As shown in the figure below:

5. In the pop-up window for "Delete Function Provisioned Concurrency Quota", click **Confirm** to proceed.

Concurrency Overrun

Last updated: 2024-06-14 14:56:55

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

Async invocation includes Cloud API triggers, COS triggers, Timer triggers, CMQ Topic triggers, CLS triggers, and MPS triggers, etc. For specific trigger invocation types, please refer to the relevant trigger documentation.
When async invocation exceeds the concurrency limit, the SCF function will automatically retry. For more details, please refer to Async Invocation Retry Strategy.

Sync invocation

Sync invocation includes Cloud API triggers for sync invocation, API Gateway triggers, and CKafka triggers.
Since the error information is directly returned to the user during the sync invocation process, the platform will not automatically retry when an error occurs in sync invocation. The retry policy (whether to retry and how many times to retry) is determined by the caller. In the case of sync invocation, the SCF function will return a 432 status code, and the request will not be retried.

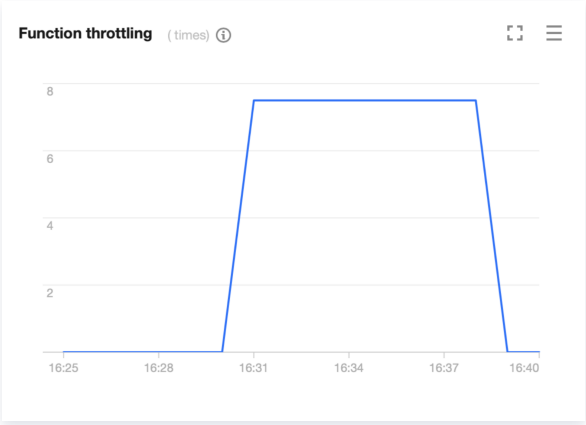
Concurrency Overrun Troubleshooting

Viewing concurrency overrun monitoring data

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

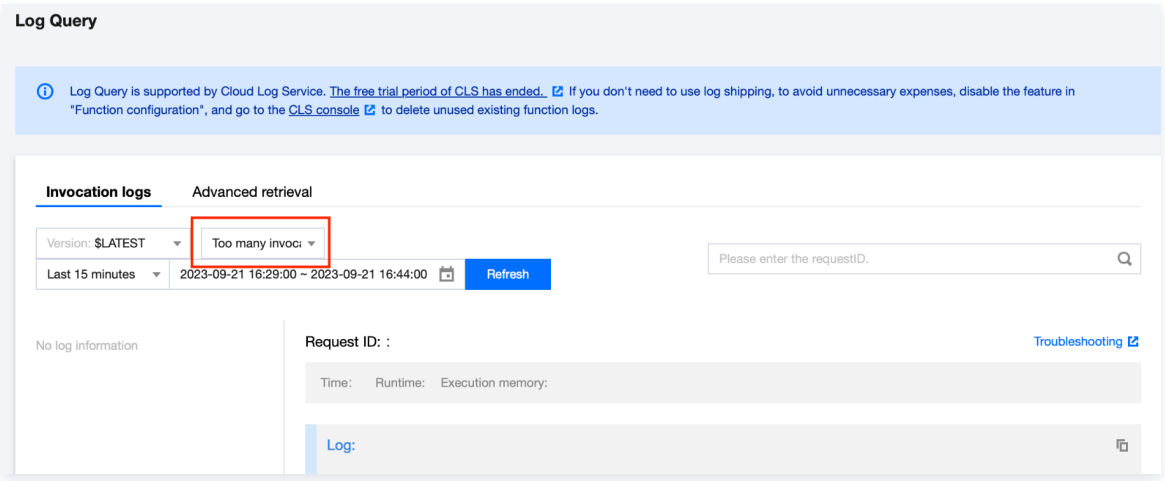
Viewing the number of limited times

1. Log in to the Serverless console and select **Function Service** on the left.
2. In the **Function Service** page, select the function you want to view to access its details page.
3. In **Function Management**, select **Monitoring Information** > **Limited Times** to view the number of limited times for the relevant function, as shown in the figure below:



Viewing function limit log

1. Log in to the Serverless console and select **Function Service** on the left.
2. In the **Function Service** page, select the function you want to view to access its details page.
3. In **Log Query**, select **Invocation logs** to view the specific limit logs for the relevant function, as shown in the figure below:



Fixing concurrency overrun

- **Asynchronous Invocation** has a platform retry strategy for concurrency overrun scenarios, which automatically handles and retries concurrency overruns. Generally, users do not need to take any action for concurrency overruns in asynchronous invocations. Within the set maximum waiting time, the function platform will automatically retry concurrency overrun errors.
- When an error occurs during a **Synchronous Invocation**, the error message is directly returned to the user, and the request will not be retried.

> **Note**
> In asynchronous invocations, if you are sensitive to timeliness, you can reduce or mitigate the impact of overruns on the business system by configuring the maximum exclusive quota. For instance, if it is crucial that important messages are not lost after exceeding the set maximum retention time, you should configure a dead letter queue as a fallback.

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 maximum reserved quota is the maximum limit that ensures the available concurrency of a function. By configuring this quota, the function can launch a sufficient number of concurrent instances within the limit, up to the configured quota. By setting this maximum reserved quota, the function no longer shares the account concurrency quota with other functions, reducing the possibility of concurrency overrun and ensuring more reliable operation. For more information, please see Setting Maximum Reserved Quota.

Trigger Management
Creating Trigger

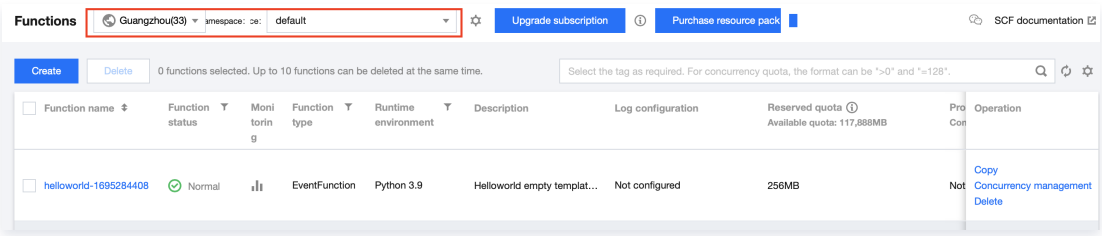
Last updated: 2023-09-28 16:08:05

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 on Serverless Cloud Framework CLI.

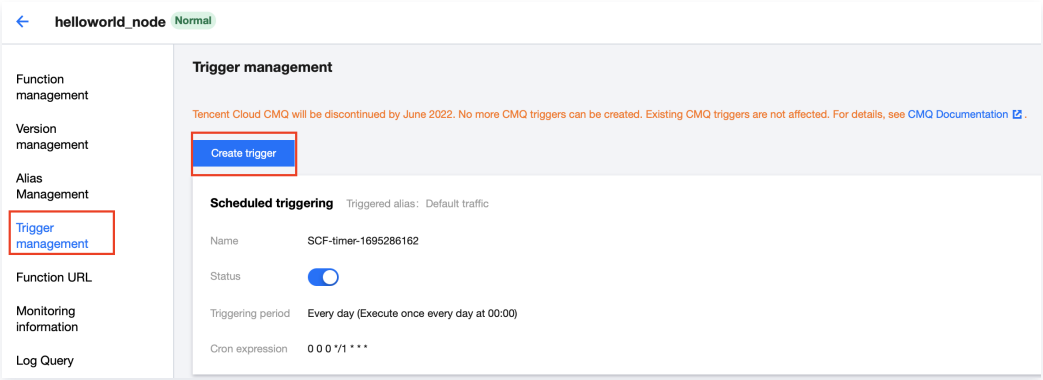
> **Note**
> HTTP-Triggered Functions only support the creation of API Gateway triggers. For more details, please refer to Creating HTTP-triggered Functions.

Creating a Trigger in the Console

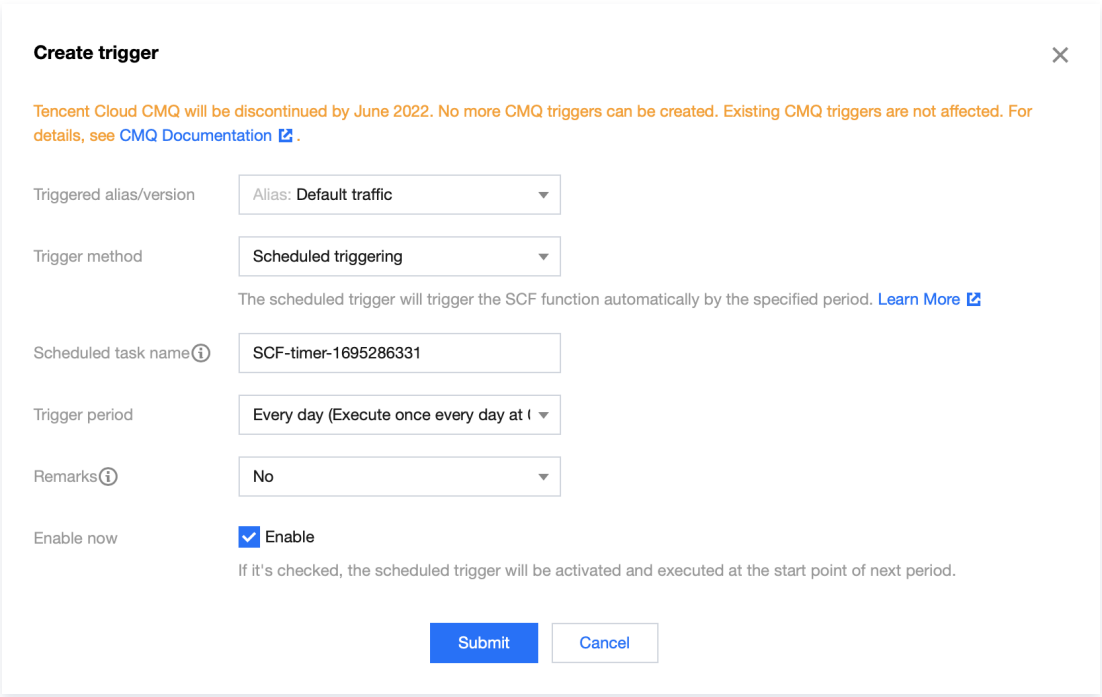
1. Log in to the Serverless console and click on Function Service in the left sidebar.
2. At the top of the "Function Service" list page, select the region and namespace where the function is located, as shown in the figure below:



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 interface. Click **Create trigger** to start creating a new trigger, as shown in the figure below:



5. In the "Create trigger" pop-up window, select the trigger alias/version and the trigger method, as shown in the figure below:



- 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 Trigger Overview.

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

Creating Trigger on Serverless Cloud Framework CLI

> **Description**
> 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.

Tutorial Video

The following video will guide you on how to create, delete, and manage triggers:
Watch video

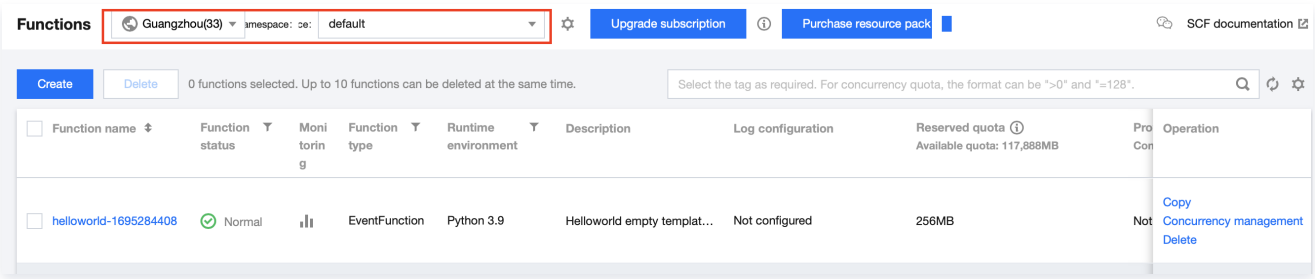
Deleting Triggers

Last updated: 2023-09-27 18:10:42

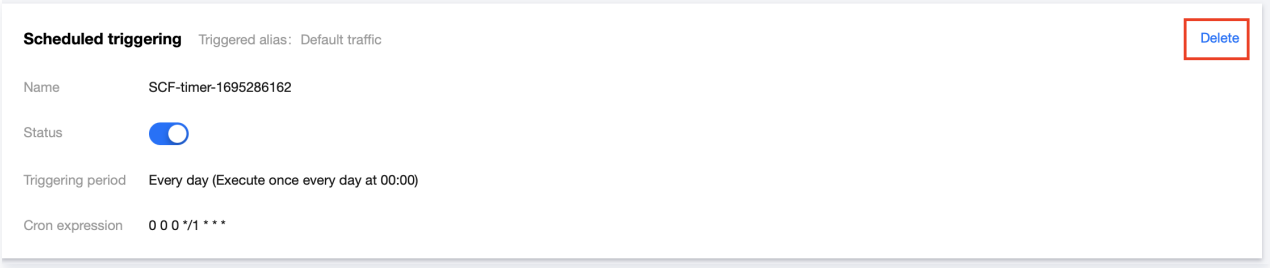
You can disassociate a function from an event source by deleting a trigger in the console. After disassociation, the event source will no longer trigger the function.

Deleting a Trigger in the Console

1. Log in to the Serverless console and click on Function Service in the left navigation bar.
2. At the top of the "Function Service" list page, select the region and namespace where the function is located, as shown in the figure 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 interface, then click **Delete** in the upper right corner of the trigger, as shown in the figure below:



Confirm the deletion in the pop-up window.

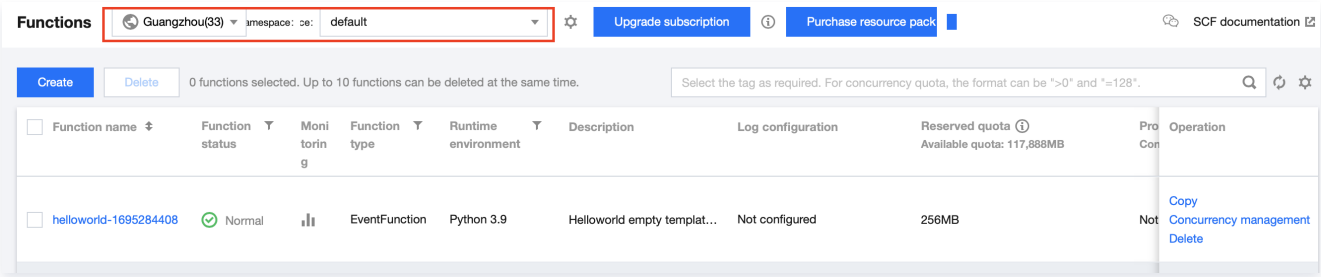
Enabling/Disabling Triggers

Last updated: 2023-09-27 18:10:55

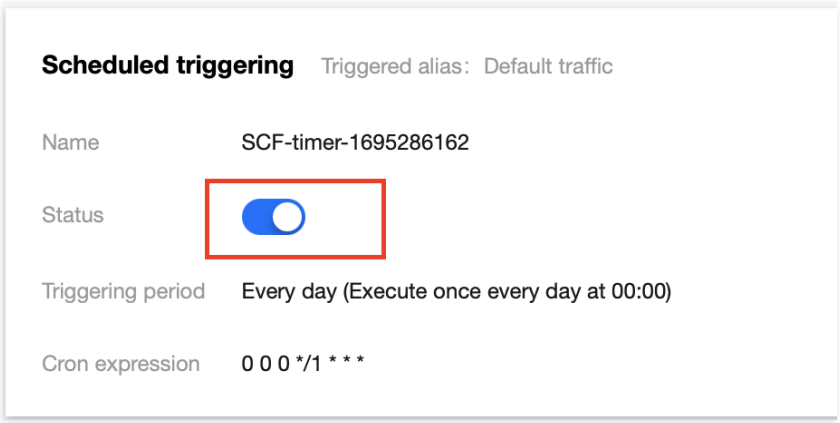
You can disable a trigger to temporarily prevent a function from being triggered by an event occurring at the event source. This document describes how to do so in the console.

Enabling or Disabling a Trigger in the Console

1. Log in to the Serverless console and select Function Service from the left navigation bar.
2. At the top of the "Function Service" list page, select the region and namespace where the function is located, as shown in the following figure:

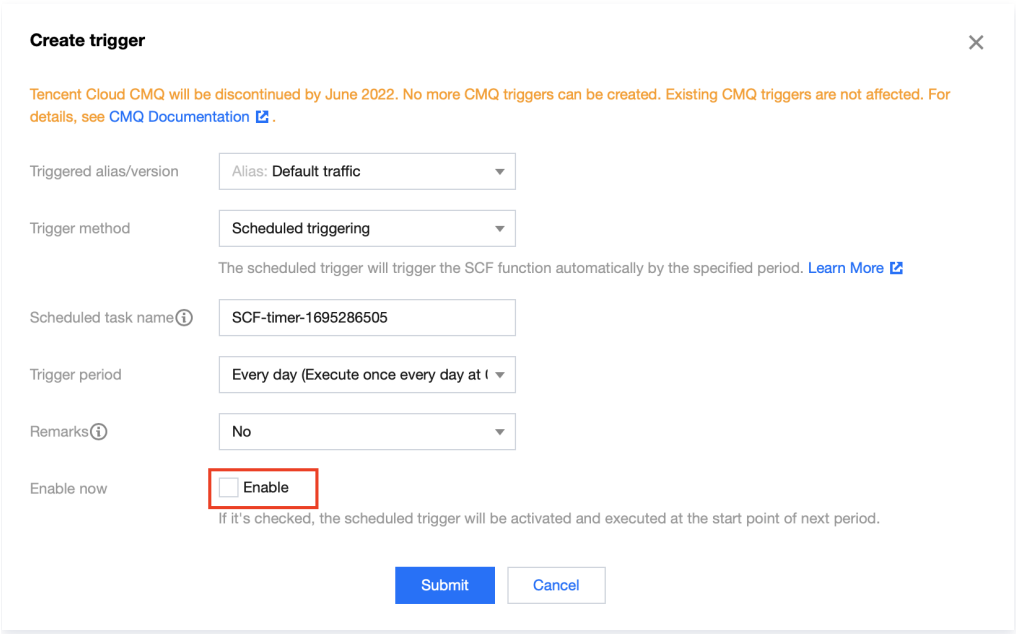


3. Click the function name to enter the function details page.
4. Select **Trigger Management** on the left to enter the trigger browsing and operation interface. Click on the in the "Status" of the trigger you wish to enable or disable to toggle its status. As shown in the figure below:



Setting the Enables/Disabled Status of a Trigger During Creation

During the creation of a trigger, you can set its enabled or disabled status. Once the trigger is created, it will be in the set status. For instance, when creating a timed trigger, if you wish for the trigger not to take effect immediately but to be activated later as needed, you can deselect **Enable Now**. As shown in the figure below:



After the trigger is created, you can activate it by toggling the enabled/disabled status.

Precautions

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: 2023-09-27 18:11:17

Feature Overview

The function URL is the dedicated HTTP(S) endpoint of a function. Once the function URL is configured for a function, it can be invoked through its HTTP(S) endpoint using a web browser, curl, Postman, or any HTTP client.
You can create and configure function URLs through the SCF console or SCF API/CLI. Once a function URL is created, its URL endpoint remains constant. The endpoint format of the function URL is as follows:

```
Public Network: https://<app-id>-<url-id>-<region>.scf.tencentcs.com
Private Network: https://<app-id>-<url-id>-<region>-in.scf.tencentcs.com
```

Function URLs exist at the same level as triggers and are applicable to both event functions and web functions. You can configure triggers such as API Gateways while enabling function URLs.
The function URL is bound one-to-one with the function's version and alias. You need to manually enable or disable the function URL for each version and alias. By default, the function URL is disabled.

Invoke parameters

Event-triggered function

Request Parameters

Upon receiving a request, the URL triggers the function to run, and the URL sends the relevant request information to the triggered function in the form of an event parameter. The relevant request information includes details such as the specific service and API rules that received the request, the actual path of the request, the method, and the request's path, headers, and query.

```
// Example of detailed Event information [Compatible with apigw protocol, remove headerParameters, isBase64Encoded,
pathParameters, queryStringParameters, requestContext related fields]:
{
  "body":"{\"test\":\"hello world\"}",
  "headers": {
    "accept":"*/*",
    "accept-encoding":"gzip, deflate, br",
    "cache-control":"no-cache",
    "connection":"keep-alive",
    "content-length":"17"
  },
  "httpMethod":"POST",
  "path":"/",
  "queryString":{
    "a":"1",
    "b":"2"
  }
}
```

Response Parameters

When the function returns a response, it parses the response and converts it into an HTTP response, with a 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 will assume the statusCode is 200, the content-type is application/json, and the body is the function response.
The standard response parameter format for the function is as follows:

Function output	HTTP Response (Content visible to the client)

`"Hello, world!"`	`HTTP/2 200`
	`date: Wed, 08 Sep 2021 18:02:24 GMT`
	`content-type: application/json`
	`content-length: 15`
	
	`"Hello, world!"`

`{`
`  "message": "Hello, world!"`
`}`
	`HTTP/2 200`
	`date: Wed, 08 Sep 2021 18:02:24 GMT`
	`content-type: application/json`
	`content-length: 34`
	
	`{`
	`  "message": "Hello, world!"`
	`}`

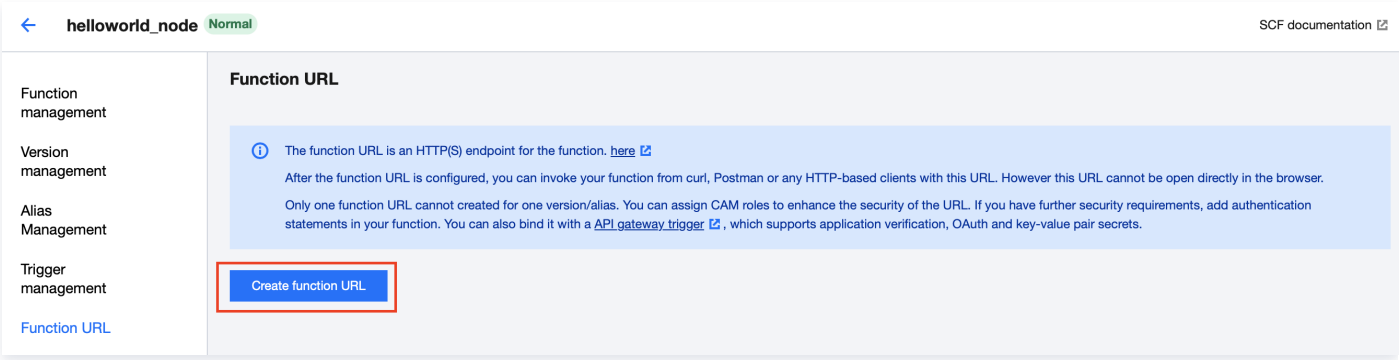
`{`
`  "statusCode": 201,`
`  "headers": {`
`  "Content-Type": "application/json",`
`  "My-Custom-Header": "Custom Value"`
`  },`
`  "body": JSON.stringify({`
`  "message": "Hello, world!"`
`  })`
`}`
	`HTTP/2 201`
	`date: Wed, 08 Sep 2021 18:02:24 GMT`
	`content-type: application/json`
	`content-length: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

HTTP-triggered function

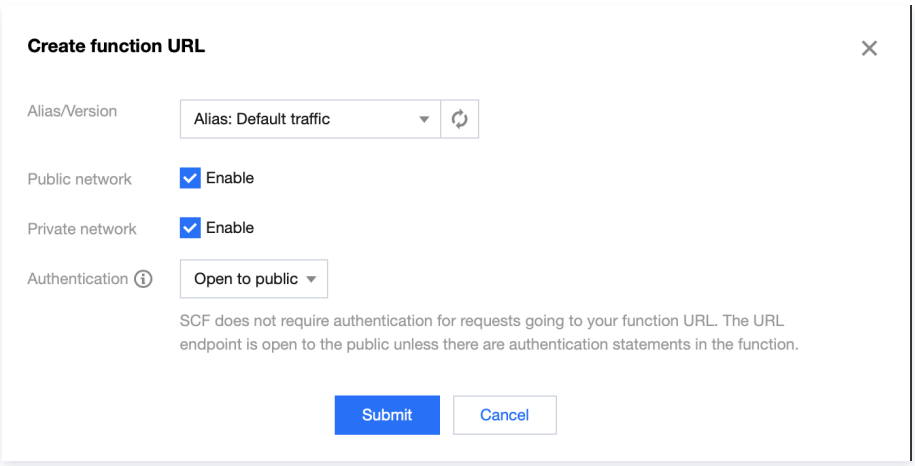
Upon receiving an HTTP request, the URL triggers the function to run. At this point, the URL directly forwards the HTTP request without any event type format conversion, and the request response is also directly forwarded.

Instructions

1. Log in to the Serverless console and click on Function Service in the left navigation bar.
2. On the **Function Service** page, click on the function name to access its detailed information.
3. Select **Function URL** from the left navigation and click on **Create Function URL**, as shown below:



4. On the **Create Function URL** page, create a new function according to the following information, as shown below:



Configuration items	Description
Alias/Version	The URL is bound to an alias or version dimension, and only one URL is permitted to be created for each alias or version.
Public/Private Network Access	Depending on your business needs, you can choose to enable public or private URL access.
Permission Type	The permission type supports the selection of **Open** and **CAM Authentication**. • **Open**: There is no need for identity verification for function requests, anonymous access is supported, and anyone can initiate an HTTP request to invoke your function. • **CAM Authentication**: Function CAM authentication verification is required. Users can manage resources and configure usage permissions based on the InvokeFunctionUrl interface. For more details, refer to the URL Authentication and Authorization documentation.

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

Function URL Authentication Configuration

Last updated: 2024-03-25 10:01:52

Feature Overview

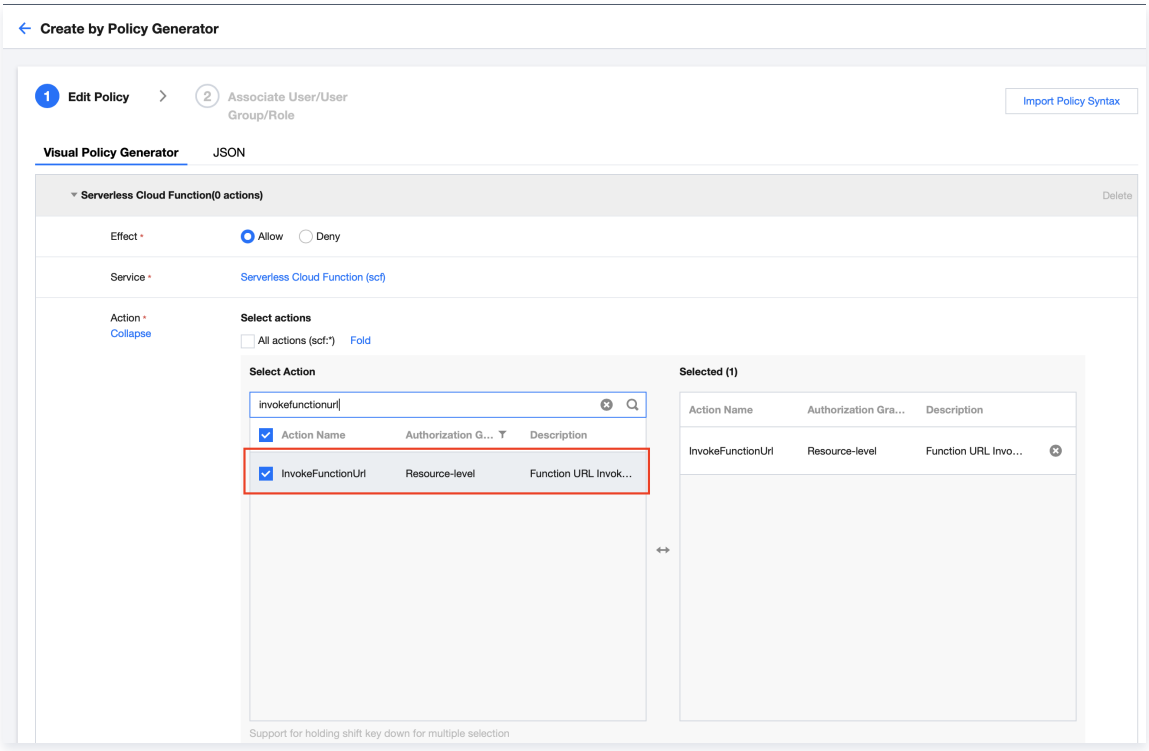
You can control access to function URLs by configuring authentication and authorization policies.
When configuring function URLs, one of the following authentication options must be specified:

- **CAM Authentication**: This requires CAM authentication verification for the function. Users can manage resources and configure usage permissions based on the InvokeFunctionUrl interface. You can open or restrict access to the interface by configuring the InvokeFunctionUrl policy permissions.
- **Open**: There is no need for identity verification for function requests, anonymous access is supported, and anyone can initiate an HTTP request to invoke your function.

Configuring InvokeFunctionUrl Policy Permissions

You can follow the steps below to configure InvokeFunctionUrl policy permissions to open or restrict access to the interface.

1. On the Policies page of the CAM console, click **Create Custom Policy** in the upper left corner.
2. In the pop-up window for selecting the creation method, click **Create by Policy Generator** to enter the policy editing page.
3. In the **Visual Policy Generator**, add a service and operation bar, supplement the following information, and edit an authorization statement.
   - Effect (Required): Select **Allow**.
   - Service (Required): Select Serverless Cloud Function (SCF).
   - Action (Required): Click **Expand** on the right side of All Actions (scf:*), search for InvokeFunctionUrl, and check it. As shown below:



   - Resource (Required): Select all resources or the specific resources you wish to authorize.
   - Condition (Optional): Set the conditions under which the above authorization will take effect.
4. Upon completion of the policy authorization statement, click **Next** to proceed to the basic information and associated user/user group/role page.
5. On the Associate User/User Group/Role page, supplement the policy name and description information. You can also quickly authorize by associating users/user groups/roles simultaneously.
6. Click **Complete** to finish creating a custom policy using the policy generator.

Signature Generation and Authentication Process

Client-side Signature Generation

Please refer to the Security Credential Service Signature Method for the signature algorithm. The code sample is as follows:

Java

```
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;
    // Set the environment variable TENCENTCLOUD_SECRET_ID, with the value as the example
AKIDz8krbsJ5yKBZQpn74WFkmLPx3***
    private final static String SECRET_ID = System.getenv("TENCENTCLOUD_SECRET_ID");
    // Set the environment variable TENCENTCLOUD_SECRET_KEY, with the value as the example
Gu5t9xGARNpq86cd98joQYCN3***
    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 algorithm = "TC3-HMAC-SHA256";
        String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        // Be mindful of the time zone to avoid errors
        sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
        String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

        // *** Step 1: Concatenate the Canonical Request String ***
        String httpRequestMethod = "POST";
        String canonicalUri = "/";
        String canonicalQueryString = "";
        String canonicalHeaders = "content-type:application/json\n"
                + "host:" + host + "\n" ;
        String signedHeaders = "content-type;host";

        // Request body
        String payload = "{\"Limit\": 1, \"Filters\": [{\"Values\": [\"\\u672a\\u547d\\u540d\"], \"Name\": \"instance-name\"}]}";
        String hashedRequestPayload = sha256Hex(payload);
```

```
        String canonicalRequest = httpRequestMethod + "\n" + canonicalUri + "\n" + canonicalQueryString + "\n"
                + canonicalHeaders + "\n" + signedHeaders + "\n" + hashedRequestPayload;
        System.out.println(canonicalRequest);

        // *** Step 2: Concatenate the String to be Signed ***
        String credentialScope = date + "/" + service + "/" + "tc3_request";
        String hashedCanonicalRequest = sha256Hex(canonicalRequest);
        String stringToSign = algorithm + "\n" + timestamp + "\n" + credentialScope + "\n" + hashedCanonicalRequest;
        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, stringToSign)).toLowerCase();
        System.out.println(signature);

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


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

Go

```
package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "os"
    "strings"
    "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() {
```

```
    // Set the environment variable TENCENTCLOUD_SECRET_ID, with the value as the example
AKIDz8krbsJ5yKBZQpn74WFkmLPx3***
    secretId := os.Getenv("TENCENTCLOUD_SECRET_ID")
    // Set the environment variable TENCENTCLOUD_SECRET_KEY, with the value as the example
Gu5t9xGARNpq86cd98joQYCN3***
    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" ] , "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 POST https : / / % s - H "Authorization: %s" - H "Content-Type: application/json; charset=utf-8"
- H "Host: %s" - H "X-Scf-Cam-Uin: %d" - H "X-Scf-Cam-Timestamp: %d" - d '%s',
        host, authorization, host, uin, timestamp, payload)
    fmt.Println(curl)
}
```

NodeJS

```
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
    // Set the environment variable TENCENTCLOUD_SECRET_ID, with the value as the example
AKIDz8krbsJ5yKBZQpn74WFkmLPx3***
    const SECRET_ID = process.env.TENCENTCLOUD_SECRET_ID
    // Set the environment variable TENCENTCLOUD_SECRET_KEY, with the value as the example
Gu5t9xGARNpq86cd98joQYCN3***
    const SECRET_KEY = process.env.TENCENTCLOUD_SECRET_KEY

    const endpoint = "1253970226-xxxxxxx-cq.scf.tencentcs.com"
    const service = "scf"
    const timestamp = getTime()
    // Time processing, obtaining Universal Time Coordinated (UTC) date
    const date = getDate(timestamp)

    // *** Step 1: Concatenate the Canonical Request String ***
    const payload = "{\"Limit\": 1, \"Filters\": [{\"Values\": [\"\\u672a\\u547d\\u540d\"], \"Name\": \"instance-name\"}]}"

    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 the String 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 the 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()
```

Python

```
# -*- 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 as the example
AKIDz8krbsJ5yKBZQpn74WFkmLPx3***
secret_id = os.environ.get("TENCENTCLOUD_SECRET_ID")
# The environment variable TENCENTCLOUD_SECRET_KEY needs to be set, with the value as the example
Gu5t9xGARNpq86cd98joQYCN3***
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 the Canonical Request String ***
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 the String 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 ***
# Function to Calculate Signature Digest
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).hexdigest()
print(signature)

# *** Step 4: Concatenate the 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 + "'")
```

Client Invoke Parameters

The following parameters need to be added to the header of the URL request:

Category	Description
Authorization	Mandatory signature-related parameters. Example: `TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WFkmLPx3***/2019-02-25/scf/tc3_request, SignedHeaders=content-type;host;xxx, Signature=be4f67d323c78ab9acb7395e43c0dbcf822a9cfac32fea2449a7bc7726b770a3`
X-Scf-Cam-Timestamp	The timestamp used to generate the signature, which is mandatory.
X-Scf-Cam-Uin	The primary account Uin, which is mandatory.
X-Scf-Cam-Token	If a temporary key is used to generate the signature, token information must be provided.

Server-Side Signature Verification

For server-side calls to the CAM service's signature and authentication, please refer to Access Management.

Version Management
Overview

Last updated: 2023-09-27 18:14:51

Overview

The version of Serverless Cloud Function (SCF) encompasses both the function's code and configuration. During the actual development process, you can solidify the function's code and configuration content by publishing versions, thereby minimizing factors that could potentially disrupt the business system.

Concepts

Most Recent/Latest Version ($LATEST)

Upon creation, a function inherently possesses a most recent/latest version ($LATEST), and only the configuration and code of the $LATEST version can be modified. When publishing, the configuration and code of the $LATEST version serve as the foundation for the release, generating a new version.

Relevant Operations

The operations that a version can perform include:

- View Version
- Publish Version
- Utilize Version

The following video will guide you on how to view, publish, and utilize versions:
Watch video

Viewing A Version

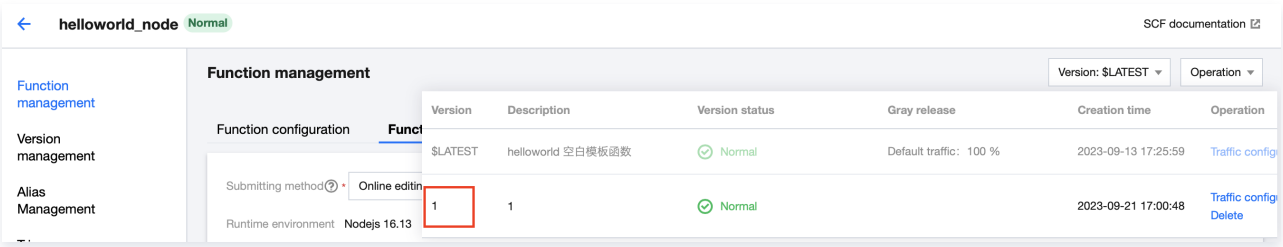
Last updated: 2023-09-27 18:15:03

Operational Overview

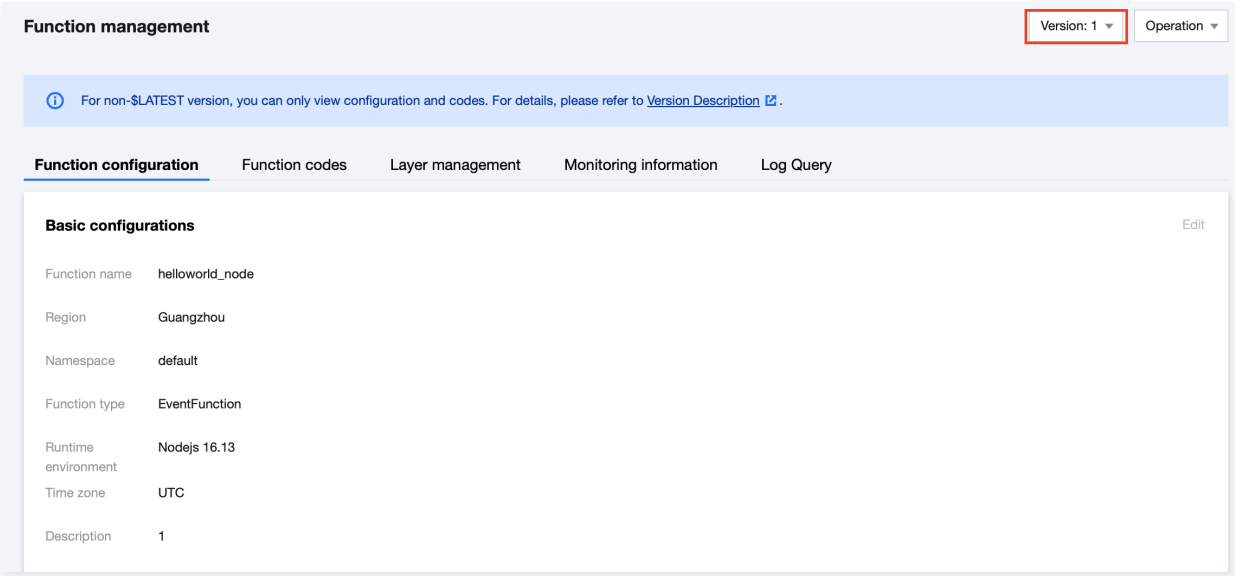
When you need to review the version configuration, code, and other information of a specific function, this document can serve as your guide.

Procedure

1. Log in to the Serverless console and select Function Service from the left navigation bar.
2. At the top of the "Function Service" list page, select the region and namespace where the function you want to view is located.
3. Click a function name to enter the function details page.
4. Select the "Version" dropdown list in the upper right corner of the function details page, and click on the name of the version you want to view. This document uses version `1` as an example, as shown below:



You can then view the relevant information for this version, as shown below:



Note

- After switching to a version, the **Function Configuration**, **Function Code**, **Layer Management**, **Monitoring Information**, and **Log Query** tabs will display the content corresponding to that version. For detailed information on each tab, please refer to Query Function.
- After switching to a version other than `$LATEST`, the function configuration and code remain in the published state and cannot be modified.
- Triggers can be configured differently on various versions.
- The logs and monitoring data respectively display the specific invocation logs and monitoring data for the corresponding version.

Releasing A Version

Last updated: 2023-09-28 15:19:03

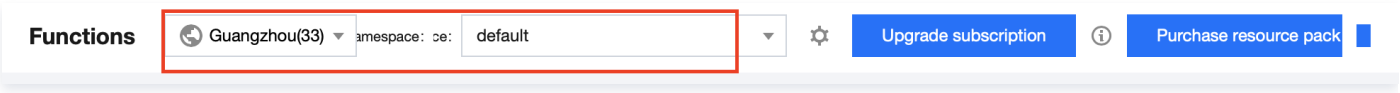
Operational Overview

Upon completing the configuration of your cloud function, submitting the code, and passing online tests, you can solidify the version of your cloud function by publishing a version. This helps to prevent online business errors or execution failures caused by subsequent code modifications and testing. You can publish a version at any time, and any version of the cloud function will publish the `$LATEST` version as the latest version.

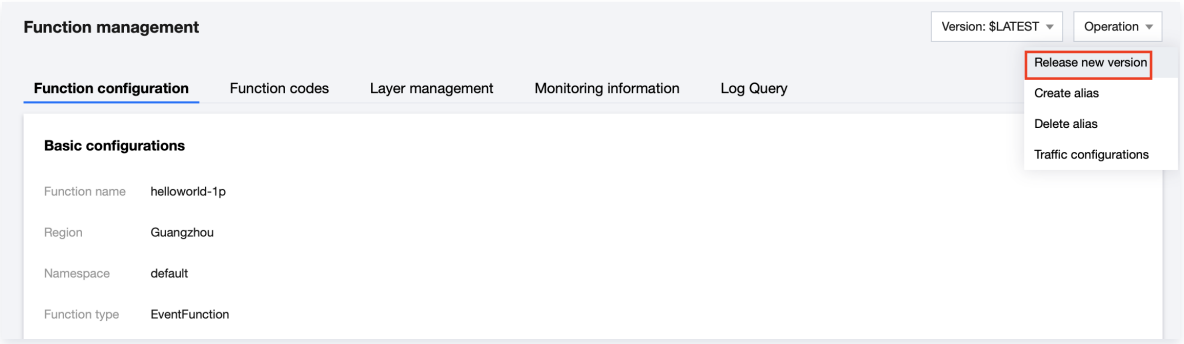
Procedure

> **Explanation**
> Upon creation, the cloud function possesses the attribute of the `$LATEST` version. The `$LATEST` version points to the currently editable version and always maintains its existence and editable status.

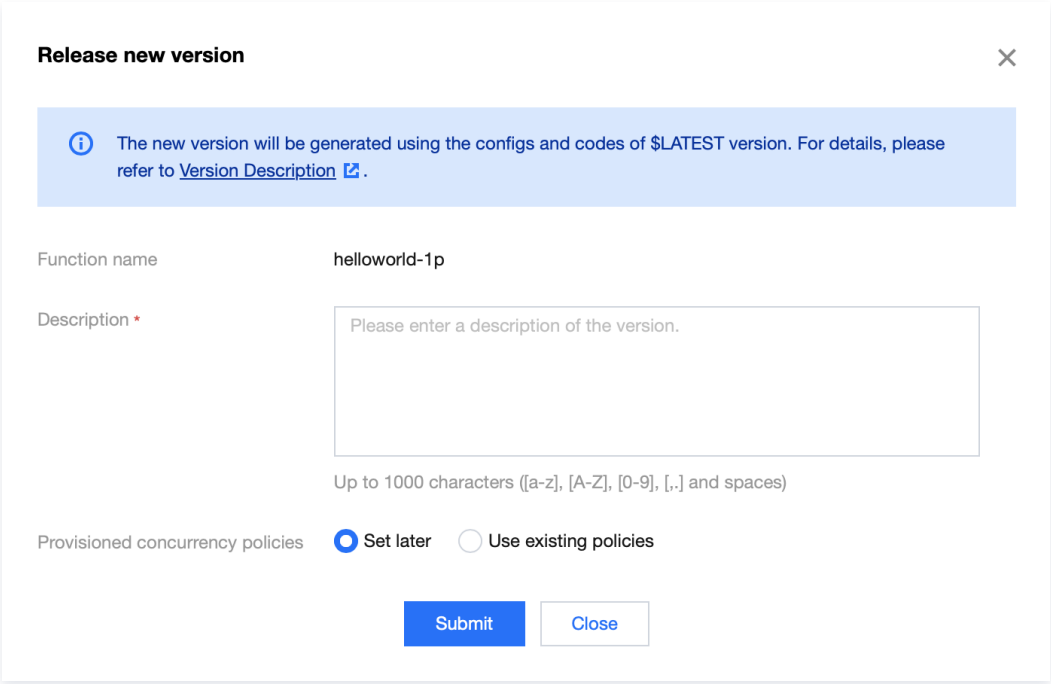
1. Log in to the Serverless console and click on Function Service in the left navigation bar.
2. At the top of the "Function Service" list page, select the region and namespace where the function to be viewed is located, as shown in the following figure:



3. Click on the function name to enter the function information page.
4. Select **Operation** > **Release new version** in the upper right corner of the function information page, as shown in the following figure:



5. In the pop-up "**Release new version**" window, enter the version description and click **Submit** to publish, as shown in the following figure:



Note

- Upon submission for publication, the cloud function platform will create a duplicate of the current function's `$LATEST` version configuration and code content, and save it as version content.
- Upon completion of the release, a version number for the current release will be generated. The version number starts at 1 and increases with each release, with no upper limit for the current version number.
- The published version only records and solidifies the configuration and code of the current function's `$LATEST` version, and does not record the function's trigger configuration. The newly published function version does not have any triggers.

Using A Version

Last updated: 2023-09-27 18:15:30

The versioning feature is primarily used to solidify function configurations and code, preventing any impact on operations during development, debugging, and testing. After publishing a version of the cloud function, you can utilize this version by invoking the specified cloud function.

> **Description**
> The `$LATEST` version is designated for development and testing, facilitating further code development and debugging.

Trigger of the version

Currently, each published version of a cloud function can independently bind with a trigger. For the same function, each version operates independently, and each trigger independently initiates the function execution.

> **Description**
> There is a certain limit to the number of triggers under a user account. For details, please refer to Quota Limits. If you need to increase the quota of triggers (i.e., quota increase), you can apply by submitting a ticket.

TencentCloud API Triggered Version

When triggering a cloud function call using the TencentCloud API InvokeFunction interface, you can specify the exact version to be triggered through the optional parameter `Qualifier`. If this parameter is not provided, the `$DEFAULT` alias will be triggered by default. For more details, please refer to Alias Management.

Alias Management
Related Operations For Alias Management

Last updated: 2023-09-28 09:33:12

Overview

An alias in Serverless Cloud Function (SCF) is a pointer that directs to a bound function version. By using an alias, you can invoke the bound function. In practical development, aliases can assist you in managing project version updates and rollbacks more effectively. A single function version can have one or more aliases. For more information on function version management, please refer to the Overview of Version Management.

Use case

Through the configuration of aliases, you can create distinctions for multiple different environments (stages) for a function. For instance:

- By creating 'test' and 'release' aliases and configuring triggers to point to these aliases, different codes and configurations can be activated.
- You can use aliases to bind different function versions. Once a version has been validated in the test environment, you can redirect the traffic of the production environment to the new version through route configuration. For the method of traffic route configuration, please refer to Traffic Routing Configuration Instructions.

Default Alias

Upon creation, a function has a default alias ($DEFAULT) that points to the most recent version ($LATEST). The default alias cannot be deleted or renamed, but it does support traffic routing configuration.

Usage of Default Alias

When configuring triggers and invoking functions through the Cloud API, it is recommended to set the `Qualifier` parameter during the call to the default alias ($DEFAULT).

> **Instructions**
> By configuring the default alias, you can control the routing of default traffic generated by triggers and Cloud API calls.

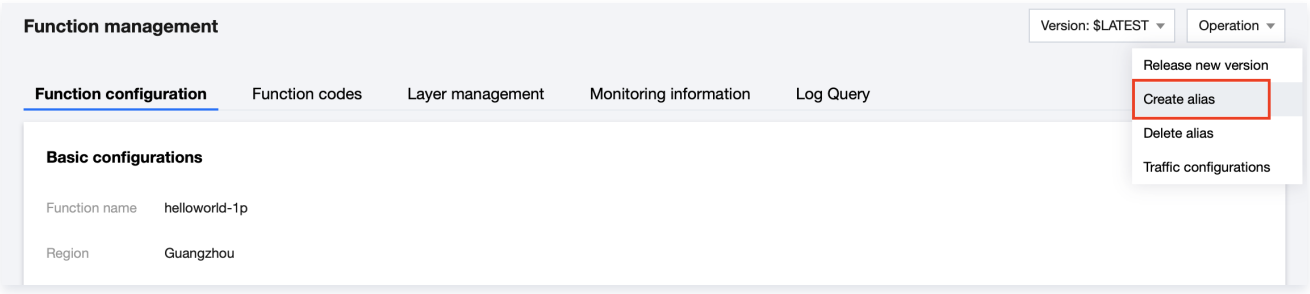
Triggering an Alias

Currently, all aliases created for cloud functions can independently bind triggers. The invocation of the trigger will pass through the alias and pull up the specific version to execute based on the routing configuration of the alias.

Procedure

Create Alias

1. Log in to the Serverless console and click on Function Service in the left navigation bar.
2. On the "Function Service" list page, click on the function name to enter the details page of that function.
3. Select **Operation** > **Create Alias** in the upper right corner of the page, as shown in the figure below:



4. In the "Create Alias" pop-up window that appears, refer to the following information to create. As shown in the figure below:

> **Note**
> Once the alias is created, the alias name cannot be modified.

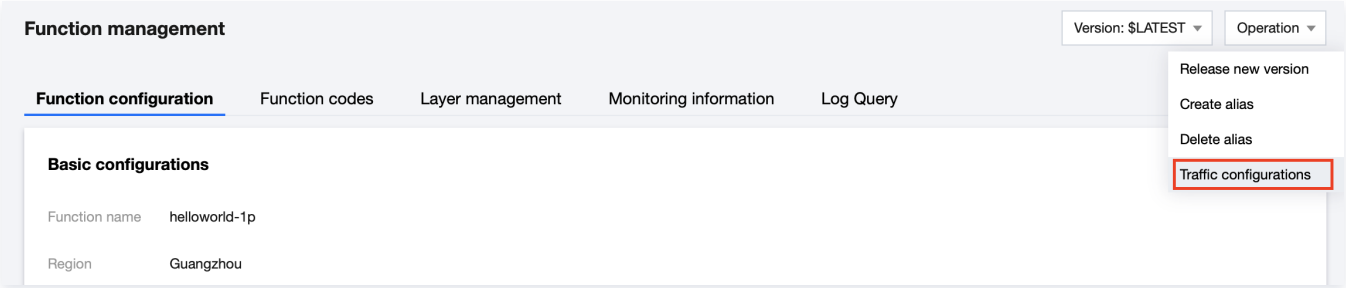
The main parameters are described as follows:

- **Alias Name**: Custom name. It should be between 2 and 60 characters long, start with a letter, and can include `a - z`, `A - Z`, `0 - 9`, `-`, `_`. It must end with a digit or a letter, for example, `Tencent-cloud_scf`.
- **Alias Description**: Custom description. It can be up to 1000 characters long and may include English letters, numbers, spaces, commas, periods, and Chinese characters.
- **Route Method and Version Weight Configuration**: For more details, refer to Traffic Routing Configuration.

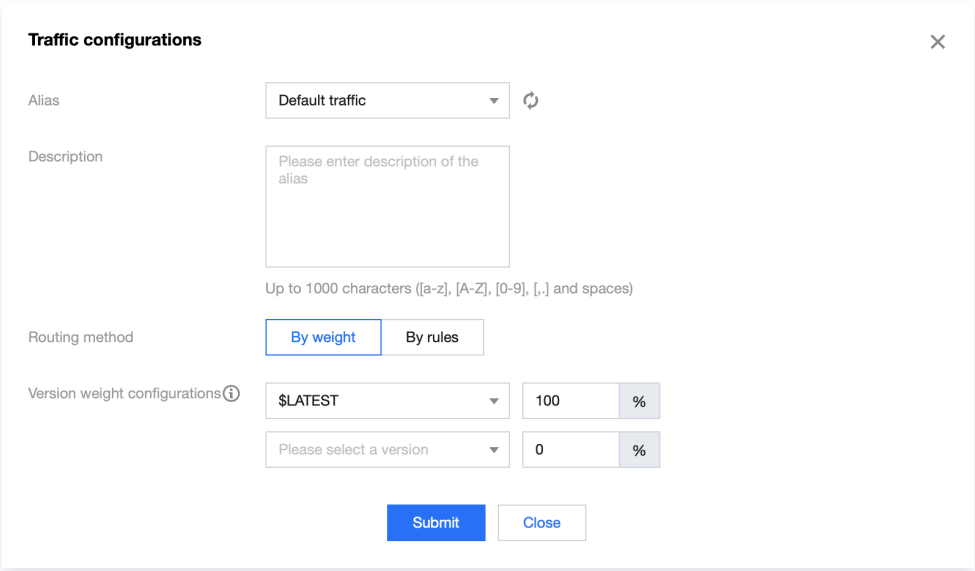
5. Click **Submit** to complete the creation.

Modify the alias bound to the function version.

1. Select **Operation** > **Traffic Settings** in the upper right corner of the function details page. As shown below:



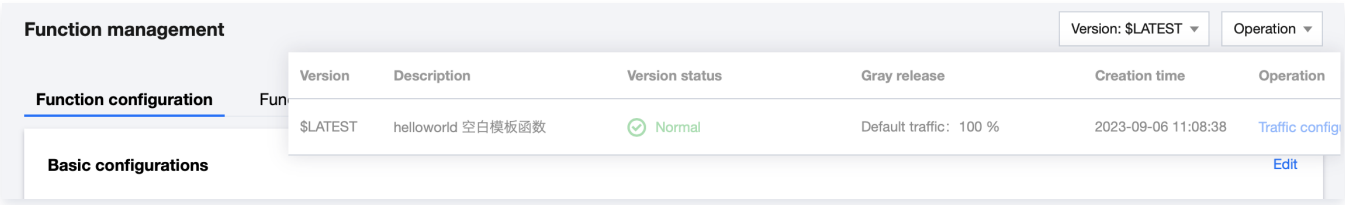
2. In the pop-up "Traffic Settings" window, configure as per the instructions below. As shown in the figure:



The main parameters are as follows:

- **Alias**: From the dropdown list, select the alias you wish to bind to this version. In this document, `test02` is used as an example.
- **Route Method and Version Weight Configuration**: For configuration details, please refer to Traffic Routing Configuration. This document uses the modification of the alias bound to the `$LATEST` version as an example:
  - Select the routing method as **Weighted Routing**.
  - The version weight configuration is as follows: The weighted routing for version `$LATEST` is 70%, and the weighted routing for version `1` is 30%.

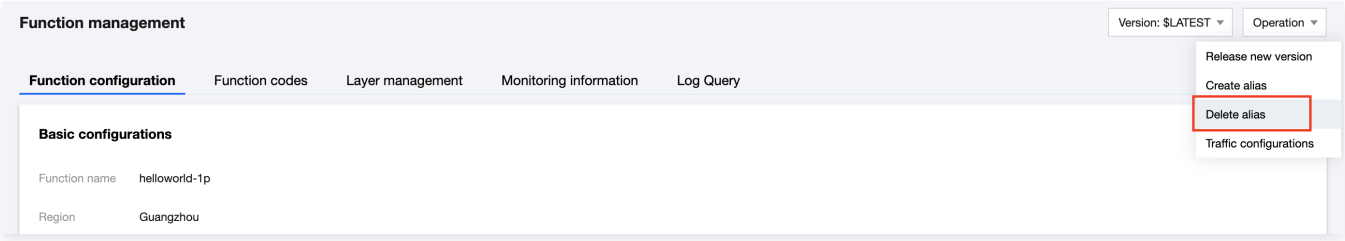
3. Click **Submit** to complete the changes. Open the version dropdown list to view the effect of the modifications. As shown in the figure:



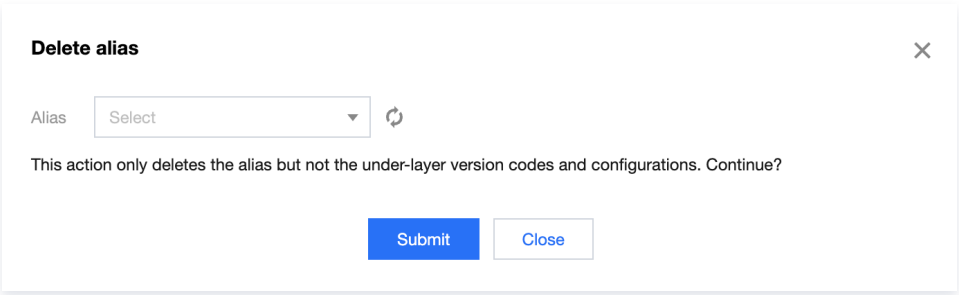
Deleting alias

> **Note**
> This action only permanently deletes the alias, excluding the underlying version codes and configurations.

1. Select **Operation** > **Delete Alias** in the upper right corner of the function details page. As shown in the figure:



2. In the "Delete Alias" pop-up window, select the alias you wish to delete from the dropdown list and click **Submit**. This document uses the deletion of alias `test02` as an example. As shown in the figure:



Traffic Routing Configuration

Last updated: 2023-09-28 09:38:44

Operational Overview

Serverless Cloud Function (SCF) supports traffic routing settings. With this configuration, you can conveniently control the gray release or rollback process of function versions in actual use cases or environments, mitigating the potential risks of a one-time launch.
When creating an alias or adjusting traffic configurations, you can control the traffic direction to two function versions through the console, enabling traffic to route between versions according to certain rules. Currently, we support two routing schemes: **Weighted Random Routing** and **Rule-based Routing**.

- When you wish to randomly route between any two versions based on a set percentage weight, you can perform the Weighted Random Routing operation.
- When you wish to route requests containing specific content to a particular version, you can perform the Rule-based Routing operation.

Procedure

Weighted Random Routing

This article takes the configuration during alias creation as an example. Once creation is complete, traffic will randomly route between the two versions according to the set percentage. The steps are as follows:

1. Refer to the Create Alias steps to access the "Create alias" window.
2. In the "Create alias" window, refer to the following information for traffic routing configuration, as illustrated in the figure below:

The main parameter information is as follows:

- **Routing method**: Select **Weighted Routing**.
- **Version weight configurations**: You can select two versions from the drop-down list and configure the percentage weight.

3. Click **Submit** to complete the setup.

By rules

When configuring routing by rules, the current rule syntax includes the following three parts:

Matching Key

The position of the value during matching, that is, determining whether it is a hit by locating and obtaining the value. The currently supported notation for Key is `invoke.headers.[userKey]`, where the `[userKey]` part represents modifiable content. This notation means that the userKey part in the HTTP request headers is matched when the `invoke` interface is called.

Matching Method

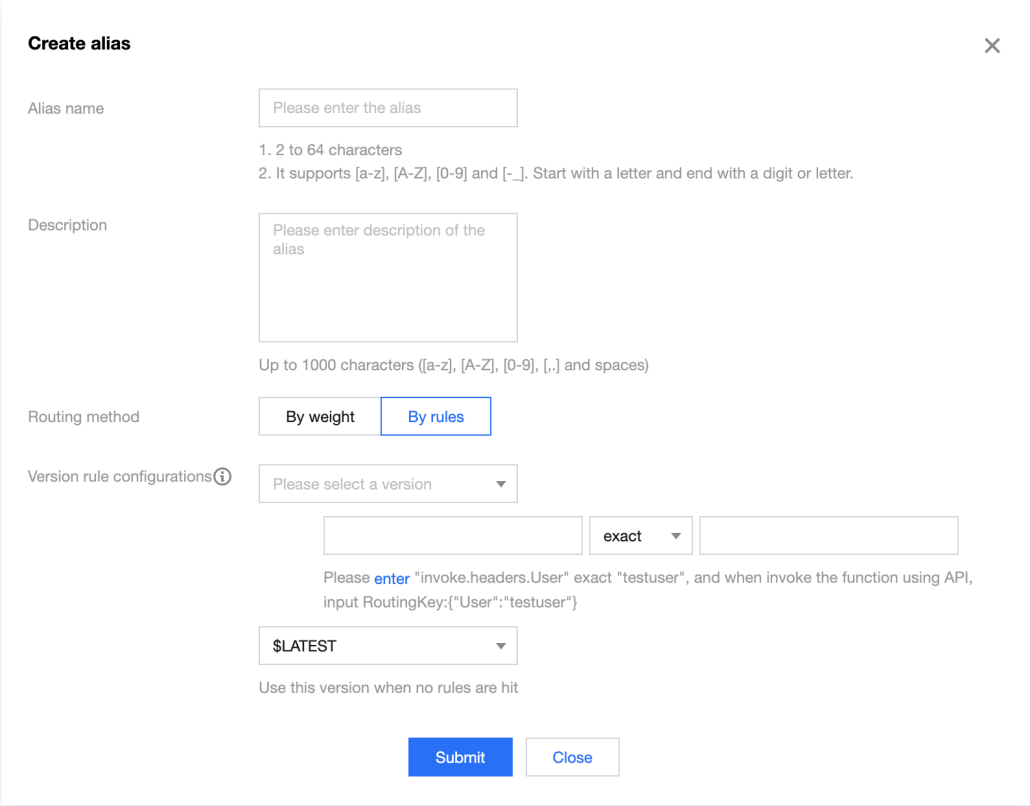
During matching, the method and expression are compared. The currently supported matching methods are `exact` and `range`.

- `exact`: Precise match. When using the `exact` method, the matching expression must be a string. The rule is hit when the value read by the matching key is exactly equal to the expression.
- `range`: Range match. When using the `range` method, the matching expression must be in the form of (a,b) or [a,b], where a and b must be integers. The rule is hit when the value read by the matching key is an integer and falls within the interval defined by the expression.

Matching Expression

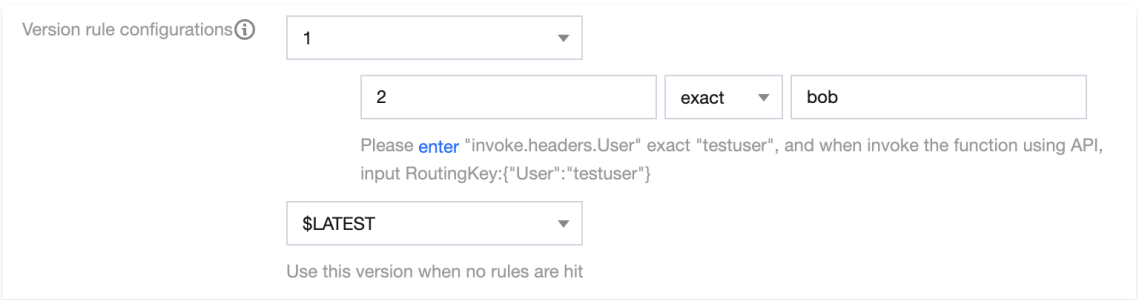
A hit occurs when the set value is matched. For the expression syntax, refer to the Matching Method description.
Please follow the steps below to configure routing based on rules:

1. Refer to the Create Alias steps to access the "Create alias" window.
2. In the "Create alias" window, refer to the following information to configure traffic routing, and click **Submit** to complete the configuration. See the figure below:



The main parameter information is as follows:

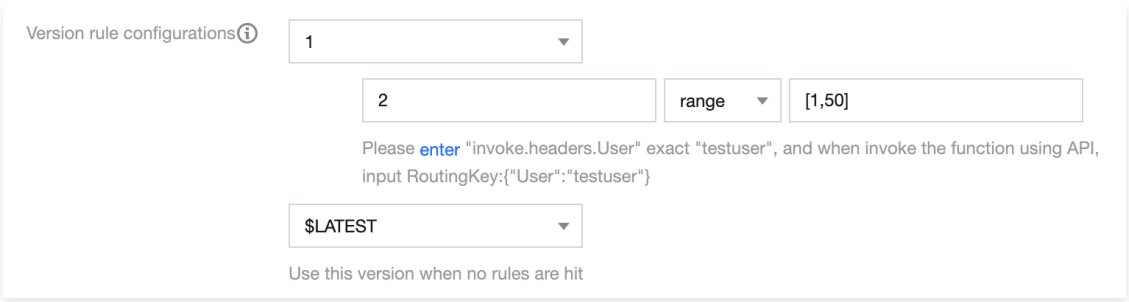
- **Routing method**: Select **Rule-based Routing**.
- **Version rule configurations**: Please configure as needed, using the following examples as a guide:
  - For instance, if you have two versions (Version 2 and Version 1), and you want Version 2's matching rule to be set as `invoke.headers.User exact Bob`, and Version 1 to be set for no hits. Refer to the figure below for settings:



Based on this configuration, when the cloud function platform calls the function alias through the `invoke` interface, if

the `routingKey` parameter is set to `{"User":"Bob"}`, the execution will use the code and configuration of Version 2. If the `routingKey` parameter is not set or if the `routingKey` is set to other values, the execution will use the code and configuration of Version 1.

- For instance, if you have two versions (Version 3 and Version 2), and you want Version 3's matching rule to be set as `invoke.headers.userHash range [1,50]`, and Version 2 to be set for no hits. Refer to the figure below for settings:



Based on this configuration, when the cloud function platform calls the function alias through the `invoke` interface, if the `routingKey` parameter is set to `{"userHash":30}`, the execution will use the code and configuration of Version 3. If the `routingKey` parameter is not set or if the `routingKey` is set to values other than [1,50], for example, `{"userHash":80}`, the execution will use the code and configuration of Version 2.

Permission Management
Permission Management Overview

Last updated: 2023-09-27 18:18:21

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	Cloud API	Console	Authorization Granularity	Temporary Certificate
Serverless	✔	✔	✔	Resource	✔

The current SCF supports the following cloud API interfaces:

API Name	Description	Level
ListFunctions	Gets the function list under the account	Account
GetAccount	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: 2023-09-28 10:43:47

Concepts

Roles

A Role is a virtual identity provided by Tencent Cloud's Cloud Access Management (CAM) that possesses a set of permissions. Roles can also be granted policies, primarily used to authorize Role Carriers with access to services, operations, and resources in Tencent Cloud. Once these permissions are attached to a role, the role can be assigned to Tencent Cloud services, allowing the services to perform operations on authorized resources on behalf of the user. The roles of Tencent Cloud Function SCF are divided into **Configuration Roles** and **Execution Roles**. You can use the Configuration Role to allow SCF to access user resources during the service configuration process. Alternatively, you can use the Execution Role to request temporary authorization for running code, facilitating code to achieve permission penetration and resource access through the role's authorization mechanism.

Policies

A Policy is a syntax rule that defines and describes one or more permissions. CAM supports two types of policies: preset policies and custom policies. Preset policies are a collection of common permissions created and managed by Tencent Cloud, such as super administrators, cloud resource administrators, etc., and these policies are read-only. Custom policies are a more detailed collection of permissions for resource management created by users. Preset policies cannot specifically describe a resource and are relatively coarse-grained, while custom policies can flexibly meet the differentiated permission management needs of users.

Permissions

Permissions describe the conditions under which certain operations are allowed or denied access to certain resources. By default, the root account is the owner of the resources and has full access to all resources under its name. A sub-account does not have access to any resources. The creator of a resource does not automatically have access to the resources they create; authorization is required from the resource owner.

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:

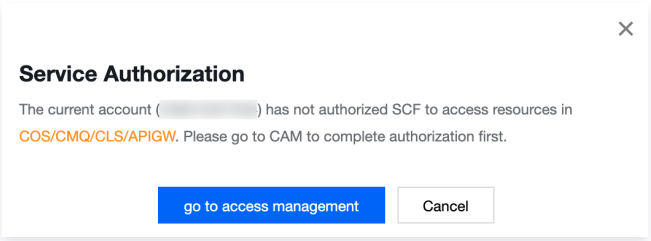
- Role name: `SCF_QcsRole`
- Role entity: `service-scf.qcloud.com`
- Role description: SCF default configuration role. This service role is used to grant the SCF configuration the permissions to connect with other resources in the cloud, including but not limited to code file access and trigger configuration. The preset policy of the configuration role can support 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 read/write access.

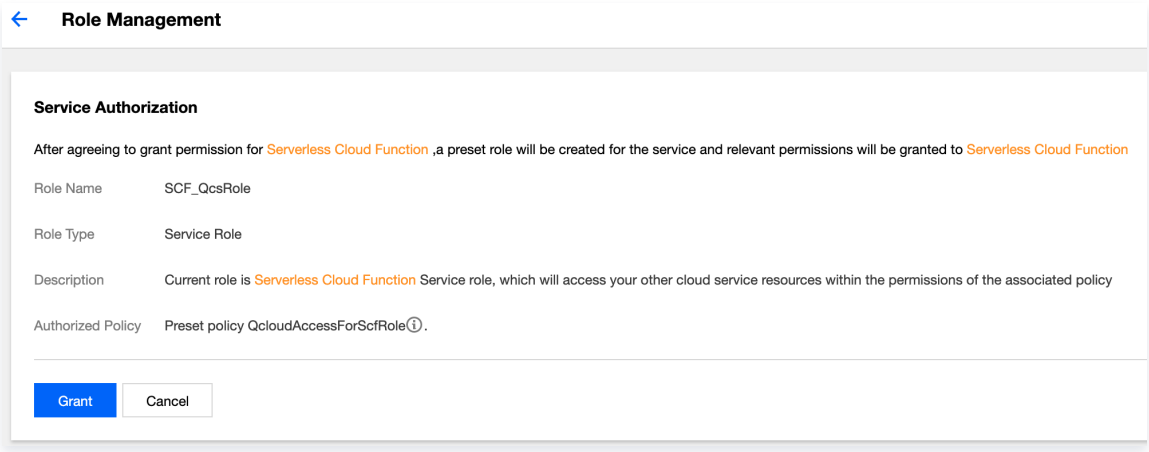
> **Note**
> Users can view and modify the policies associated with the current configuration role `SCF_QcsRole` in the CAM Console. However, modifying the role's associated policies may cause issues such as SCF not functioning properly, so it is not recommended.

Service Authorization

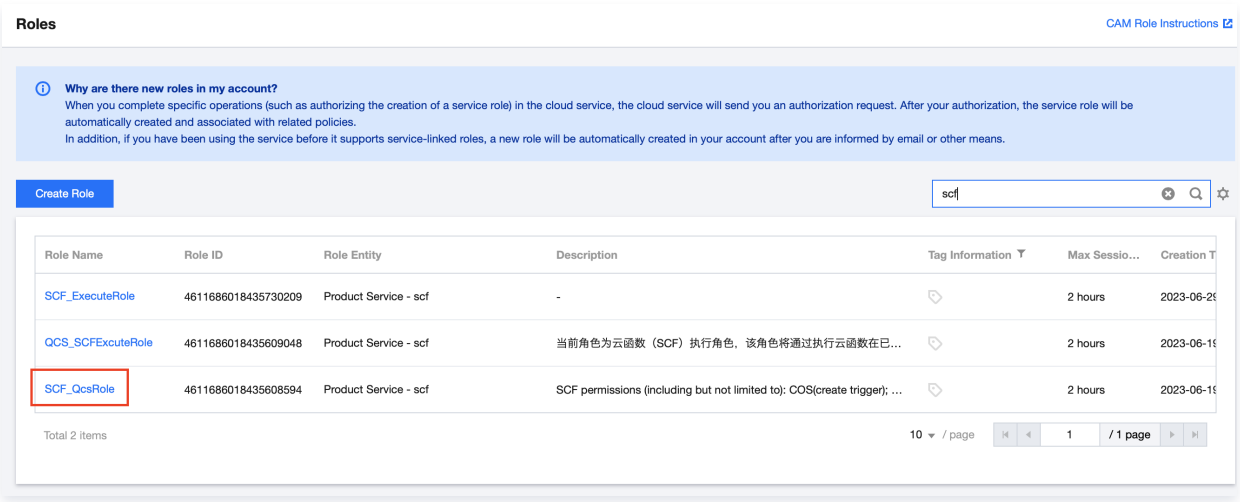
1. If you are using SCF for the first time, you will be prompted to authorize the service when you open the Serverless Console, as shown below:



2. Select **go to access management** to enter the "Role Management" page, and click **Grant** to confirm the authorization, as shown below:



3. Upon confirmation of authorization, the role `SCF_QcsRole` will be automatically created for you. You can view it under Roles, as shown below:

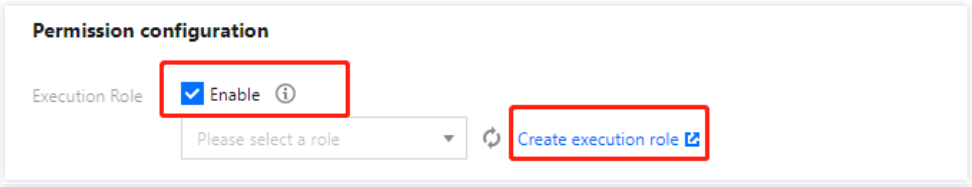


Execution Roles

The execution role serves the user's code, with the role entity being `product service-scf.qcloud.com`. After users add the corresponding execution role to the function, SCF applies for temporary authorization for the user's running code within the permission scope of the associated policy of the execution role, facilitating the code to achieve permission penetration and access to other cloud resources through the role's authorization mechanism.
Taking `SCF_QcsRole` as an example, users can also choose `SCF_QcsRole` as the function's execution role, which means granting the permissions corresponding to the associated policy of `SCF_QcsRole` to SCF, enabling SCF to obtain the right to apply for access to other cloud resources for the user's code.

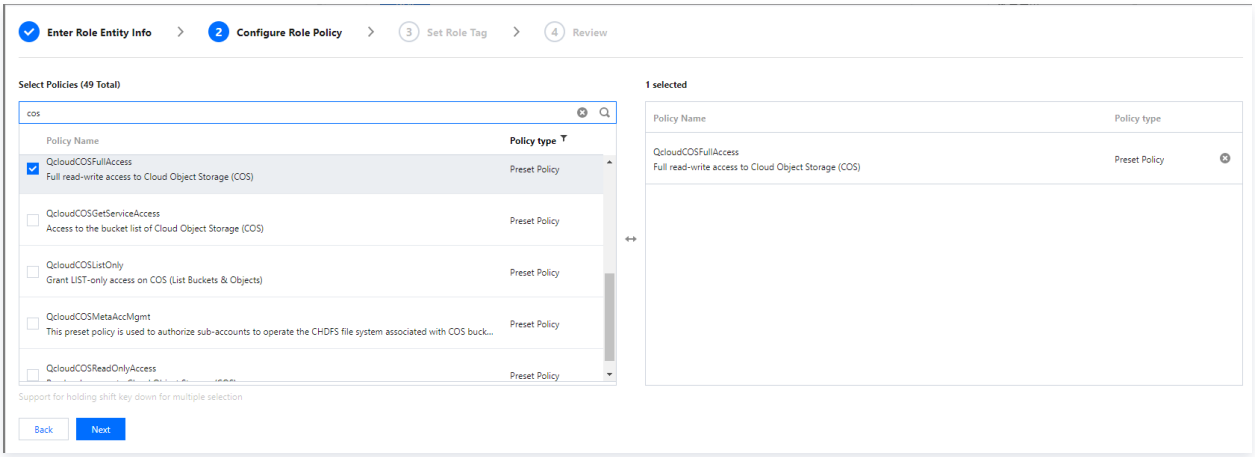
Creating execution roles

1. Log in to the Serverless Console and click on Function Service in the left navigation bar.
2. On the **Function Service** list page, click on the name of the function for which you need to create an execution role, and you will be directed to the function configuration page.
3. Select **Edit** at the top right corner of the function configuration page, check **Enable** under "Execution Role", and click on **Create execution role**, as shown in the figure below:

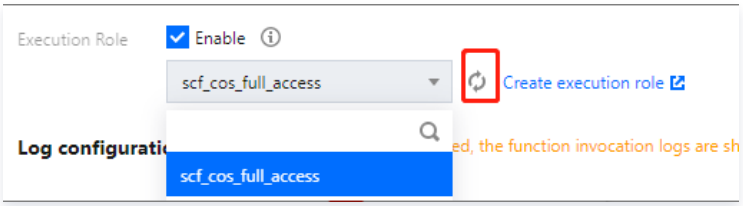


4. In the "Enter Role Entity Info" step, select **Cloud Function (scf)** and click **Next**.
5. In the "Configure Role Policy" step, select the policy required for the function and click **Next**, as shown in the figure below:

> **Note**
> This document uses the selection of `QcloudCOSFullAccess` (full access permissions of COS) as an example. Please select the policies as needed.



6. After configuring the role tags as needed, fill in the role name in the "Review" step and click **Complete**. This document uses `scf_cos_full_access` as an example for the role name.
7. Return to the function configuration page and click on the to the right of "Execution Role". You can then select the execution role you just created from the dropdown list, as shown in the figure 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's policy syntax follows CAM's syntax structure and resource description method, which is based on the JSON format. For more information, please 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 `SesstionToken`.

- For **functions not created from an image:**
  The relevant content will be passed into the runtime environment in the form of environment variables, as shown in the figure below:

```
TENCENTCLOUD_SECRETID: 'AKIDhOeFP[redacted]3Hm0ji8Qqov9xVj',
TENCENTCLOUD_SECRETKEY: 'r24xDBuIc[redacted]TxwkE=',
TENCENTCLOUD_SESSIONTOKEN: 'GJmIp[redacted]yEZE86V9c
```

  Taking Python as an example, you can pass the above information into the function runtime environment and obtain it as an environment variable using the following code.

```
secret_id = os.environ.get('TENCENTCLOUD_SECRETID')
secret_key = os.environ.get('TENCENTCLOUD_SECRETKEY')
token= os.environ.get('TENCENTCLOUD_SESSIONTOKEN')
```

- For **functions created from an image:**
  The relevant content will be passed into the `context` parameter in the form of HTTP headers. For more details, please refer to Image Function Parameter Description.

SCF Policy Syntax

Last updated: 2023-09-27 18:20:11

Policy Syntax

For more information on how to create custom policies, please see Creating Custom Policies. SCF's policy syntax follows CAM's syntax structure and resource description method, which is based on the JSON format, and all resources can be 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**
> When configuring the policy syntax, you also need to use the monitor APIs to get the monitoring information under the account. For more information about using the monitor APIs, please see the sample policy.

Policy Examples

```
{
    "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"
        ]
      }
    ]
}
```

- When the action requires associated resources, the resource is defined as `*`, indicating that all resources are associated.
- When the action does not require associated resources, the resource must 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.

Specified 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	Sample
ip_equal	IP equal to	qcs:ip	{"ip_equal":{"qcs:ip ":"10.121.2.0/24"}}
ip_not_equal	IP not equal to	qcs:ip	{"ip_not_equal":{"qcs:ip ":["10.121.1.0/24", "10.121.2.0/24"]}}
date_not_equal	Date is not equal to	qcs:current_time	{"date_not_equal":{"qcs:current_time":"2016-06-01T00:01:00Z"}}
date_greater_than	Date is greater than	qcs:current_time	{"date_greater_than":{"qcs:current_time":"2016-06-01T00:01:00Z"}}
date_greater_than_equal	Date is greater than or equal to	qcs:current_time	{"date_greater_than_equal": {"qcs:current_time":"2016-06-01T00:01:00Z"}}
date_less_than	Date is less than	qcs:current_time	{"date_less_than":{"qcs:current_time":"2016-06-01T 00:01:00Z"}}
date_less_than_equal	Date is less than or equal to	qcs:current_time	{"date_less_than":{"qcs:current_time":"2016-06-01T 00:01:00Z"}}
date_less_than_equal	Date is less 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"}
```

- Restrict access to IPs `101.226.***.185` and `101.226.***.186` as shown below:

```
"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",
            "ckafka:List*",
            "ckafka:Describe*",
            "ckafka: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*",
        "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"
      }
    ]
}
```

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 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.

Sub-users And Authorization

Last updated: 2023-09-27 18:20:27

> **Note**
> The primary account needs to check on the Roles page whether it has `SCF_QcsRole`. If not, please follow the Roles and Policies section's **Service Authorization** operation to complete the authorization. Otherwise, the sub-user will not be able to use the Serverless console and call other 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 Access Management Console and select **Users** > **User List** from the left sidebar.
2. On the **User List** page, select **Create User** > **Custom Creation** to navigate to the **Create Sub-User** page.
3. In the **Select Type** step, choose **Can Access Resources and Receive Messages**, then click **Next** to fill in the user information.
4. Follow the on-screen instructions to fill in and confirm the information, then click **Complete** to finish the custom creation of the sub-user.

> **Note**
> For more information, please see Creating a Custom Sub-user.

Step 2. Create a custom policy

1. On the Policies page of the CAM console, click **Create Custom Policy** in the upper left corner.
2. In the pop-up window for selecting the creation method, click **Create by Policy Generator** to enter the policy editing page.
3. In the "Visual Policy Generator" page for selecting services, supplement the following information to edit an authorization statement.
   - **Effect**: allow
   - **Service**: SCF
   - **Action**: all
   - **Resource description**: `*`
   - **Condition (optional)**: empty
4. Upon completion of the policy authorization statement, click **Next** to proceed to the basic information and associated user/user group/role page.
5. On the Associate User/User Group/Role page, supplement the policy name and description information. You can simultaneously associate users/user groups/roles for quick authorization.
6. Click **Complete** to finish creating a custom policy using the policy generator.

Step 3: Grant CAM read-only permissions to the sub-user

1. Log in to the Cloud Access Management Console and navigate to the User List management page.
2. On the User List management page, select the sub-user for whom you need to set permissions.
3. Click **Authorize** in the operation column on the right.
4. In the pop-up policy association window, check the `QcloudCamReadOnlyAccess` policy.
5. Click **OK** to complete the authorization of "User and Permissions (CAM) Read-Only Access" for the sub-user.

Completion

Upon completion of the above settings, users can log in to the sub-account to view permissions. Log in to the Access Management Console, select Overview from the navigation bar on the left to enter the Overview page, where you can 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 Access Management Console and select **Users** > **User List** from the left sidebar.
2. On the **User List** page, select **Create User** > **Custom Creation** to navigate to the **Create Sub-User** page.
3. In the **Select Type** step, choose **Can Access Resources and Receive Messages**, then click **Next** to fill in the user information.
4. Follow the on-screen instructions to fill in and confirm the information, then click **Complete** to finish the custom creation of the sub-user.

> **Note**
> For more information, please see Creating a Custom Sub-user.

Step 2. Create a custom policy

1. On the Policies page of the CAM console, click **Create Custom Policy** in the upper left corner.
2. In the pop-up window for selecting the creation method, click **Create by Policy Generator** to enter the **Edit Policy** page.
3. Copy the policy example code from SCF Policy Syntax, and modify 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 function name under the root account, and the "region" should be consistent with the function.

4. Click **Next** to proceed to the Basic Information and Associated User/User Group/Role page.
5. On the **Associate User/User Group/Role** page, supplement the policy name and description information, and you can quickly authorize by associating users/user groups/roles simultaneously.
6. Click **Complete** to finish creating a custom policy using the policy generator.

Step 3: Grant CAM read-only permissions to the sub-user

1. Log in to the Cloud Access Management Console and navigate to the User List management page.
2. On the User List management page, select the sub-user for whom you need to set permissions.
3. Click **Authorize** in the operation column on the right.
4. In the pop-up policy association window, check the `QcloudCamReadOnlyAccess` policy.
5. Click **OK** to complete the authorization of "User and Permissions (CAM) Read-Only Access" for the sub-user.

Completion

Upon completion of the above settings, users can log in to the sub-account to view permissions. Click Overview in the left navigation bar to enter the overview page, where you can 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`.

Managing Monitors And Alarms
Descriptions Of Monitoring Metrics

Last updated: 2023-09-28 15:36:11

The Tencent Cloud Observability Platform 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 average execution duration of a function at the function/region level, indicating the time from the start to the end of the user's function code execution, calculated with a specified granularity (1 minute, 5 minutes).	Milliseconds	Function Region
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, summed by granularity (1 minute or 5 minutes).	-	Function Region
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
Public traffic out	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	http_433	The number of 433 status codes returned after function execution, summed by granularity (1 minute or 5 minutes).	-	Function

timed out (HTTP 433)				
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

Description
To **obtain the required monitoring data**, you can visit the Cloud Function Monitoring Interface.

Configuring Alarm

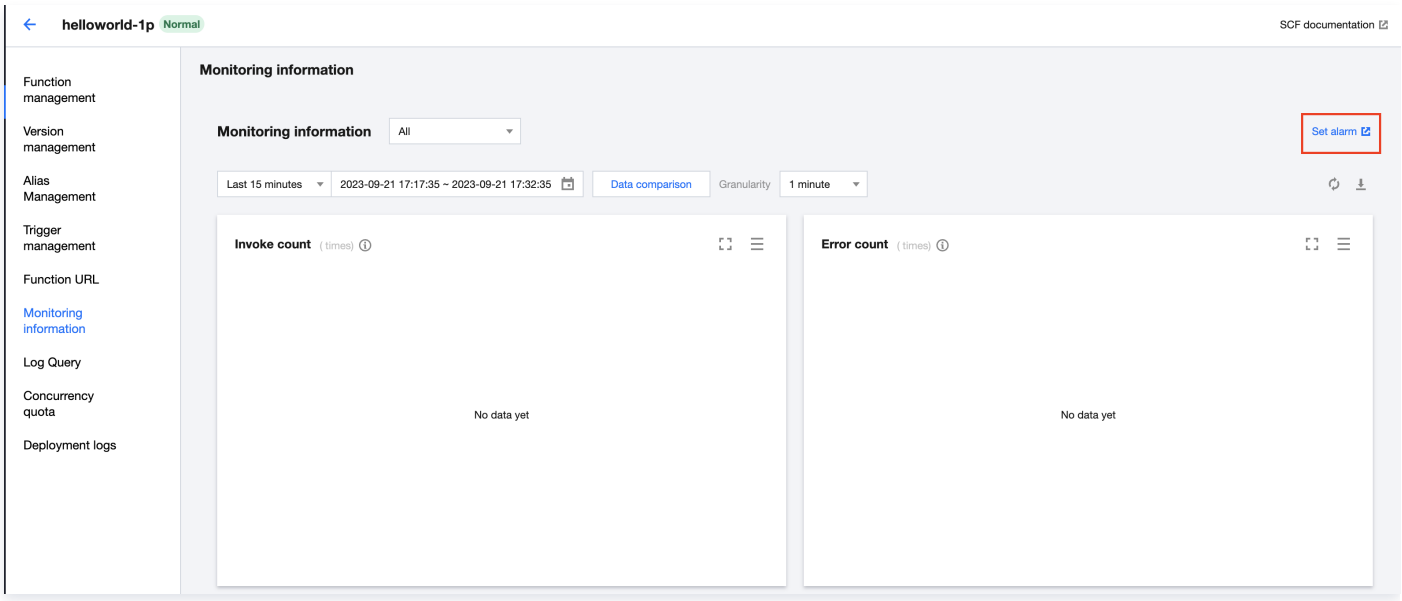
Last updated: 2023-09-28 10:50:54

Operational Overview

You can configure alarm policies for cloud functions through the Tencent Cloud Observability Platform, allowing you to monitor the operational status of the functions. Currently, the monitoring metrics for which alarms can be configured for cloud functions include runtime, number of calls, number of errors, etc. For a complete list of supported metrics, please refer to Monitoring Metrics Description. Additionally, the alarm system allows you to select user groups to receive alarms and choose from various notification channels such as email, SMS, and WeChat.

Procedure

1. Log in to the Serverless console and select **Function Service** from the left navigation bar.
2. On the **Function Service** list page, click on the function name to enter the function details page.
3. Select **Monitoring information** from the left navigation and click **Set alarm** on the Monitoring Information Details page, as shown in the figure below:



4. On the **Create Alarm Policy** page, configure the alarm policy as follows:
   - **Policy name**: Custom.
   - **Monitoring Type**: Select "Cloud Product Monitoring".
   - **Policy Type**: Supports selection of "Cloud Function/Version" or "Cloud Function/Alias".
   - **Alarm Object**: Set according to actual needs. If "Instance ID" is selected, its default region is set to Guangzhou. In different regions, you can see the corresponding functions, please select the functions that need to apply the alarm policy.
     For more detailed alarm policy configuration, please refer to Create Alarm Policy.
5. Click **Complete**. In **Alarm Management** > Policy Management, you can view the configured policy and choose to start or stop it at any time based on actual needs.

Tutorial Video

The following video will guide you on how to configure alarms and view logs:
Watch video

Viewing Execution Logs

Last updated: 2023-09-28 10:53:16

Operational Overview

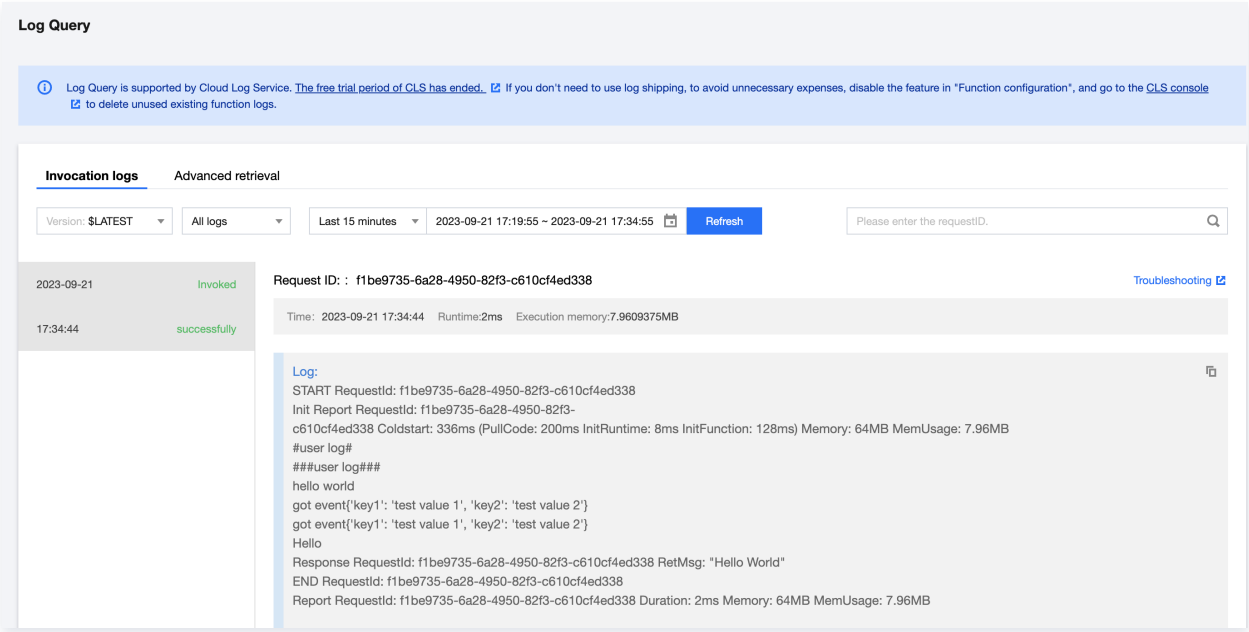
You can view the function execution logs in the SCF console. You can filter to display real-time logs or logs from the last 24 hours. You can also specify a custom time range. You can choose to view all logs, or logs of successful, failed, timed-out, and excessive invocations and code exceptions in the SCF console.

Procedure

You can view function logs in the SCF console.

Viewing execution logs in console

1. Log in to the Serverless console and click on Function Service in 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** interface for the function, as shown below:

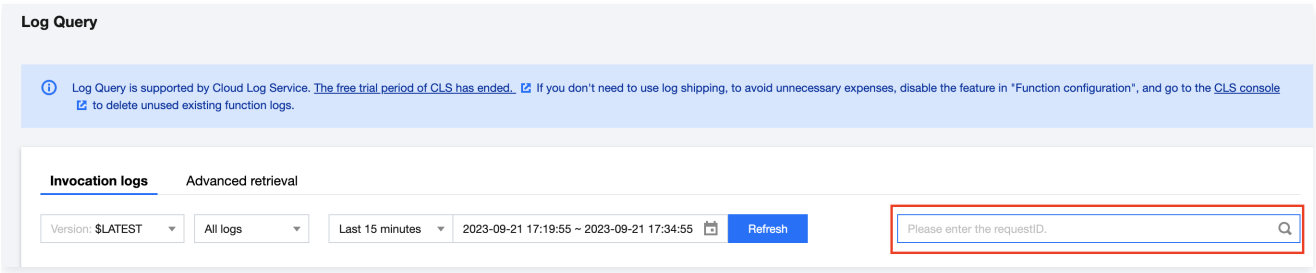


Finding execution log

Follow the steps below to find an execution log as needed.

Invocation log

In the search box at the top right, enter the requestID of the execution log you want to view and press Enter to view a specific execution log, as shown below:



You can also set custom search criteria in the top-left corner based on actual needs.

- **All Logs**: you can select successful or failed invocations logs.
- **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.

- **Real-time**: 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: 2023-09-27 18:40:47

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 and Static Public Network Egress IP	The cloud function can access public network resources using a fixed IP address.
Only private network access is enabled	Upon configuring this setting, the cloud function can access resources such as CVM, Light Application Server (LH), databases, and Redis that are configured in this private network.
Enable both private and public network access	Upon configuring this setting, the cloud function can access public network resources as well as resources such as CVM, Light Application Server (LH), databases, and Redis that are configured in this private network.
Enable private network access, public network access, and static public network egress IP simultaneously.	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.

Explanation
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 Serverless console and click on **Function Service** in the left navigation bar.
2. Choose a region at the top of the page and click the function to be configured.
3. On the **Function Configuration** page, select **Edit** in the upper right corner.
4. Based on your actual needs, refer to Fixed Public Network Exit IP and Private Network Communication for configuration.

Viewing network configuration

1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
2. Select a region at the top of the page and click on the function name to view the current network configuration in **Function Configuration**.

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.

Bandwidth cap

Public network

When configuring public network access and requiring a fixed public network egress IP, the public network bandwidth (out) is limited to 0.1 Gbps.

Private network

When configuring a VPC for private network access, the current bandwidth limit (outbound + inbound) for a specific VPC is 1.5 Gbps. This bandwidth is shared by functions configured with the same VPC and multiple concurrent instances of the function. If you need to increase the private network bandwidth, please submit a ticket to apply.

Fixed Public Outbound IP

Last updated: 2023-09-27 18:41:31

Scenario

When users need to access databases, WeChat Official Account API interfaces, or other third-party services within a cloud function, they can utilize the fixed public network egress IP feature of the cloud function to achieve control and management of the cloud function's 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.

Usage 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 the fixed public outbound IP feature is enabled at the same time, the function will be assigned a randomly allocated EIP. Functions under the same VPC subnet will share this fixed outbound IP when the fixed outbound IP feature is enabled.

Sample

To facilitate your understanding of the usage restrictions of the fixed public outbound IP, here is a simple illustrative example for you.
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

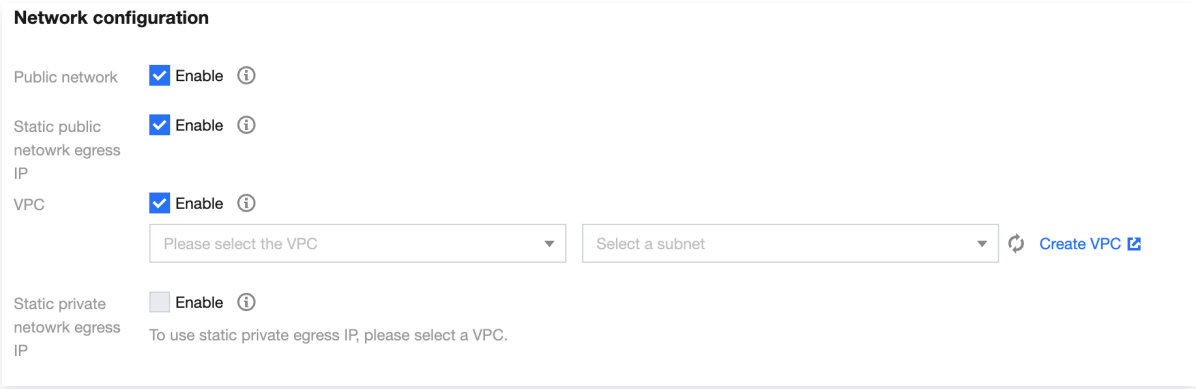
Instructions

⚠ Note

> Each user is limited to five fixed IP addresses per region.

1. Log in to the Serverless console and click on Function Service in the left navigation bar.
2. Select the function region at the top of the page and click the function name.
3. Navigate to the **Function Configuration** tab and click **Edit** in the upper 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.



After configuring, click **Save**.

VPC Communication

Last updated: 2023-09-27 18:41:52

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.

Precautions

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:
  - To configure the public network access capability for SCF and ensure that the outbound address is unique, see Static Public Network Egress IP.
  - To add a NAT gateway through a VPC, see Configuring NAT in a Private Network.
- Currently, functions cannot be connected with resources on the classic network.

Prerequisites

You have created a cloud function.

Procedure

Modifying network configuration

1. Log in to the Serverless console and click on Function Service in 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 upper right corner.
4. Enable the **Private Network** feature and select the VPC network and subnet you wish to connect to.

Using VPC

Upon completion of the private network access configuration for the SCF and the initiation of VPC network usage, the SCF will transition from its current independent network environment to the configured VPC. When the SCF is launched, it will occupy an IP address in the user's VPC subnet as the IP address for the SCF runtime environment.
Once the SCF is launched, resources in the VPC, such as TencentDB for Redis, Cloud Relational Database, and user-configured CVMs in the VPC, can be accessed via code and private IP addresses.
Below is an example code for accessing TencentDB for Redis, where the IP address of the Redis instance within 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:abcd1234")
    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 names in the VPC network (recommended)
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.

Setting the Name Server in the SCF Environment

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	Usage
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 Serverless console and click on **Function Service** in the left navigation bar.
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**.

Layer Management Overview

Last updated: 2023-09-27 18:42:30

Overview

If your Serverless Cloud Function (SCF) has numerous dependency libraries or public code files, you can manage them using layers in SCF. By utilizing layer management, you can place dependencies in layers instead of deployment packages, ensuring the deployment packages maintain a minimal size. For Node.js, Python, and PHP functions, as long as the deployment package is kept under 10MB, you can edit the function code online in the SCF console.

Mode of Operation

Creation and Binding

The compressed file created for a layer is stored according to the layer's version. When a layer is bound to a function, it is done so according to the specific layer version and function version. Currently, a function can bind up to five specific versions of a layer, and there is a certain sequence during binding.

Runtime Loading and Access

When a function bound to a layer is triggered to run and concurrent instances are launched, the function's runtime code will be decompressed and loaded into the **/var/user/** directory, while the layer content will be decompressed and loaded into the **/opt** directory.
If the file **file** that needs to be used or accessed is placed in the root directory of the compressed file when creating the layer, it can be directly accessed through the directory **/opt/file** after decompression and loading. If, when creating the layer, the file is compressed through a folder **dir/file**, then during function execution, the specific file must be accessed through **/opt/dir/file**.
In the case where a function is bound to multiple layers, the decompression and loading of files in the layers will be carried out in the order they were bound. They will be sorted in ascending order by number, with the layers bound later loading later, but all will be loaded before the function's concurrent instances are launched. The files in the layers can be used as soon as the function code is initialized.

Recommended Usage

Layers are typically used to store static files or code libraries that do not change frequently. When storing code libraries, you can directly package the available libraries and upload them to the layer. For example, in a Python environment, you can package the code library folder directly and create it as a layer, then directly reference it in the function code using import. In a Nodejs environment, you can package the project's node_modules library folder and create it as a layer, then directly reference it in the function code using require.
By using layers, you can separate the function code from the dependent libraries or static files, keeping the function code compact. Whether using command line tools, IDE plugins, or editing functions in the console, you can quickly upload updates.

Notes

- The files in the layer will be added to the `/opt` directory, which is accessible during the execution of the function.
- If your function is bound with multiple layers, these layers will be merged into the `/opt` directory in order. If the same file appears in multiple layers, the SCF platform will retain the file in the layer with the highest sequence number.

Relevant Operations

You can create layers, bind layers, and utilize layers through the Serverless console.

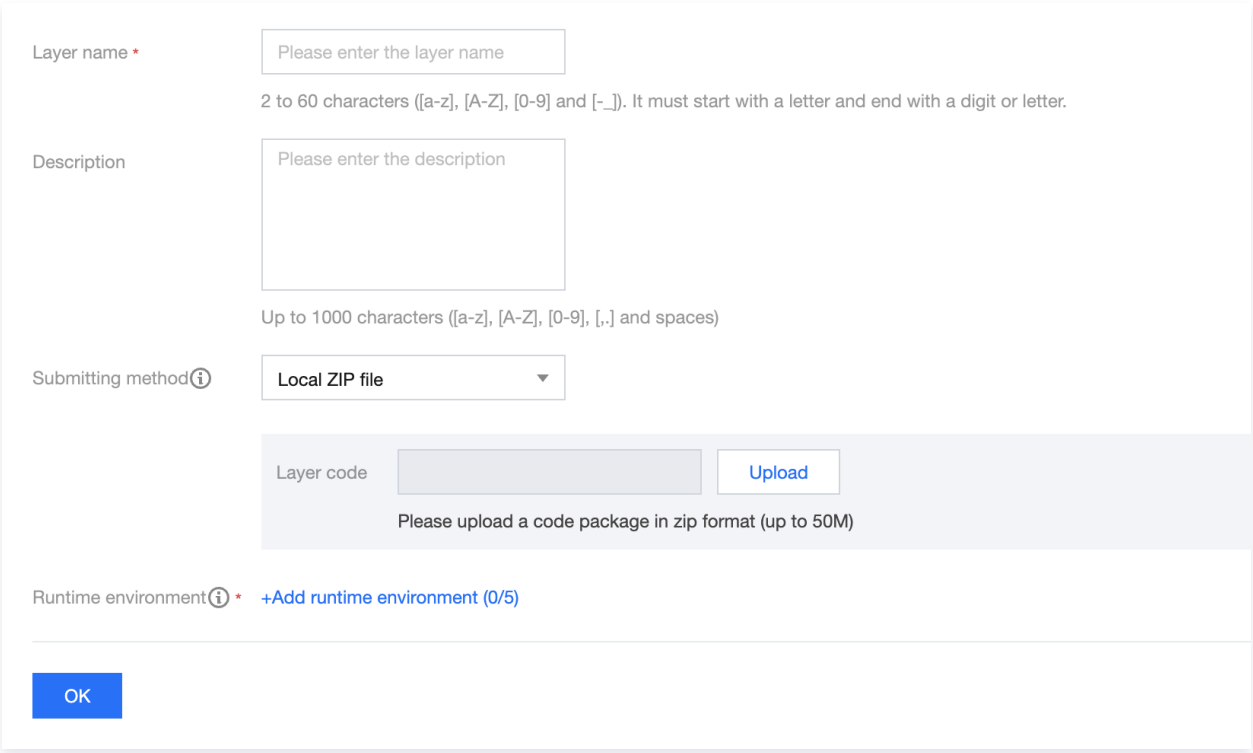
Creating Layer

Last updated: 2023-09-28 10:56:56

This document outlines how to create a layer via the Serverless console. Upon creation, a version will be automatically generated for you.

Procedure

1. Log in to the Serverless console, and select **Advanced Capabilities > Layer** from the left sidebar.
2. On the **Layer Management** page, select the region where you want to use the layer, and click **Create**.
3. On the **Create Layer** page, configure the layer information according to your actual needs, as shown in the following figure:



   - **Layer name**: enter a custom layer name.
   - **Description**: enter descriptive information of the layer as needed.
   - **Submission method**: **Local ZIP file**, **Local folder**, and **Upload a ZIP pack via COS** are supported. Please select an appropriate layer file submission method based on your actual needs.
     - **Upload Local ZIP File:** Click **Upload** and submit a code package in zip format, with a maximum size of 50MB.
     - **Upload Local Folder:** Click **Upload** and select a folder, with a maximum size of 250MB. This upload method does not retain the file's executable permissions. If the folder contains executable files, please set the executable permissions locally first and then upload them via a zip package.
     - **Upload ZIP File via COS:** Select the COS bucket to use as the event source, which must be located in the same region as the function. Enter the full path of the zip code file from the root directory ("/") of the Bucket.
   - **Runtime environment**: up to 5 runtime environments compatible with this layer can be set.
4. Click **OK**. You can view the created layer in the layer list.

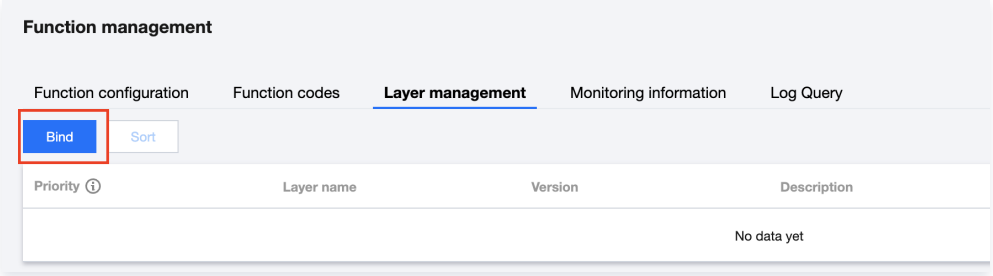
Binding Function To Layer

Last updated: 2023-09-28 10:59:20

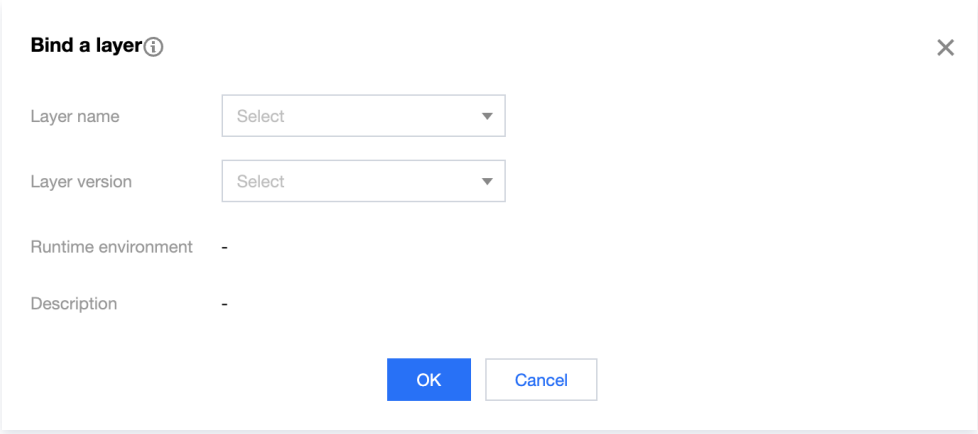
This document outlines how to bind a layer to a Serverless Cloud Function via the console.

Procedure

1. Log in to the Serverless console and select **Function Service** from the left-hand navigation bar.
2. On the "Function Service" page, select the Function ID that requires layer management to proceed to the function details page.
3. Select the **Layer management** tab and click **Bind**, as shown in the figure below:



4. In the pop-up "Bind a layer" window, select the corresponding **Layer name** and **Layer version**, as depicted in the figure below:



5. Click **OK** to complete the binding.

Using Layer

Last updated: 2023-09-28 11:12:41

This article outlines how to utilize layers via the Serverless console.

Use 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 Variables	Path
PYTHONPATH	/var/user:/opt
CLASSPATH	/var/runtime/java8:/var/runtime/java8/lib/*:/opt
NODE_PATH	/var/user:/var/user/node_modules:/var/lang/node6/lib/node_modules:/opt:/opt/node_modules

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. Refer to the Create Layer steps to upload `node_modules` and generate a layer. The local function directory structure is as shown below:

```
bash-3.2$ ls
index.js        node_modules    package.json
```

2. Refer to Deploy Function to package and upload the local function code. Execute the following command to exclude the `node_modules` folder during packaging.

```
zip -r package name.zip . -x "node_modules/*"
```

As depicted 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**
> - Since the NODE_PATH environment variable includes the `/opt/node_modules` path, there is no need to specify the absolute path of the dependency. The SCF runtime will load files according to the path specified in the environment variable.
> - 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. Refer to the Create Layer steps to upload and generate a layer for `cos-python-sdk-v5`.
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
```

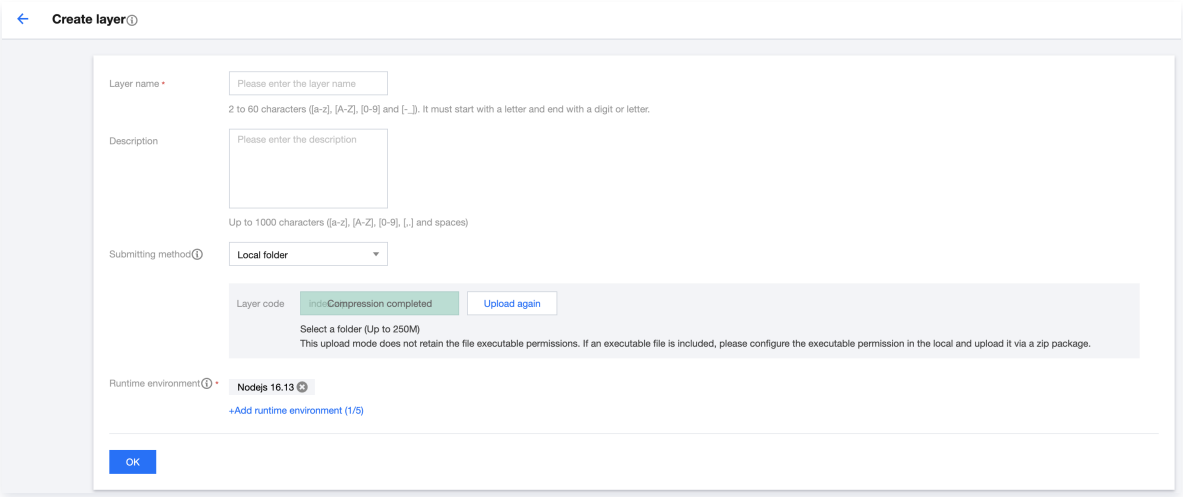
Note

- Since the PYTHONPATH environment variable includes the `/opt` path, there is no need to specify the absolute path of the dependency. The SCF runtime will load files according to the path specified in the environment variable.
- 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.

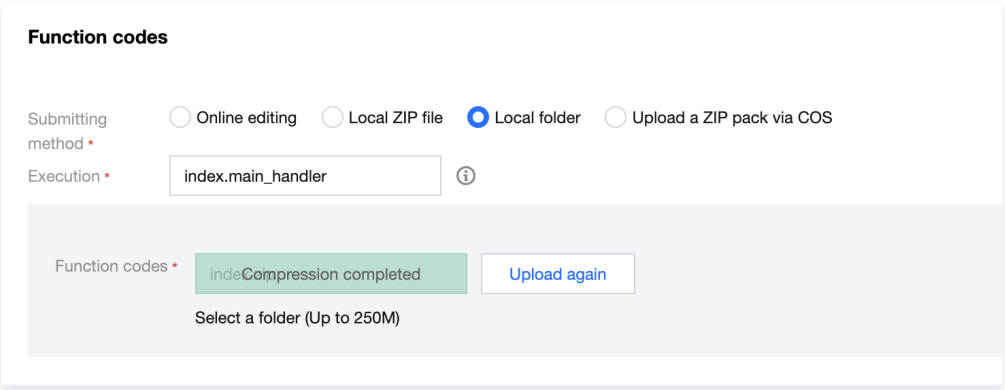
Sample Code

This example demonstrates how to use layers and test functions.

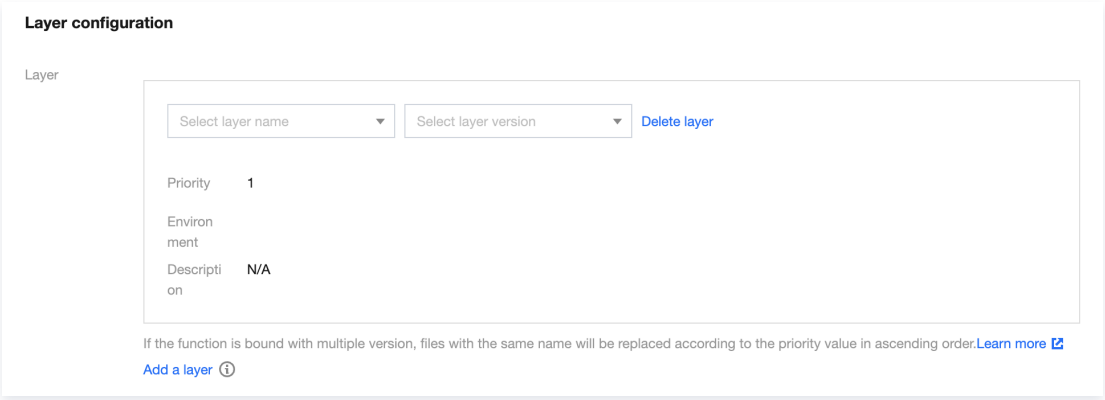
1. Navigate to scf_layer_demo, select **Clone or download** > **Download ZIP** to download the sample to your local machine and unzip it.
2. Log in to the Serverless Console and create a layer. For detailed operation steps, see Creating a Layer. Set the parameters as shown in the following figure:



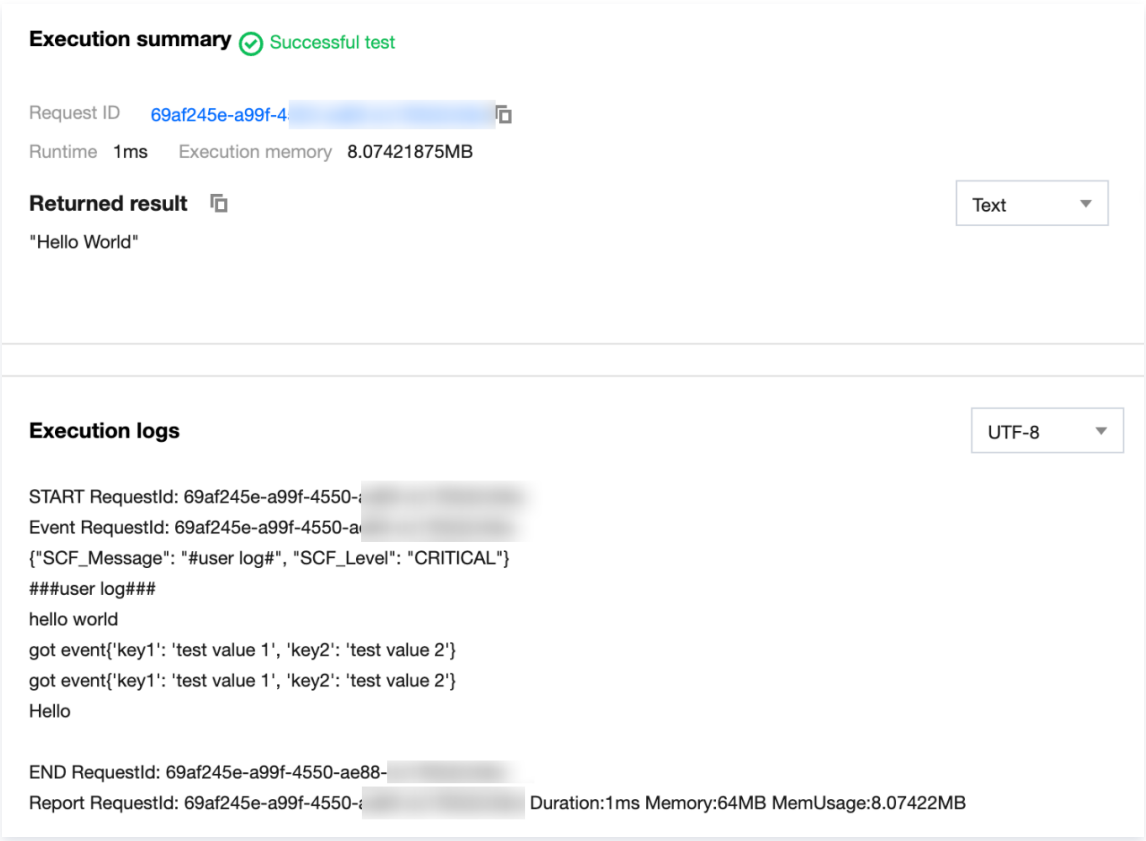
   - **Layer name**: enter a custom name. This document uses `demo` as an example.
   - **Submiting method**: Select "Upload local folder" and choose the `layer` folder from the directory obtained in Step 1.
   - **Runtime environment**: Select "Nodejs 12.16".
3. Log in to the Serverless Console and create a new function. For detailed operation steps, see Creating a Function. The basic configuration is as follows:
   - **Creation Method**: Select "From Scratch".
   - **Function Type**: Select **Event Function**.
   - **Function Name**: In this context, `layerDemo` is used as an example.
   - **Region**: The region is filled in by default.
   - **Runtime Environment**: Select "Nodejs 12.16".
   - **Time Zone**: The cloud function uses UTC time by default.
4. In the **Function codes** section, select "Local folder" and choose to upload the `function` folder from the folder obtained in Step 1. As shown in the image below:



5. In the **Advanced Settings > Layer configuration** section, click **Add a Layer**.
6. Choose the layer name and layer version for the function, as depicted in the figure below:



   - **Layer name**: Select the `demo` layer created in Step 2.
   - **Layer version**: select v1.
7. Click **Complete** at the bottom of the page. After creation, you can view the function details.
8. In "Function Management", select the **Function Code** tab and click **Test** at the bottom of the page to view the results, as depicted in the figure below:



Execution Configuration
Async Execution

Last updated: 2023-09-27 18:48:13

Use case

In scenarios such as audio and video transcoding, ETL large volume data processing, and AI inference where single tasks require heavy computation, the single instance of a function needs more computing power and longer stable operation time. If the function's caller is blocked for a long time waiting for execution results, it will not only continuously occupy the caller's resources, but also place higher demands on the stability of the call link.

Tencent Cloud's Serverless Cloud Function (SCF) offers a novel function execution mechanism. By utilizing the asynchronous execution mode provided by SCF, you can increase the execution timeout limit and address issues with the existing execution mechanism.

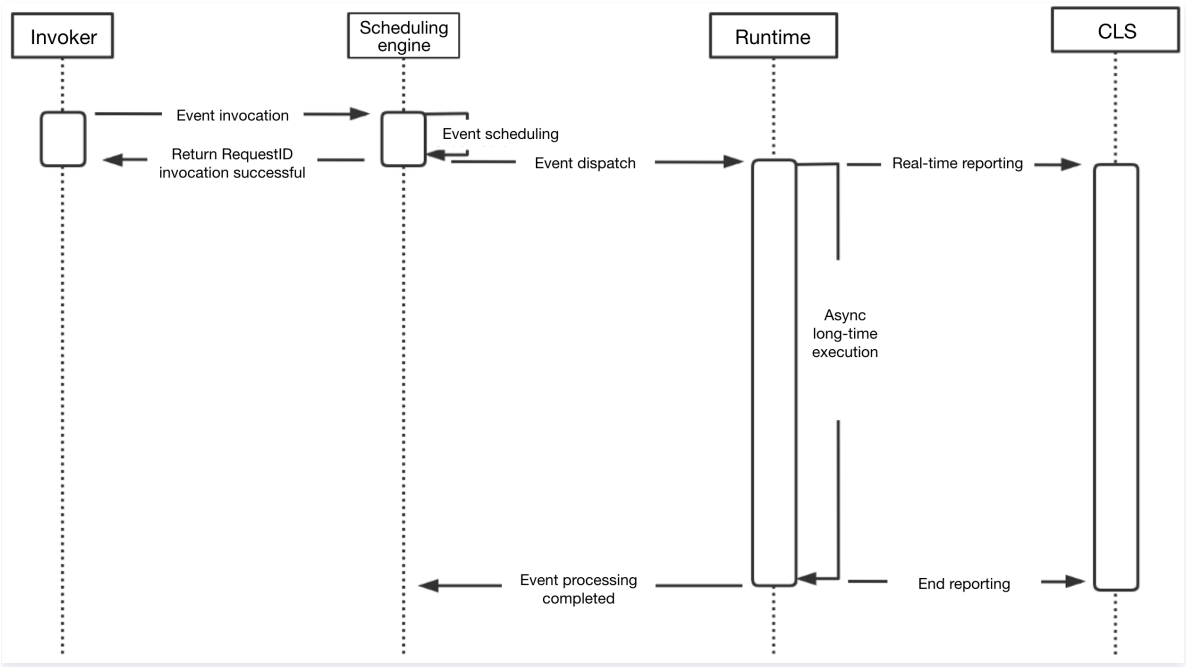
Execution Mechanism

How it works

Once the function is set to execute asynchronously, it responds to events in asynchronous execution mode when called by either synchronous (such as API Gateway) or asynchronous (such as COS, CKafka, Timer, etc.) callers.

Upon completion of event scheduling, it immediately returns the event call identifier, RequestId, and ends the call operation, eliminating the need for the caller to wait in a blocked state. Simultaneously with the return of the RequestId, the call engine dispatches the event to the function runtime in parallel, initiating the execution of the function logic.

Once in asynchronous execution mode, execution logs are reported in real time to the log service, providing real-time feedback on the execution status of asynchronous events. The principle is illustrated in the following diagram:



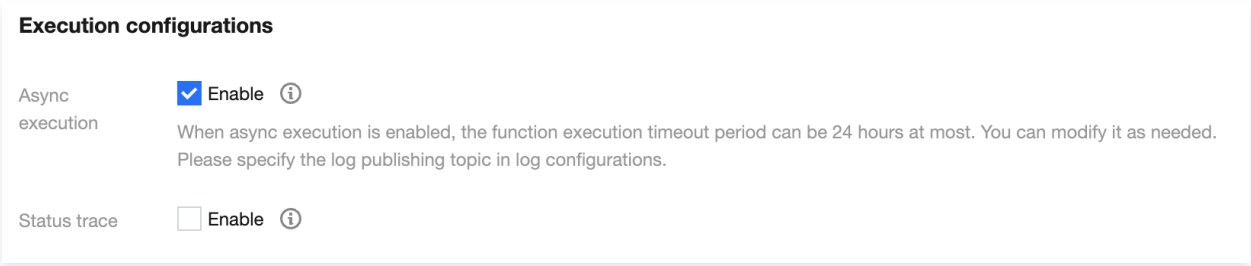
Precautions

- 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.

Procedure

1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
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. Choose to create a function using a **template** or **from scratch**.
4. On the **Function Configuration** page, expand **Advanced Settings** and select **Asynchronous Execution**, as shown below:



5. Click **Complete**.

Status Trace

Last updated: 2023-09-28 11:15:10

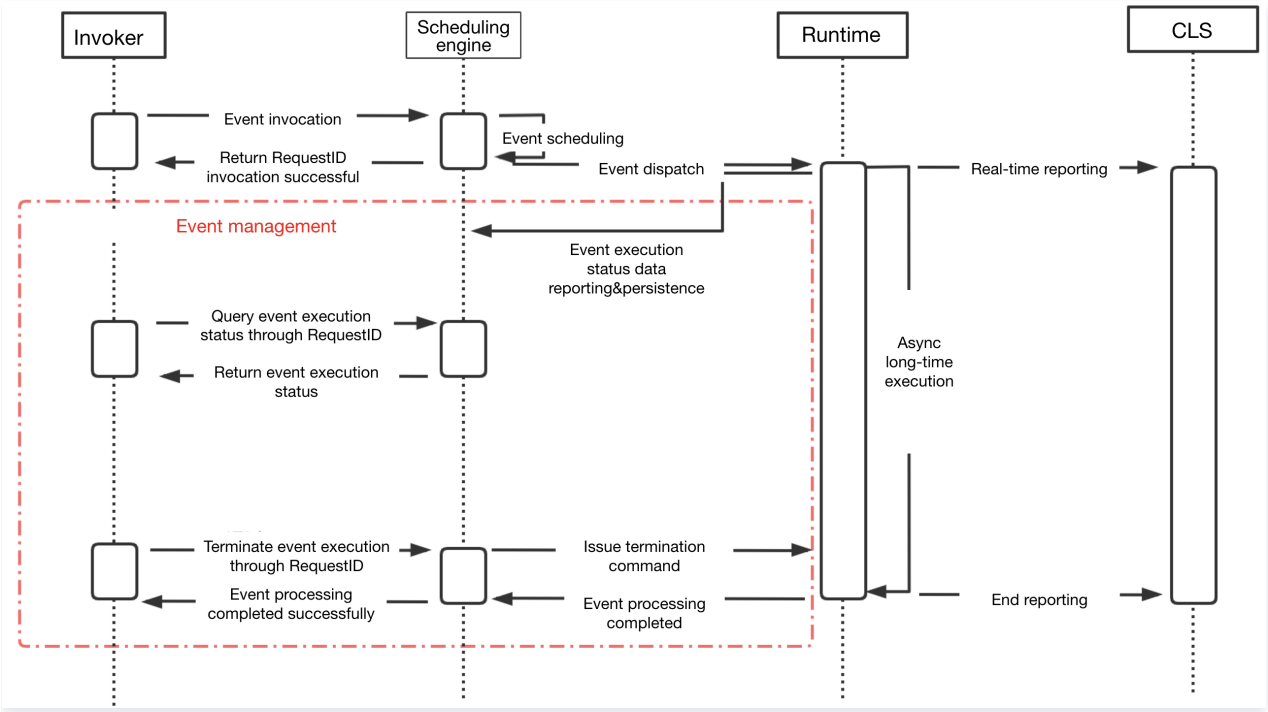
Use case

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

Once the status trace feature for asynchronously executed functions is enabled, the platform will start recording and reporting the real-time status of events. The principle is illustrated in the following diagram:

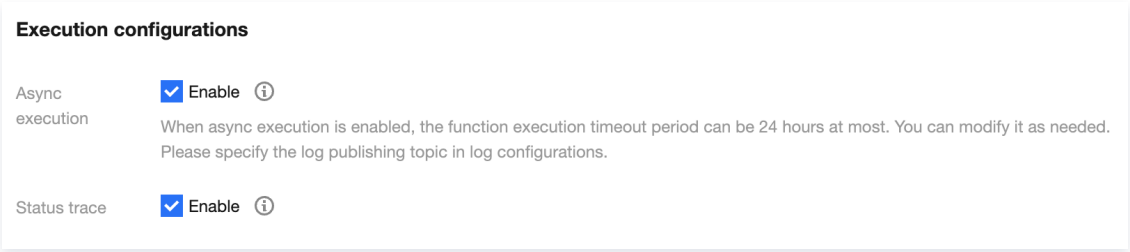


Precautions

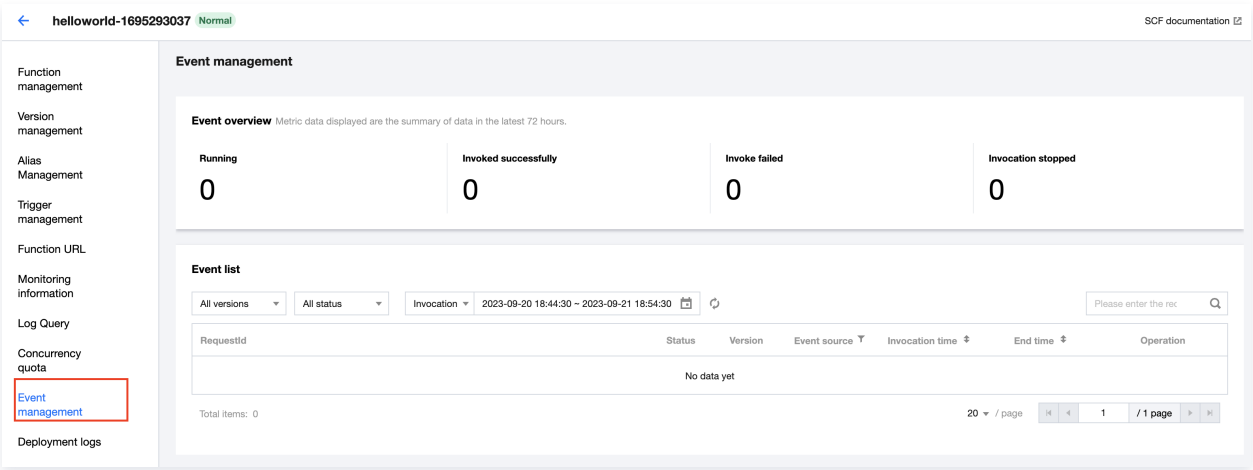
- The status data of asynchronously executed events is retained for only 3 days and will be cleared on a rolling basis in a time window of 3 days. If you want to keep all records, you need to periodically pull them and save them to your own storage.
- After status trace is disabled, event management 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.

Procedure

1. Log in to the Serverless console and click on **Function Service** in the left navigation bar.
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. Choose to create a function using a **template** or start **from scratch**.
4. On the **Function Configuration** page, expand **Advanced Settings**, select **Asynchronous execution**, and then select **Status trace**, as shown below:



5. Click **Complete**. After the function is created, you can click **Event management** to view the asynchronous event list.



Async Event Management

Last updated: 2023-09-27 18:49:43

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 is asynchronously executed unsuccessfully with an abnormal response.
- Invocation terminated: the user actively terminated the event in progress, and async execution stopped.

Relevant APIs

Related APIs	Note	Documentation
ListAsyncEvents	This API is used to list 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.	Pull Asynchronous Function Event List
GetAsyncEventStatus	This feature allows you to retrieve the execution status of an asynchronous event based on the `RequestId`. The event status is retained for 72 hours, starting from the end of the event.	Retrieve Asynchronous Function Event Status

> **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.

Terminates 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

Related APIs	Note	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.	Terminating an ongoing asynchronous function event

Terminating invocation

When a function is invoked, SCF allocates an instance to handle the function request or event. Once the function code has finished running and returned, the instance will handle other requests. If all instances are in use when a request arrives, a new instance will be allocated by the cloud function. (For more information on instances, please refer to 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.

Applicable scenarios

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.

Considerations

Termination invocation will forcibly stop the instance and trigger instance reclamation, which means that cached information in the instance (such as files in the /tmp directory) will not be retrievable. If you need to use this feature, please promptly write the cached issues in the instance to other persistent storage media to avoid file loss after instance reclamation.

Sending termination signal

When the asynchronous event API of the termination function is called and the `GraceShutdown` parameter is set to `True`, SCF will send a termination signal to the event specified in the API parameters. This signal is fixed as `SIGTERM`. You can listen for this signal in the function and customize the processing logic after receiving the signal, including but not limited to stopping function execution. After an asynchronously executed function event receives the `SIGTERM` signal:

- If the function code listens and defines a signal handling function, the corresponding signal handling function logic will begin to execute;
- 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:
  - Signal successfully received: The log will record [PLATFORM] Signal received successfully.
  - Signal reception failed: The log will record [PLATFORM] Signal reception failed.

Scenarios

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.

How to Use

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

```python
# -*- 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 graceshutdown(self, *agrg):
        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")
```

Golang

```
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, *agrg):
        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)
```

Namespace Management

Last updated: 2023-09-28 11:25:36

Operational Overview

Namespaces provide a relatively independent runtime environment for functions. When creating a cloud function, you can select the namespace in which the function resides, thereby managing your cloud functions more effectively.

Use limits

The following restrictions apply to the use of namespaces:

- The namespace name can contain up to 60 characters, must start with a letter, and can include a - z, A - Z, 0 - 9, -, _. It must end with a number or a letter, for example, `Tencent-Cloud_Space1`.
- The Default space cannot be modified or deleted.
- Currently, you can create up to 5 namespaces in each region by default, and up to 50 functions within each namespace. If you wish to increase these quotas, you can apply by submitting a ticket.

Procedure

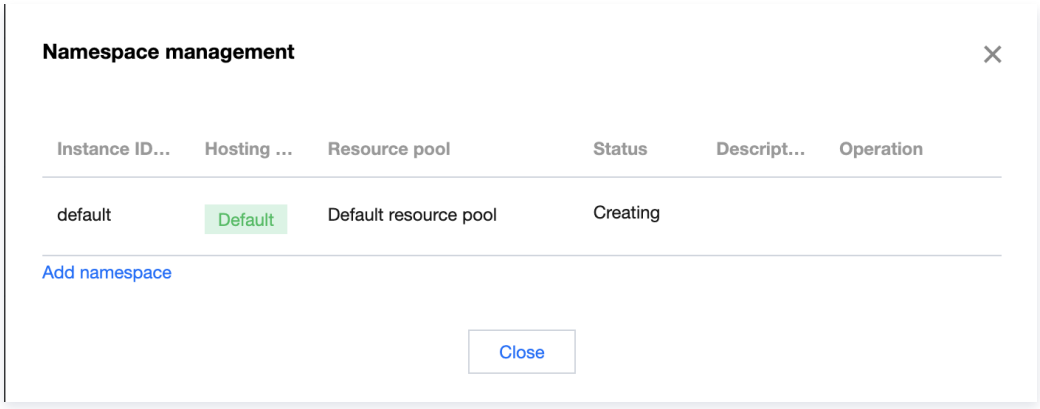
Viewing namespaces through the console

1. Log in to the SLS console.
2. Click **Function Service** on the left sidebar to enter the Function Service management page.
   At the top left of this page, you can see the information of the **namespace**, such as default(1), where 'default' is the namespace where the current function is located, and '1' is the number of functions in the namespace.
   If you need to view functions deployed under different namespaces, you can select and switch by clicking the **namespace** dropdown list.

Managing Namespaces

You can independently manage namespaces through the Serverless Console, such as creating, modifying, and deleting namespaces. The specific steps are as follows:

1. Log in to the SLS console.
2. Click **Function Service** on the left sidebar to enter the Function Service management page.
3. At the top left of this page, find **Namespace** and click on the right ⚙ to enter the Namespace management interface, as shown in the figure below:



> **Note**
> - Once a space name is determined, it cannot be altered.
> - The **default** space is the default namespace. If not specifically set, functions will be created within this space by default.

4. Within the Namespace management page, the following operations can be performed:

- To create a new namespace within this region: Click on **Add namespace**, enter the namespace name and select **Submit** to complete the creation.
- To modify the namespace description: Edit the description information and click **Submit** to finalize the changes.
- To delete a namespace: Please first remove all functions under the namespace to be deleted, then click **Delete** on the right side of the namespace on the "Namespace Management" page.

ICP Filing

Last updated: 2023-09-27 18:59:46

Why is ICP filing necessary?

In accordance with the State Council Order No. 292 "Internet Information Service Management Measures" and the Ministry of Industry and Information Technology Order No. 33 "Non-operational Internet Information Service Filing Management Measures", the state implements a licensing system for operational Internet information services and a filing system for non-operational Internet information services. It is illegal to engage in Internet information services without obtaining a license or completing the filing procedures.
Therefore, when using Serverless services in the Chinese mainland to establish a website and bind domain name services, it is mandatory to first complete the website filing. Only after successfully filing and obtaining the ICP filing number issued by the Communications Administration can domain name access be activated.

- The ICP filing number is subject to public query on the Ministry of Industry and Information Technology website: Query Entrance
- For more information on relevant laws and regulations, please see: Laws and Regulations

ICP Filing Scenarios

If your website is hosted in Tencent Cloud's Serverless service in the Chinese mainland, and the organizer and domain name of the website have never obtained an ICP filing, then you need to perform the initial ICP filing operation in the Tencent Cloud ICP filing system first before activating the Serverless service and using SCF for HTTP access to the custom domain name.

ICP Filing Preparations

- To expedite the filing process and ensure a successful submission, we recommend familiarizing yourself with the ICP filing procedures in advance.
- Given the varying requirements of different local administrations, the materials needed for preparation may differ. We recommend that you familiarize yourself in advance with the ICP filing requirements of each province, autonomous region, and municipality, as well as the related ICP filing restrictions.
- Serverless ICP Filing Requirements: While there is no charge for the ICP filing itself, filing via Serverless requires the purchase of a cloud function package with **50 million** invocation times and a resource usage package of **400,000 GBs**. Please proceed to the Resource Package Purchase Page to complete your purchase.

> **Note**
> The Serverless ICP filing method is now available to all users.

ICP Filing Process

Please refer to Initial ICP Filing and complete the ICP filing operation through the mini program.

> **Note**
> During the Website Information Entry step, you need to initiate the cloud function filing first, that is, select **Serverless** in the filing type.

Troubleshooting

Is it necessary to file an ICP for the access domain name when using Serverless?

The necessity of filing depends on the actual situation. If the default access is through the third-level domain provided by the API Gateway, there is no need for filing. However, if you need to customize the domain name and this domain name points to a Serverless service in mainland China, filing is required. For instance, scenarios such as accessing blog pages. You can determine whether filing is necessary based on the following scenarios:

- No ICP filing required:
  When the domain name resolution points to a Serverless service hosted outside mainland China, such as a server in Hong Kong, China, there is no need for ICP filing.
- ICP filing required:

When the domain name points to a Serverless service in mainland China, it is necessary to complete the ICP filing.

Is the ICP filing process for Serverless the same as that for CVM?

There is no substantive difference between the ICP filing process for Serverless and CVM, the experience is entirely consistent. The only difference is that the IP filled in during the CVM filing process will be directly exposed to the user, while the IP in the Serverless filing process will be automatically fetched and provided by the system.

Why is the IP address inaccessible during the ICP filing process?

The ICP filing is done with Tencent Cloud, and the DNS resolution also needs to be done with Tencent Cloud. During the ICP filing process using Tencent Cloud's Serverless method, the IP address for domain name resolution should point to the IP address of a Tencent Cloud site.

Are there any restrictions on Serverless ICP filing?

Given the lightweight nature of Serverless, it may be deleted or added in real-time. To cater to the user habits of using Serverless, the ICP filing for Serverless will be based on the account dimension. Each account is allowed to purchase one cloud function resource package and file for two websites.

Does a domain that has already been filed with CVM need to be re-filed?

If a domain has already been filed for ICP under the same account in CVM, it can be directly bound through the API Gateway, eliminating the need for duplicate filings.

Does a currently unused registered domain still require ICP filing?

The domain itself does not require ICP filing, but real-name authentication is necessary. ICP filing is only required when the domain initiates a Web service.

Does a Serverless site that has not been fully set up require ICP filing?

No, ICP filing is not required. It is only necessary before your Serverless site officially provides external access. As the filing process takes some time, it is recommended that you handle it in advance with Tencent Cloud, so your site can be put into use immediately after it is ready.

Does a domain that has already been filed for ICP need to be re-filed when accessing Tencent Cloud Serverless services?

No, re-filing is not necessary, but ICP filing for access is required. For more details, please see ICP Filing for Access.

Under what circumstances is it necessary to file for a new website ICP?

- If you have multiple domain names, each one requires ICP filing.
- You have already filed an ICP for a domain name, and now you need to file for a new domain name.

Can ICP filing for access be applied to multiple Serverless services?

The same entity can simultaneously file for access to multiple websites, with a maximum of 10 cloud functions or CVM filing information at the same time.

Is a refund available after the purchase of a Serverless ICP filing resource package?

The resource package takes effect immediately after purchase and refunds are not currently supported. For a five-day no-reason refund and other special refunds, please consult with online customer service. After a successful refund through manual channels, you will not be able to repurchase the ICP filing resource package within seven days.

Extended Storage Management
Mounting CFS File System

Last updated: 2023-09-28 11:26:42

Overview

Tencent Cloud File Storage (CFS) offers a scalable shared file storage service, which can be used in conjunction with Tencent Cloud servers, container services, or batch processing services. Adhering to the standard NFS file system access protocol, CFS provides a shared data source for multiple computing nodes, supports the elastic expansion of capacity and performance, and can be mounted for use without modification to existing applications. It is a highly available and reliable distributed file system, suitable for scenarios such as big data analysis, media processing, and content management.

CFS is cost-effective, adopting a pay-as-you-go billing model, with an hourly billing cycle. You only need to pay for the actual storage space used. For details on CFS billing, please refer to the 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 function execution space is unlimited.
- Multiple functions can share the same file system so as to share files.

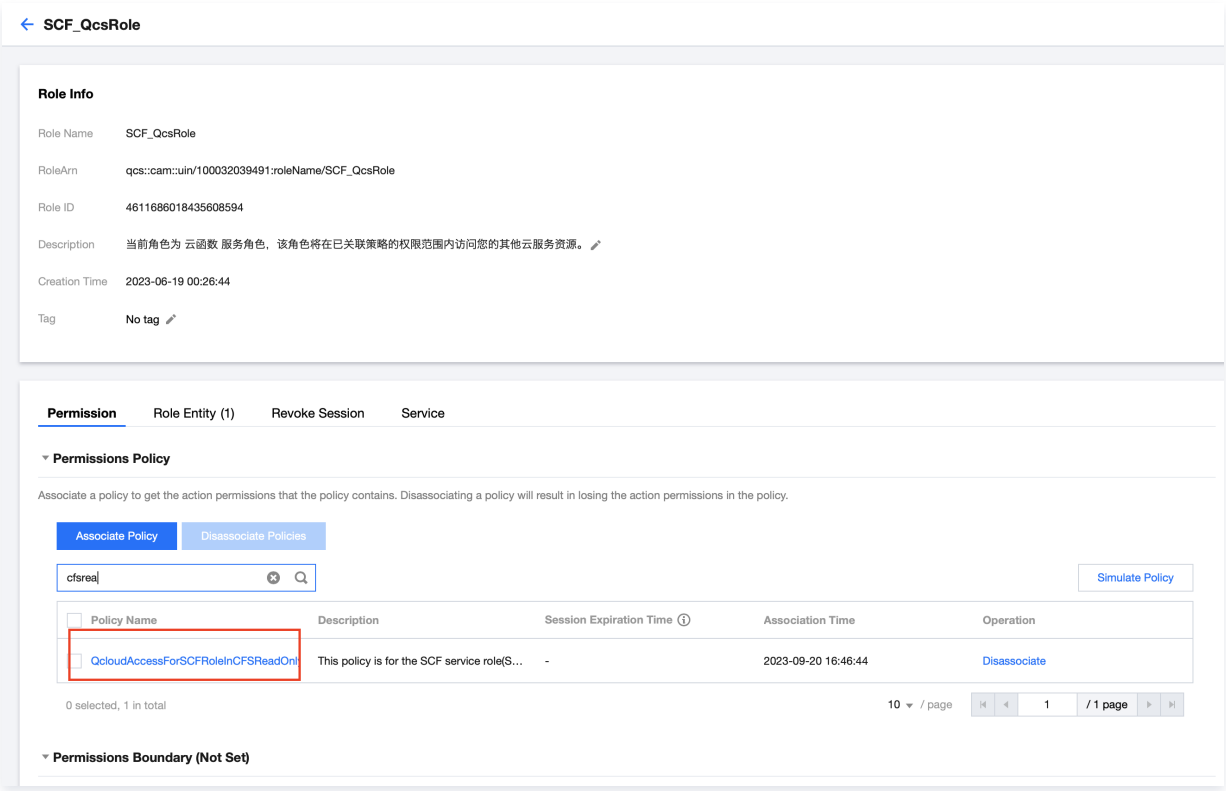
Procedure

Associating authorization policy

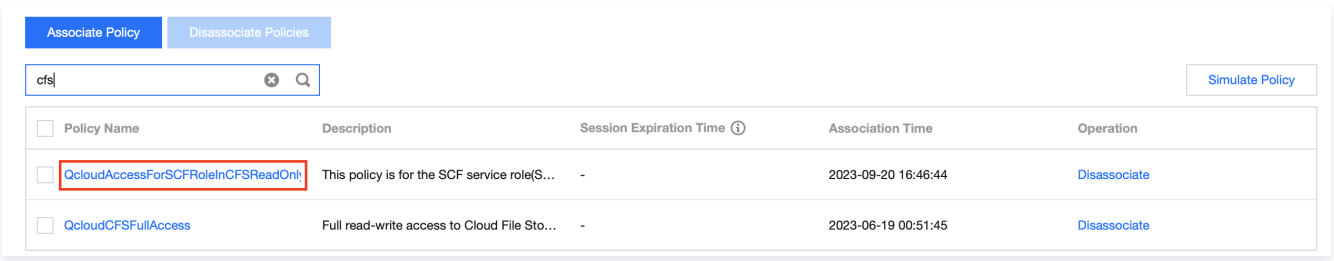
> **Note**
> To use the CFS service, SCF needs permission to operate on your CFS resources.

Please follow the steps below to authorize the account:

1. Please refer to Modify Role to associate the `SCF_QcsRole` role with the `QcloudCFSReadOnlyAccess` policy. Successful association is as shown in the figure below:
   If the account you are using has not performed this operation, you may encounter issues such as inability to save functions and unavailability of CFS related functions.



2. If you are using a sub-account, please contact the root account and refer to Sub-user Permission Settings to associate your sub-account with the `QcloudCFSReadOnlyAccess` policy. Successful association is as shown in the figure below:
   If the sub-account you are using has not performed this operation, you may encounter issues with using CFS related functions.



Creating a Virtual Private Cloud (VPC)

Please refer to Quickly Building an IPv4 Private Network to complete the creation of the VPC.

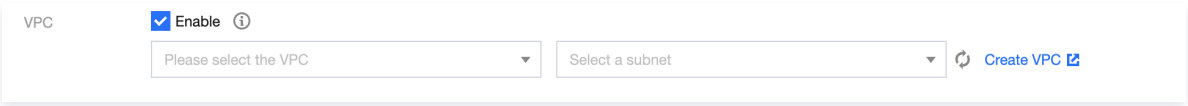
Creating CFS resource

Please refer to Creating a CFS File System to complete the creation process.

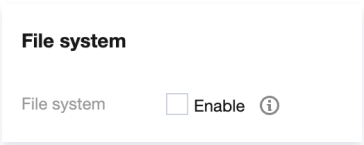
> **Note**
> Currently, SCF allows only CFS file systems whose network type is VPC to be added as mount targets. When creating a CFS file system, please select the same VPC as that of the target function, so as to enable communication.

Mounting and using CFS file system

1. Log in to the Serverless console and click on Function Service in the left navigation bar.
2. On the "Function Service" page, select the function name that needs to be configured.
3. On the "Function Management" page, under the **Function Configuration** tab, click **Edit** in the upper right corner.
4. In "Private Network", check the box to enable and select the VPC where the CFS file system is located, as shown below:



5. In "File System", check the box to enable, and mount according to the following information, 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**: remote directory in the CFS file system to be accessed by the function, which consists of a file system and 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 to complete the configuration.
   You can execute 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/filel.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: 2023-09-28 14:34:15

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.

Applicable Scenario

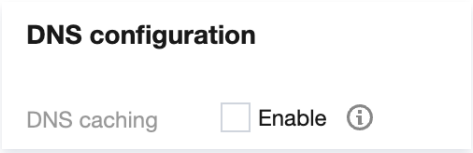
This feature is applicable to scenarios where an address is requested in the function code and the function is invoked frequently.

Procedure

Due to different implementation mechanisms, please refer to the following steps to enable DNS caching for event functions, HTTP-triggered Functions, and functions deployed with images, respectively.

Code deployment-based event-triggered function

1. Log in to the Serverless Console, select the function for which you want to enable DNS caching, and navigate to the function details page.
2. On the function configuration page, click **Edit** in the upper right corner, and in the edit mode, check **Enable DNS Caching**. As shown in the figure below:



3. Click **Save**.

HTTP-triggered function

1. In the startup file scf_bootstrap of the HTTP-triggered function, add the following command to start the nscd process and enable DNS caching.

```
/var/lang/bin/nscd -f /var/lang/conf/nscd.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 the image creation process. For instance, in CentOS, 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. In the startup file `scf_bootstrap`, add the following command to start the nscd process and enable DNS caching. For instance, in CentOS, add the following command to the startup file:

```
${PATH}/nscd -f /etc/nscd.conf
```

> **Note**
> `${PATH}` is the absolute path where nscd is installed.

Managed Resource Hosting Mode

Last updated: 2024-03-25 16:21:42

Overview of Function Resource Hosting Mode

The function resource hosting 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 128GB function concurrency quota. You can also enhance the concurrency limit of a region or even a namespace by purchasing package deals. 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 business scenarios, we now support a custom resource hosting mode for functions. This allows functions to run on your specified infrastructure, such as public cloud TKE clusters, hybrid clouds, IDCs, and more.
The platform has now launched the K8s resource hosting 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 Hosting Modes

SCF supports two types: the **Default Resource Hosting Mode** and the **K8s Resource Hosting Mode**.

- In the **Default Resource Hosting 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 business scale requirements and ensure business operations by adjusting the concurrency limit of the function.
- In the **K8s Resource Hosting 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 invoking them within the given resource pool to fully utilize resources.

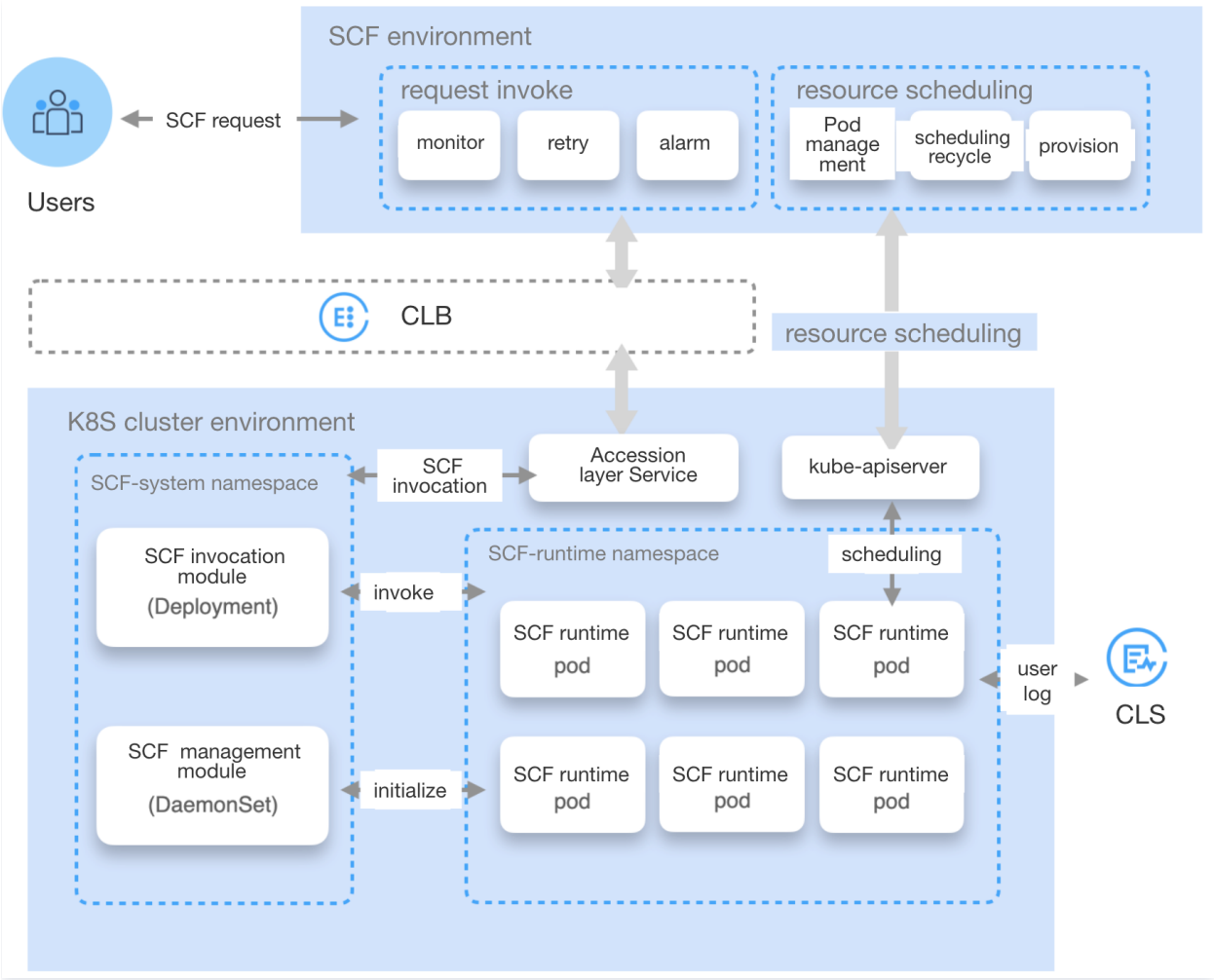
K8s Resource Hosting Mode

Overview

In the K8s Resource Hosting Mode, you can select a TKE cluster as the computing resource pool for the function. All function request invocations 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 hosting 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 Hosting 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, internal network clb service components, and so on.

Function Request Scheduling Lifecycle

The function request sent by the user will first be directed to the request invocation 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 internal network CLB. Then, the function invocation 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 invocation module in the function environment.

Advantages

Compared to the default resource hosting model, the K8s resource hosting model offers the following advantages:

- Functions can run in your designated K8s cluster, offering greater flexibility and control, 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. Not

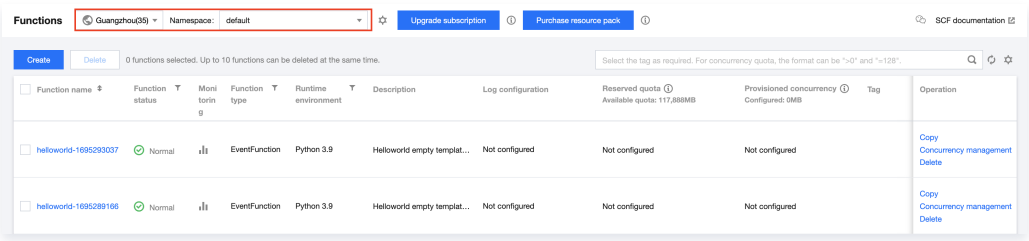
only does it improve R&D efficiency through the function development experience, but it also reduces resource waste, truly achieving cost reduction and efficiency enhancement.

- By integrating 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 business developers and higher availability guarantees for online businesses.

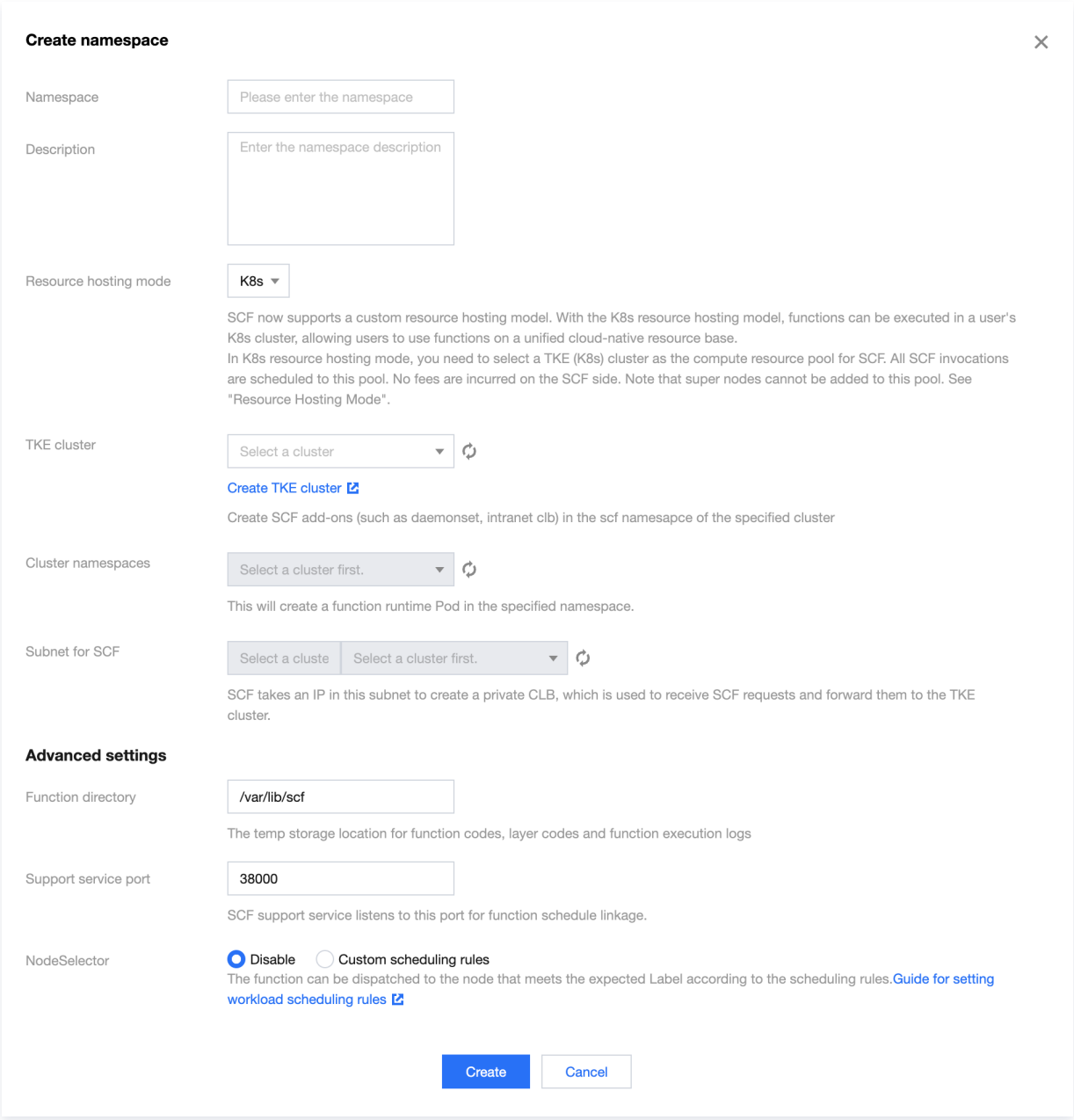
Instructions

Create a function namespace and bind it to the TKE cluster

1. Log in to the Serverless Console and click **Function Service** on 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:



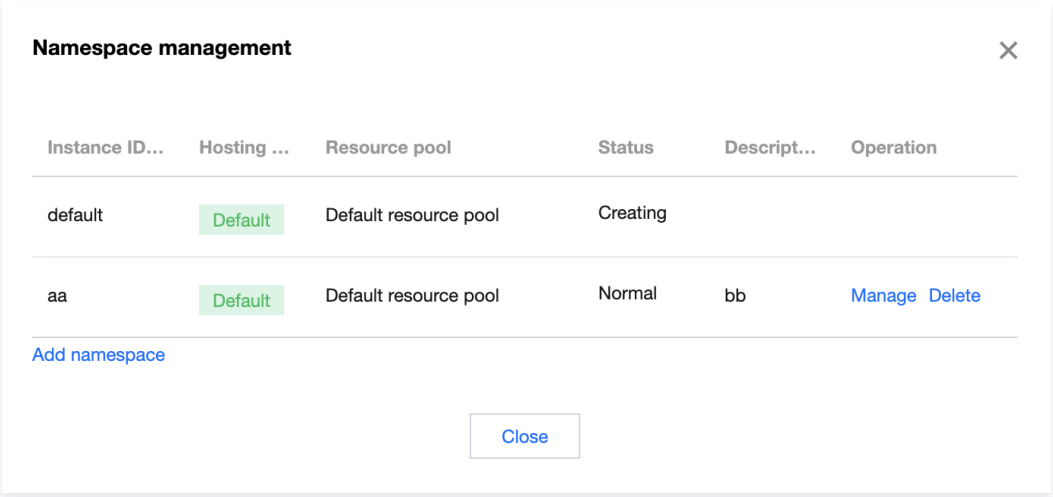
3. In the **Namespace** pop-up window, click **Add New Namespace** to navigate to the namespace creation page, as shown in the figure below:



4. In the **Resource Hosting Mode** option, 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 the **TKE Cluster Configuration**, select the TKE cluster and the namespace under that cluster. The platform will create function service support components such as daemonset and internal network 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 the initialization process is completed smoothly.**
6. In the **Function VPC Subnet** configuration, specify the subnet. The platform will consume an IP under this subnet to create an internal network CLB as the function request entry point, enabling function request forwarding to the TKE cluster. **Please note that the subnet does not support the 9.x.x.x network segment.**
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, refer to the following section Setting the Scheduling Strategy for Function Instances in the TKE Cluster.

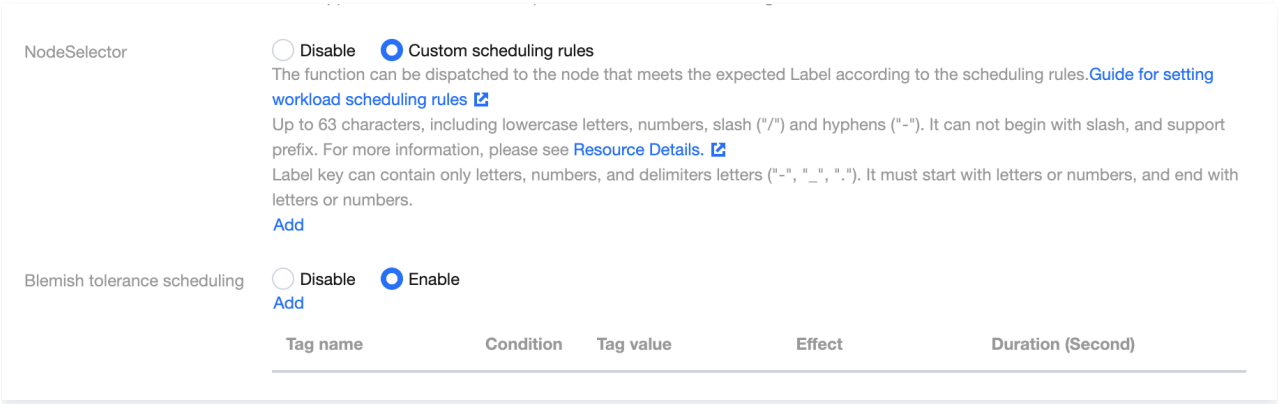
- **Taint Tolerance Scheduling**: Function instances can be scheduled to nodes with expected taints based on the scheduling rules. For more details, refer to the following section Setting the Scheduling Strategy for Function Instances in the TKE Cluster.

8. Click **Create**, 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:



9. Switch to the created function namespace to start creating and using functions.

Setting the Scheduling Strategy for Function Instances in the TKE Cluster



By setting the NodeSelector and Taint Tolerance scheduling strategies, 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 – Rational 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, machine types, etc.).

Prerequisites

- A 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 Node Labels and Setting Node Taints.

Setting a 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 refer to K8s Node Affinity.

Taint Tolerance Scheduling

Through custom scheduling rules, node taints can be tolerated, allowing function instances to be scheduled onto specified nodes. For more details, please refer to K8s Taints and Tolerations.

Contact Us

Should you encounter any issues during use, feel free to join our group for communication: