# Serverless Cloud Function

# Developer Tools

# Contents

# Developer Tools
# Serverless Web IDE

Last updated: 2023-09-27 21:01:59

## Overview

Serverless Web IDE is an IDE for functions launched by Tencent Cloud Serverless in partnership with CODING based on CloudStudio, an integrated development environment for browsers. It delivers an on-cloud development experience comparable to native IDEs.
Serverless Web IDE supports:

- Complete function development, deployment, and testing capabilities.
- Terminal capabilities. Common development tools such as pip and npm and programming language development environments already supported by SCF are pre-configured in it.
- The basic capabilities of a complete IDE, such as smart prompt and code autocomplete.
- User-defined IDE configuration, which ensures a consistent IDE user experience for the development of different functions.

> **Note**
> - We will keep the personalized configuration and code status in Serverless Web IDE for you. To ensure that function modifications will take effect, deploy the modifications to the cloud in a timely manner.
> - We recommend you use the latest version of Google Chrome to get the best IDE user experience.

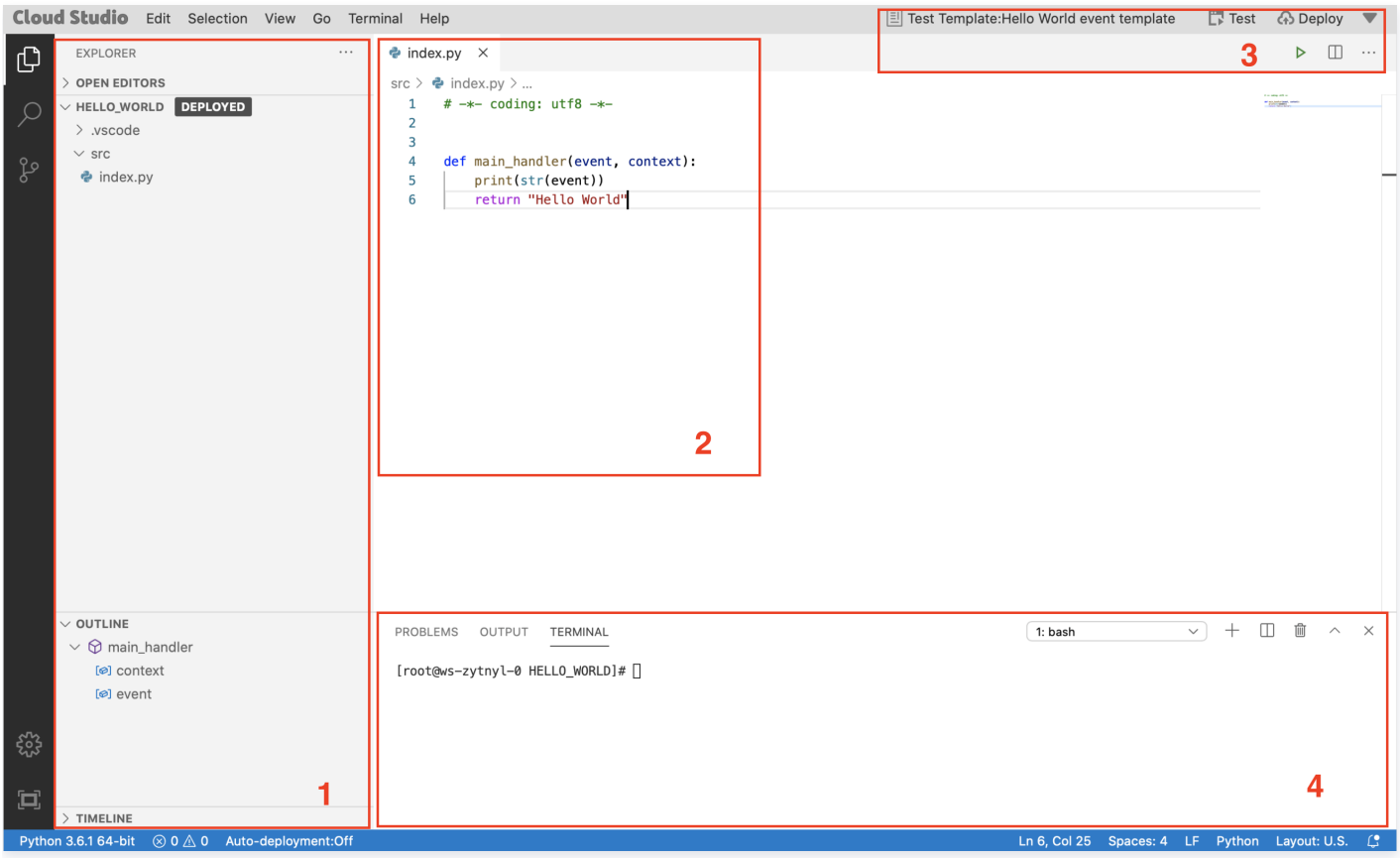
## Directions

1. Log in to the Serverless console and select **Function Service** on the left.
2. In the function list, click a function name to enter the function details page.
3. On the function management page, select **Function Code** > **Online Edit** to view and edit the function.

> **Note**
> Java and Go runtimes currently do not support online editing and only support uploading ZIP packages or binary files that have been developed and compiled. The SCF environment does not currently provide Java and Go compilation capabilities. For more information, please refer to the Golang Deployment Guide and Java Deployment Guide.

## Overview

This document will introduce the Serverless Web IDE tool in sequence from left to right as shown in the following figure:




1. **Resource Manager**
2. **File editing section**
3. **Function operation section**
4. **Command line terminal**

## Function Operations

In the Serverless Web IDE, you can complete the entire process of function code editing, deployment, and testing. Common operations such as function testing, deployment, and test template selection are all set in the operation area in the upper right corner of the IDE, as shown in the figure below:


### Function deployment

Serverless Web IDE enables you to deploy a function either manually or automatically and to install dependencies online.

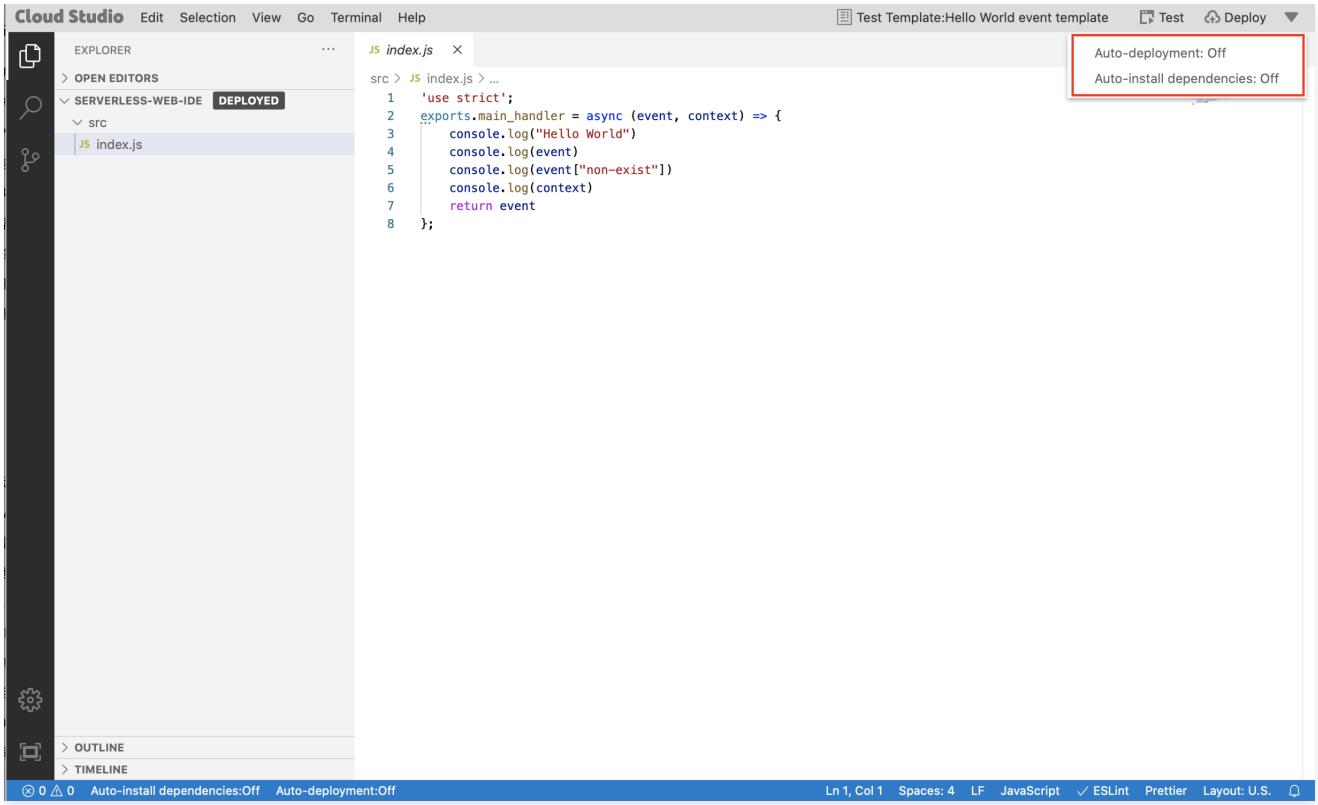
- **Deployment mode:**
  - **Manual deployment:** In manual deployment mode, you can trigger function deployment to the cloud by clicking **Deploy** in the top-right corner of the IDE.
  - **Automatic deployment:** in automatic deployment mode, you can trigger function deployment to the cloud by saving the

function (pressing Ctrl + S or Command + S).

- **Online Dependency Installation**: Currently, only the Node.js runtime environment is supported. Once online dependency installation is enabled, dependencies will be automatically installed based on the configuration in package.json during function deployment. For more details, refer to Online Dependency Installation.

> **Note**
> - The root directory of the function is `/src`, and the deployment operation will package and upload the files in the `/src` directory by default. Place the files that you want to deploy to the cloud in the `/src` directory.
> - In automatic deployment mode, you can trigger function deployment to the cloud by saving the function. Therefore, we recommend you not enable automatic deployment for functions with traffic.

You can switch between deployment modes and enable online dependency installation by clicking on **Automatic Deployment** in the dropdown list of the operation area in the top-right corner of the IDE. **Automatic Deployment: Off** indicates manual deployment mode.

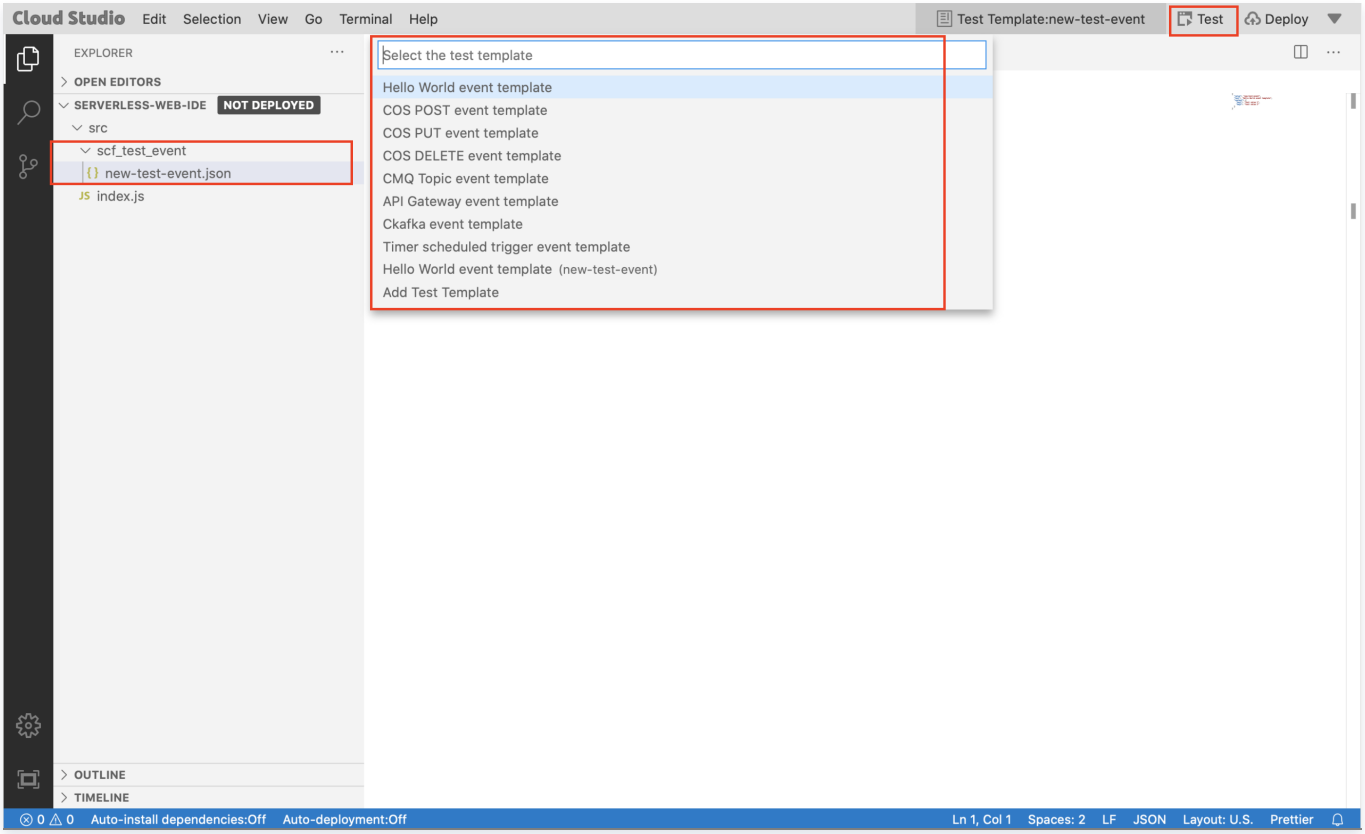



## Function testing

You can click **Test** in the operation section in the top-right corner of the IDE to trigger the function and view the result in the output.

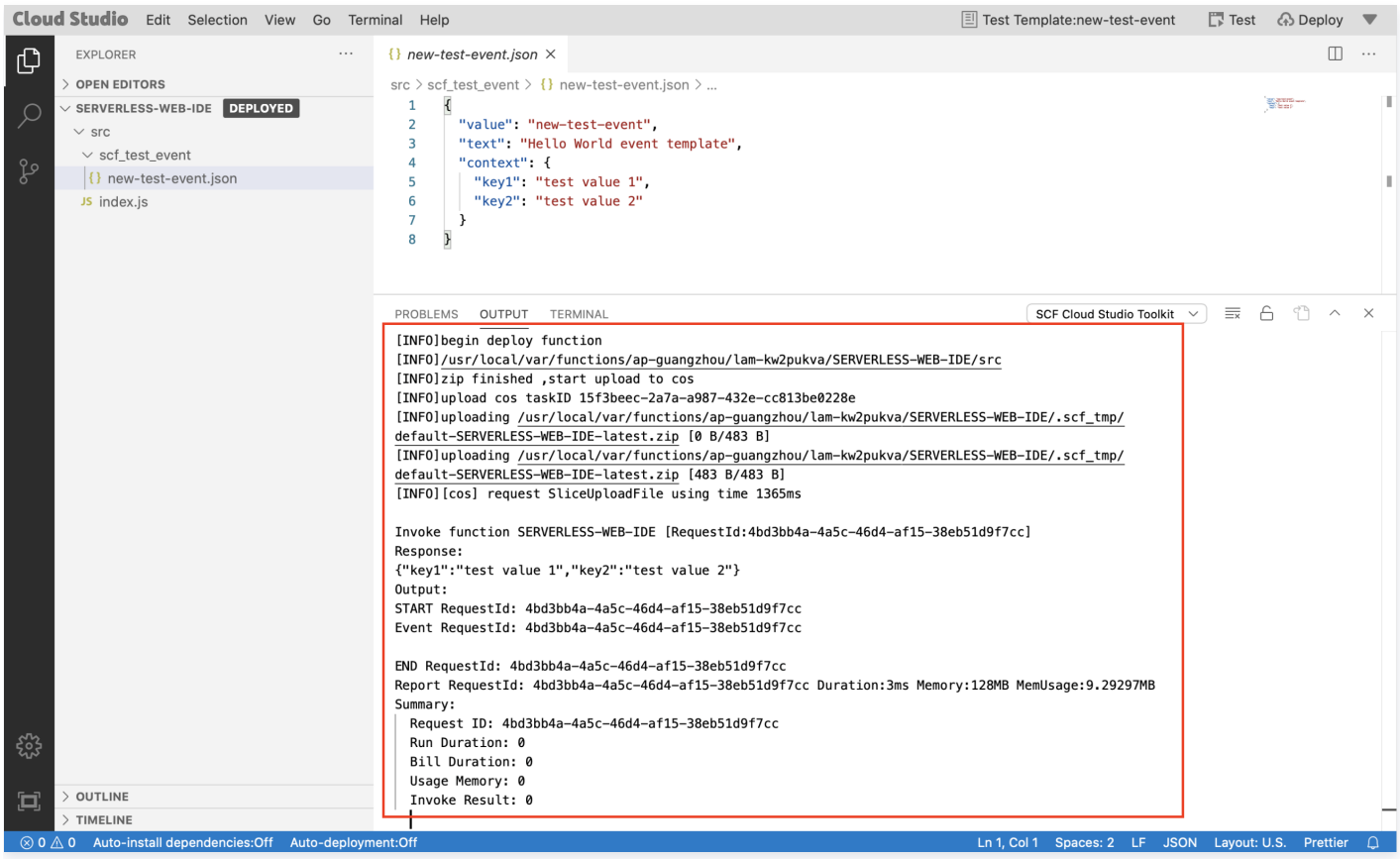
- **Select testing template**: Click **Testing Template** in the operation section of the IDE to select a function testing triggering event.
- **Add a new testing template**: If the existing testing templates do not meet your testing requirements, you can select **Add a new testing template** from the dropdown list to customize the testing event. The new testing event will be stored in the scf_test_event folder under the function root directory /src in the format of a JSON file, and will be deployed to the cloud along

with the function.




## Viewing log

You can view the function test results in the output, including the returned data Response, log Output, and function execution Summary.



### More Operations

In addition to operations such as function deployment and testing as well as testing template adding, the list expanded by right-clicking the function file in the Resource Manager contains all operations related to the function, including:

- **Generate serverless.yml**: Write the current function configuration into the serverless.yml configuration file, which can then be used for further development using the Serverless Cloud Framework command line tool.
- **Discarding current modifications**: You can re-pull the function deployed in the cloud to overwrite the current workspace.

## IDE Operations

The common commands, runtime environments, and preset extensions in Serverless Web IDE and their versions are as listed below:

### Common Commands

| Command | Version |
| --- | --- |
| python3 | Python 3.7.12. The default python3 follows the latest Python 3 version. |
| python37 | Python 3.7.12 |
| python36 | Python 3.6.1 |
| python27 | Python 2.7.13 |
| python | Python 2.7.13 |
| node | Node.js 16.13.1. The 'node' command defaults to the latest Node.js version. The environment also includes Node.js 14.18, Node.js 12.16, and Node.js 10.15, which can be switched using the 'n' command in the terminal. |
| php80 | PHP 8.0.13 |
| php74 | PHP 7.4.26 |
| php72 | PHP 7.2.2 |
| php56 | PHP 5.6.33 |
| php | PHP 7.2.2. The default PHP version in the terminal is 7.2.2. You can switch versions by executing the 'ln' command in the terminal. For instance, execute 'ln -sf /var/lang/php80/bin/php /usr/bin/php' to switch to PHP 8.0. |
| pip3 | pip 22.0.4 (python 3.7) |
| pip37 | pip 22.0.4 (python 3.7) |
| pip36 | pip 21.3.1 (python 3.6) |
| pip | pip 20.3.4 (python 2.7) |
| npm | 8.1.2 |
| composer | 2.2.9 |

### Common tools

| Tool | Version |
| --- | --- |
| yarn | 1.22.18 |
| wget | 1.14 |
| zip、unzip | 6 |
| Git | 2.24.1 |

| | |
|---|---|
| zsh | 5.0.2 |
| dash | 0.5.10.2 |
| make | 3.82 |
| jupyter | 4.6.3 |
| pylint | 1.9.5 |
| Serverless Cloud Framework | 3.2.1 |

### Runtime Environment

| Runtime Environment | Version |
|---|---|
| Node.js | 16.13、14.18、12.16、10.15 |
| Python | 3.7、3.6、2.7 |
| PHP | 8.0、7.4、7.2、5.6 |

### Others

| Others | Version |
|---|---|
| Python | 2020.11.371526539 |
| Jupyter | 2020.12.411183155 |
| PHP-IntelliSense | 2.3.14 |
| ESLint | 2.1.13 |
| Prettier | 5.8.0 |

## Quota Limit

- The IDE provides each user with a storage capacity of 5GB. Any excess will prevent write operations, so please clean up in a timely manner. Note that deleting functions will not clear the IDE's storage space. After backing up workspace changes, please manually perform the "Reset Workspace" operation. Alternatively, you can choose to switch to the older version of the editor to circumvent this limitation.
- To ensure a smooth user experience, it is not recommended to open more than three functions simultaneously across multiple browser pages.

## Supports and Limits

Performing the following actions in the IDE may pose security risks such as data leakage. If these actions are necessary, please proceed with caution.

- Install high-risk open-source components such as phpMyAdmin and Struts 2.

## FAQs

### 1. How should I handle an abnormal IDE load?

If you encounter issues such as an abnormal workspace that cannot start normally, you can click "**Having Issues**" at the bottom right corner of the IDE, and then click "**Reset Workspace**" on the expanded page to initialize the workspace.

### 2. Why does the function execute successfully in the terminal but fail when clicked for testing?

The online IDE terminal and the SCF cloud runtime environment are independent of each other. The execution result returned after clicking "**Test**" represents the actual outcome of the function execution.

### 3. The command executed in the terminal does not yield the expected result?

If you run a dependent package installation command, make sure that you run it under the `src` directory.
Before using a command, please check its version. For a list of default supported commands, refer to Common Commands.

# Serverless Cloud Framework

# Overview

Last updated: 2023-09-27 21:08:18

## Overview

Serverless Cloud Framework is a highly popular serverless application framework in the industry, allowing developers to deploy fully functional Serverless application architectures without concerning about underlying resources. Equipped with resource orchestration, automatic scaling, and event-driven capabilities, Serverless Cloud Framework covers the entire lifecycle including coding, debugging, testing, and deployment. It assists developers in rapidly building Serverless applications by linking cloud resources.

## SCF Component

Serverless Cloud Framework offers the SCF component, which allows you to quickly package and deploy cloud function projects. You can familiarize yourself with and start using the component by following these steps:

1. Get started quickly with Serverless Cloud Framework through Function Operations.
2. Understand how to develop and debug cloud functions with Serverless Cloud Framework through Development and Debugging.
3. Learn how to perform project management and resource orchestration for multiple SCF functions as instructed in Project Application.

## Best Practices

The SCF component provided by Serverless Cloud Framework facilitates the creation and orchestration of cloud function resources. It also encapsulates more components and offers best practice examples for typical user application scenarios, such as support for the Express framework and website deployment. For more details, please refer to the Serverless Application Center.

| Project Name | Description |
| --- | --- |
| Deploy Static Website Hosting | Use the `Serverless Website` component to quickly host a static website. |
| Deploy Express.js Application | Use the `Serverless SCF` component to quickly construct an Express.js project. |
| Deploy Vue + Express + PostgreSQL Full-Stack Website | Use Vue as the frontend and Express framework as the backend to deploy a serverless full-stack application through multiple Serverless Components. |
| Deploy Nuxt.js Application | Use the `Serverless Components Nuxt.js` component to quickly deploy an SSR project based on Nuxt.js. |

# Installation

Last updated: 2023-09-27 21:08:25

You can install Serverless Cloud Framework through npm.

## Installation through npm

### Prerequisites

Before installing through npm, you need to make sure that Node.js (**later than v12**) and npm have been installed in your environment (for more information, see Node.js Installation Guide) .

```
$ node -v
v12.18.0

$ npm -v
7.0.10
```

**Note**

- To ensure the installation speed and stability, we recommend you use cnpm for installation: download and install cnpm first, and then replace all the npm commands used below with cnpm commands.
- `scf` is short for the `serverless-cloud-framework` command.

### Installation Steps

Run the following command on the command line:

```
npm i -g serverless-cloud-framework
```

**Description**

If macOS prompts that you have no permission, you need to run `sudo npm i -g serverless-cloud-framework` for installation.

If you have already installed Serverless Cloud Framework, you can run the following command to upgrade it to the latest version:

```
npm update -g serverless-cloud-framework
```

### Viewing version information

After the installation is completed, run the `scf -v` command to view the version information of Serverless Cloud Framework:

```
scf -v
```

## Relevant Operations

Next step: getting started

- Quick Deployment of Function Template
- Quick Creation of Application Template

# Permission Management

Last updated: 2023-09-28 16:12:09

This document describes several authorization methods of Serverless Cloud Framework and demonstrates actual operations by configuring sub-account permissions.

## Preparations

Serverless Cloud Framework helps you quickly deploy your project to **SAC**. Before deploying, make sure that you have registered a Tencent Cloud account and completed identity verification.

## Authorization Method

### Authorizing by scanning code

When deploying via `scf deploy`, you can authorize and deploy quickly by scanning a QR code. For detailed steps, please refer to Deploy Function. After scanning the code for authorization, temporary key information (with an expiration time of 60 minutes) will be written into the `.env` file in the current directory:

```
TENCENT_APP_ID=xxxxxx     # AppId of authorizing account
TENCENT_SECRET_ID=xxxxxx  # SecretId of authorizing account
TENCENT_SECRET_KEY=xxxxxx # SecretKey of authorizing account
TENCENT_TOKEN=xxxxx       # Temporary token
```

For more information about the permissions obtained during quick authorization, see scf_QcsRole role permission list.

> **Note**
> If your account is a **Tencent Cloud sub-account**, policy authorization needs to be configured by the root account first. For more information on the configuration, please see Sub-account Permission Configuration.

### Authorizing with local key

To avoid repeated authorization due to the expiration of QR code authorization, you can use key authorization. Create a `.env` file in the root directory of the deployment and configure Tencent Cloud's SecretId and SecretKey information:

```
# .env
TENCENT_SECRET_ID=xxxxxxxxxx # SecretId of your account
TENCENT_SECRET_KEY=xxxxxxxx # SecretKey of your account
```

You can obtain the SecretId and SecretKey from API Key Management.

> **Note**
> To ensure the account security, we recommend you use a **sub-account** key for authorization. The sub-account can deploy the project only after being granted the relevant permission. For more information on the configuration, please see Sub-account Permission Configuration.

### Configuring with permanent key

By using the `scf credentials` command, you can quickly set and permanently save global key information. This command must be configured in an already created SCF project. Please ensure that you have already created your project with serverless.yml through `scf init` or manually.

- **Below are all the commands:**

```
scf credentials   Manage global user authorization information
  set             Store user authorization information
    --secretId / -i       (Required) Tencent Cloud CAM account SecretId
```

```
    --secretKey / -k        (Required) Tencent Cloud CAM account SecretKey
    --profile / -n {name}   Authorization name, default is "default"
    --overwrite / -o        Overwrite the key of an existing authorization name
remove              Remove user authorization information
    --profile / -n {name}   (Required) Authorization name
list                View user authorization information
```

- **Configure global authorization information:**

```
# Configure authorization information through the default profile name
$ scf credentials set --secretId xxx --secretKey xxx

# Configure authorization information by specifying the profile name
$ scf credentials set --secretId xxx --secretKey xxx --profile profileName1

# Update the authorization information in the specified profile name
$ scf credentials set --secretId xxx --secretKey xxx --profile profileName1 --overwrite
```

- **Delete global authorization information:**

```
$ scf credentials remove --profile profileName1
```

- **View all current authorization information:**

```
$ scf credentials list
```

- **Deploy through global authorization information:**

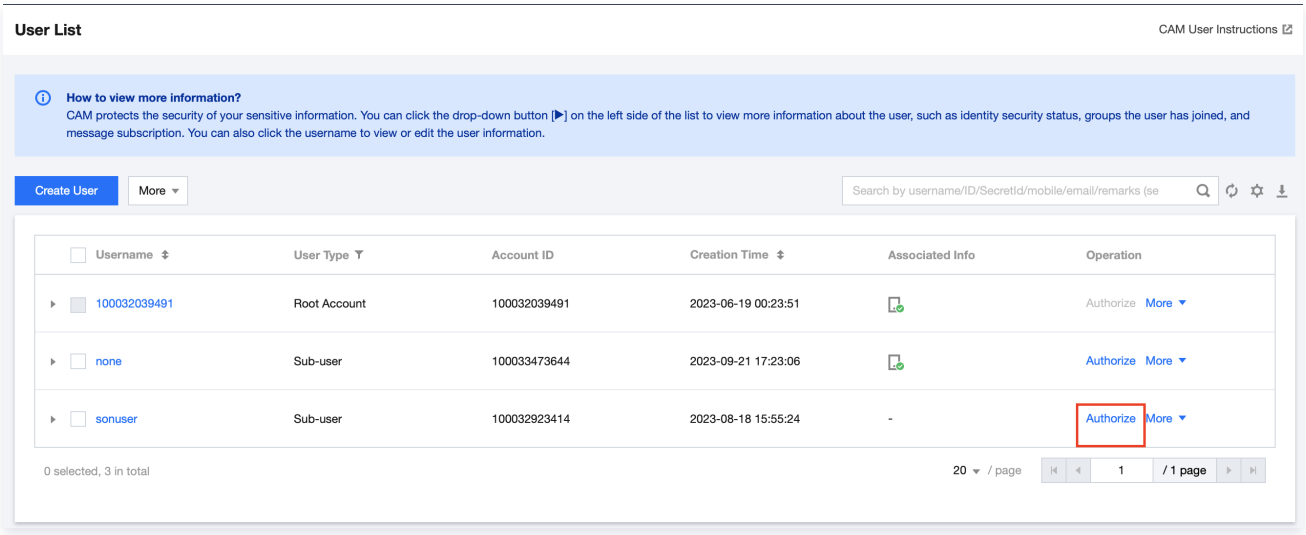
```
# Deploy through the default profile
$ scf deploy
# Deploy through the specified profile
$ scf deploy --profile newP
# Ignore global variables, deploy by scanning code
$ scf deploy --login
```

## Sub-account Permission Configuration

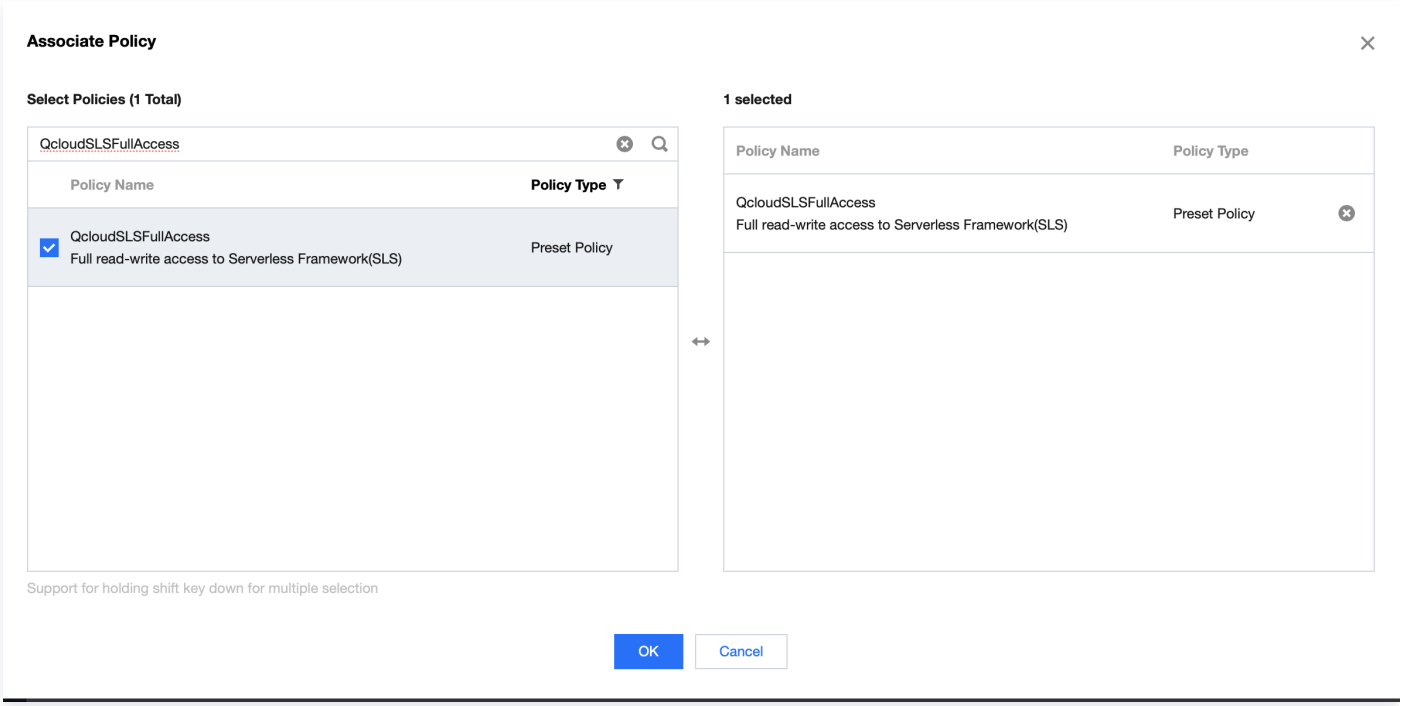
### Configuration steps

If you use a Tencent Cloud sub-account, it does not have the operation permissions by default; therefore, it needs to be authorized by the **root account (or a sub-account with the authorization permission)** in the following steps:

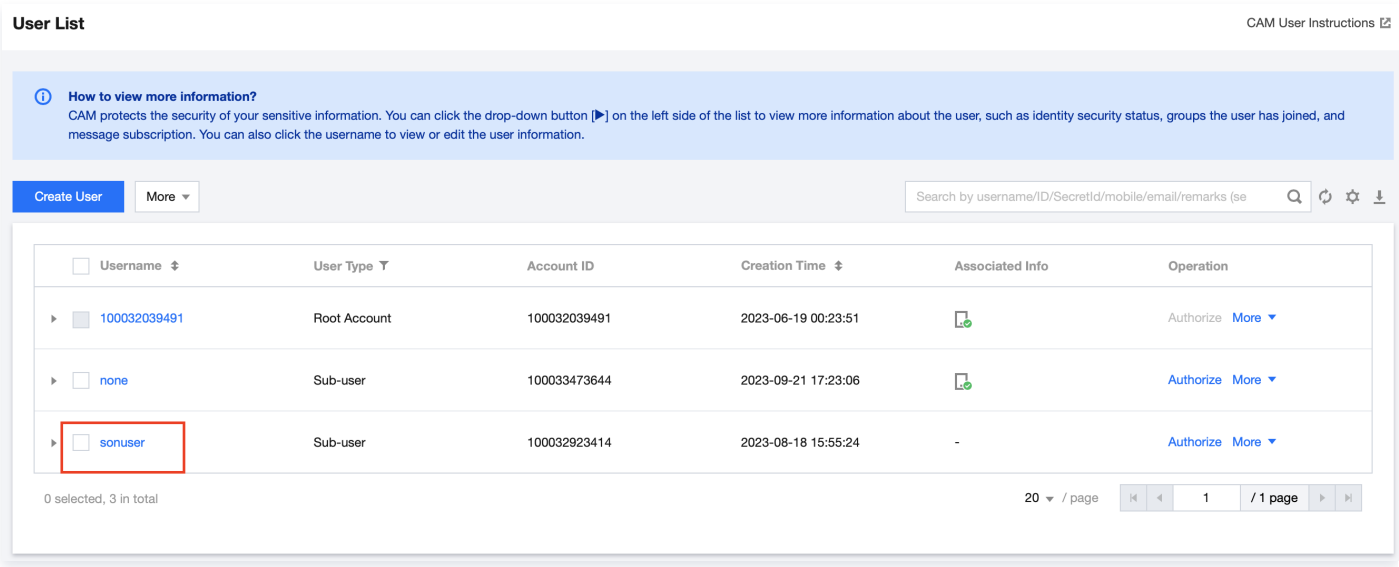
1. On the CAM User List page, select the corresponding sub-account and click **Authorize**.



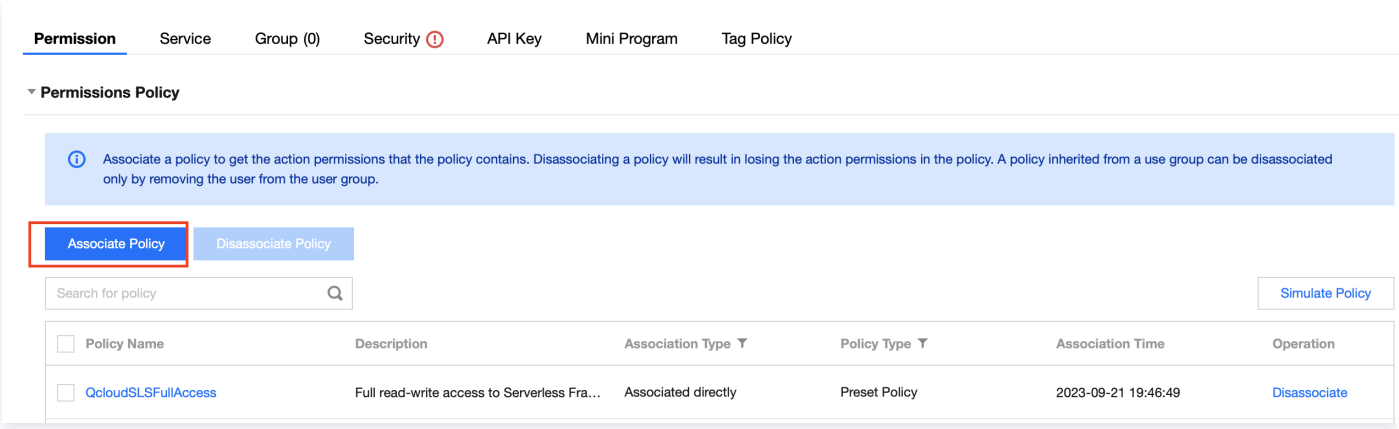

2. Search for and select `QcloudSLSFullAccess` in the pop-up window, then click **OK** to grant the sub-account the permission to manipulate all Serverless Framework resources.



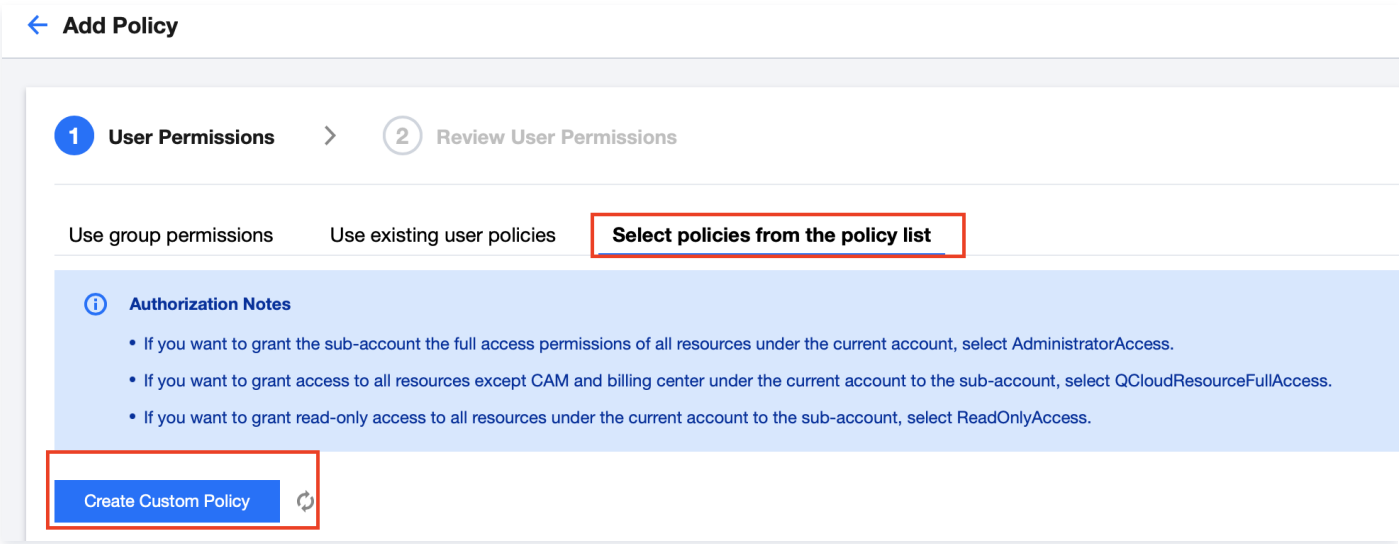

3. On the CAM User List page, select the corresponding sub-account and click on the username to enter the user details page.




4. Click **Assign Policy**, then on the Add Policy page, click **Select Policy from Policy List** > **Create Custom Policy**.
Assign Policy page:




Create Policy page:

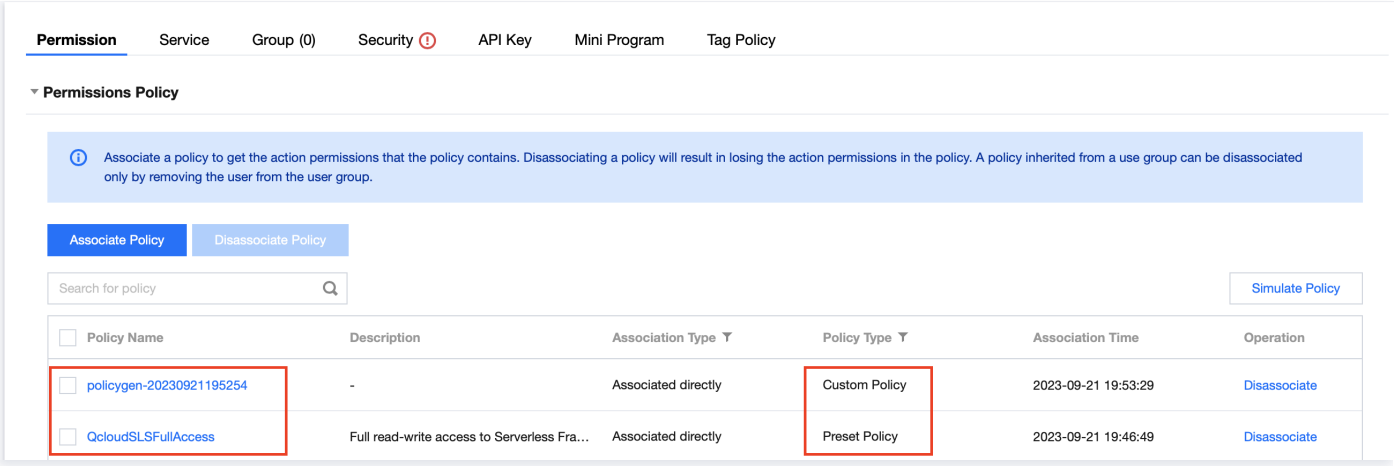



5. Choose **Create by Policy Syntax** > **Blank Template** and enter the following content. Make sure to replace the role parameter with the UIN of your root account:

```
{
"version": "2.0",
"statement": [
```

```
    {
      "action": [
        "cam:PassRole"
      ],
      "resource": [
        "qcs::cam::uin/${Enter the account's UIN}:roleName/scf_QcsRole"
      ],
      "effect": "allow"
    },
    {
      "resource": [
        "*"
      ],
      "action": [
        "name/sts:AssumeRole"
      ],
      "effect": "allow"
    }
  ]
}
```

6. After completing the custom policy configuration, return to the authorization page from step 4, search for the custom policy you just created, and click **Next** > **OK** to grant the sub-account the scf_QcsRole operation permission. At this point, your sub-account should have a custom policy and a preset policy of **QcloudscfFullAccess**, enabling normal use of the Serverless Framework.




**Note**

In addition to authorizing the invocation of the default role scf_QcsRole, you can also authorize sub-accounts to invoke custom roles. Through the fine-grained permission policies in custom roles, the goal of permission contraction can be achieved. For more details, please see Specifying Operation Role Configuration.

## scf_QcsRole role permission list

| Policies | Description |
| --- | --- |
| QcloudCOSFullAccess | Full access to COS |
| QcloudSCFFullAccess | Full access to SCF |
| QcloudSSLFullAccess | Full access to SSL Certificate Service |
| QcloudTCBFullAccess | Full access to TCB |
| QcloudAPIGWFullAccess | Full access to API Gateway |
| QcloudVPCFullAccess | Full access to VPC |

| | |
|---|---|
| QcloudMonitorFullAccess | Full access to Tencent Cloud Observability Platform |
| QcloudSlsFullAccess | Full access to SLS |
| QcloudCDNFullAccess | Full access to CDN |
| QcloudCKafkaFullAccess | Full access to CKafka |
| QcloudCodingFullAccess | Full access to CODING DevOps |
| QcloudPostgreSQLFullAccess | Full access to TencentDB for PostgreSQL |
| QcloudCynosDBFullAccess | Full access to TDSQL-C for MySQL |
| QcloudCLSFullAccess | Full access to CLS |
| QcloudAccessForscfRole | This policy can be associated with the SLS service role (scf_QCSRole) for SCF's quick experience feature to access other Tencent Cloud service resources. It contains permissions of CAM-related operations. |

# Function Operations

Last updated: 2023-09-27 21:12:23

> **Note**
> Due to the domain name's ICP filing update, you currently cannot log in by scanning the QR code during CLI deployment. You can log in by configuring a permanent key locally or visiting the URL as prompted on the command line. For more information, see Account and Permission Configuration.

## Operational Context

This document describes how to use Serverless Cloud Framework to quickly create, configure, and deploy an Serverless Cloud Function (SCF) application on Tencent Cloud.

## Prerequisites

- You have installed Serverless Cloud Framework 1.67.2 or later.

```
npm install -g serverless-cloud-framework
```

- You have registered a Tencent Cloud account and completed identity verification.

> **Note**
> If your account is a **Tencent Cloud sub-account**, get the authorization from the root account first as instructed in Account and Permission Configuration.

## Procedure

### Quick deployment

In an **empty folder** directory, run the following command:

```
serverless-cloud-framework
```

Next, initialize the project as prompted. Select the `scf-starter` template for the application and select the runtime you want to use (Node.js is used as an example here):

```
serverless-cloud-framework: No Serverless project is detected. Do you want to create a project? Yes
serverless-cloud-framework: Select the Serverless application you want to create: scf-starter - quickly deploys an SCF
function

react-starter - swiftly deploys a basic React.js application
restful-api - swiftly deploys a RESTful API using Python + API Gateway
❯ scf-starter - quickly deploys an SCF function
vue-starter - swiftly deploys a basic Vue.js application
website-starter - swiftly deploys a static website
eggjs-starter - quickly deploys a basic Egg.js application
express-starter - rapidly deploys a basic Express.js application

serverless-cloud-framework: Select the runtime of the application: scf-nodejs - quickly deploys an SCF function in Node.js

scf-golang - rapidly deploys an SCF function in Golang
❯ scf-nodejs - quickly deploys an SCF function in Node.js
scf-php - rapidly deploys an SCF function in PHP
scf-python - swiftly deploys an SCF function in Python

serverless-cloud-framework: Enter the project name: demo
serverless-cloud-framework: Installing the scf-nodejs application...
```

```
scf-nodejs › Created


The demo project has been successfully created!
```

Click **Deploy Now** to quickly deploy the initialized project to the SCF console:

```
serverless-cloud-framework: Do you want to deploy the project in the cloud now? Yes

xxxxxxxx
x  QR  x
x CODE x
xxxxxxxx
Please scan the QR code above with WeChat or click the link below to log in.
https://scflogin.qcloud.com/XKYUcbaK
Logged in successfully!

serverless-cloud-framework
Action: "deploy" - Stage: "dev" - App: "scfApp" - Instance: "scfdemo"

functionName: helloworld
description:  helloworld blank template function
namespace:    default
runtime:      Nodejs10.15
handler:      index.main_handler
memorySize:   128
lastVersion:  $LATEST
traffic:      1
triggers:
  apigw:
    - http://service-xxxxxxx.gz.apigw.tencentcs.com/release/

27s › scfdemo › Success
```

After deployment, complete the remote invocation of the function by running the following command:

```
scf invoke --inputs function=helloworld
```

> **Note**
> `scf` is short for the `serverless-cloud-framework` command.

## Viewing the Deployment Information

To view the application deployment statuses and resources again, go to the directory that is successfully deployed and run the following command:

```
cd demo # Enter the project directory. Please change to your actual project's directory name here
scf info
```

## Viewing directory structure

In the directory of the initialized project, you can see the most basic structure of a serverless function project:

```
.
├── serverless.yml  # Configuration file
├── index.js    # Entry function
└── .env # Environment variable file
```

- The serverless.yml configuration file facilitates rapid configuration of basic function information. All configuration items supported by the function console can be configured in the yml file (see full configuration information of the cloud function).
- `index.js` is the entry function of the project, which is the `helloworld` template here.
- The `.env` file stores user login authentication information. You can also configure other environment variables in it.

### Redeployment

In the local project directory, you can modify the function template and configuration file and then redeploy the project by running the following command:

```
scf deploy
```

> **Note**
> To view the detailed information during removal, you can add the `--debug` parameter.

### Continuous development

After the deployment is completed, Serverless Cloud Framework allows you to run different commands to implement capabilities such as continuous development and grayscale release for the project. You can also use this component in conjunction with other components to manage the deployment of multi-component applications.
For more information, see Application Management and List of Supported Commands.

## Troubleshooting

### No Chinese guide pops up by default when `serverless-cloud-framework` is entered.

Solution: Add the configuration SERVERLESS_PLATFORM_VENDOR=tencent to the .env file.

### What should I do if the deployment is very slow after `scf deploy` is entered in a network environment outside the Chinese mainland?

Solution: Add the configuration `GLOBAL_ACCELERATOR_NA=true` to the .env file to enable acceleration outside the Chinese mainland.

### What should I do if the deployment reports a network error after `scf deploy` is entered?

Solution: Add the following proxy configuration to the .env file.

```
HTTP_PROXY=http://127.0.0.1:12345 # Replace "12345" with your proxy port
HTTPS_PROXY=http://127.0.0.1:12345 # Replace "12345" with your proxy port
```

# Development Debugging

Last updated: 2023-09-27 21:12:43

## Development Mode

Serverless Cloud Framework supports a Development Mode (dev mode), which facilitates a more convenient process for code writing and debugging for projects in development. Within this mode, users can continuously engage in the development-debugging cycle, minimizing disruptions from other tasks such as packaging and updates.

### Entering Development Mode

Within a project, executing the `scf dev` command allows you to enter the project's Development Mode. Here is an example:

> **Note**
> Currently, the `scf dev` command only supports Node.js 10.15 and 12.16 runtime environments.

```
$ scf dev
serverless-cloud-framework
Dev Mode - Watching your Component for changes and enabling streaming logs, if supported...
Debugging listening on ws://127.0.0.1:9222.
For help see https://nodejs.org/en/docs/inspector.
Please open chorme, and visit chrome://inspect, click [Open dedicated DevTools for Node] to debug your code.
---------- The realtime log ----------
17:13:38 - express-api-demo - deployment
region: ap-guangzhou
apigw:
  serviceId:   service-b77xtixx
  subDomain:   service-b77xtixx-12539702xx.gz.apigw.tencentcs.com
  environment: release
  url:         http://service-b77xtixx-12539702xx.gz.apigw.tencentcs.com/release/
scf:
  functionName: express_component_6r6xkh60k
  runtime:      Nodejs10.15
  namespace:    default
express-api-demo › Watching
```

Upon entering dev mode, the Serverless tool will display the deployed content and initiate continuous file monitoring. When code files are updated, the tool will automatically redeploy, updating the local files to the cloud.

### Exiting Development Mode

In Development Mode, you can exit by pressing `Ctrl+C`. The resulting output is as follows:

```
express-api-demo › Disabling Dev Mode & Closing ...
express-api-demo › Dev Mode Closed
```

## Command Debugging

The Serverless Cloud Framework supports the use of the `invoke` command to trigger cloud functions for debugging. For cloud functions successfully deployed using the `scf deploy` command, execute the following command in the project directory for debugging:

```
scf invoke  --inputs function=functionName  clientContext='{"weights":{"2":0.1}}'
```

> **Note**
> - The `invoke` command must be executed in the same directory as the serverless.yml file where the function is deployed.
> - `clientContext` is the JSON string passed when triggering the function. Different triggering events can be simulated

according to the JSON string format of the trigger event template.

## In-cloud Debugging

For projects with a Node.js 10+ runtime, in-cloud debugging can be enabled, allowing debugging tools to connect to the remote environment for debugging. Examples include Chrome DevTools and VS Code Debugger.

### Enable In-Cloud Debugging

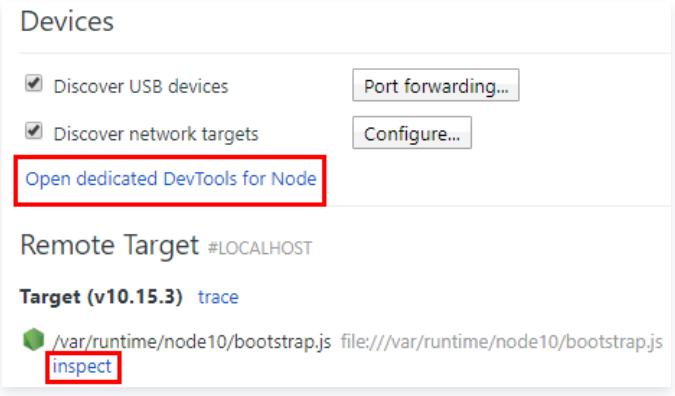
When executing step Enter Development Mode, if the project is a function with a Node.js 10 or higher runtime, it will automatically enable in-cloud debugging and output related debugging information. For instance, if the output during the activation of development mode contains information similar to the following, it indicates that the in-cloud debugging for this project has been initiated:

```
Debugging listening on ws://127.0.0.1:9222.
For help see https://nodejs.org/en/docs/inspector.
Please open chorme, and visit chrome://inspect, click [Open dedicated DevTools for Node] to debug your code.
```

### Utilizing Debugging Tool Chrome DevTools

The following steps illustrate how to use the DevTools tool in the Chrome browser to connect to a remote environment for debugging:

1. Launch Google Chrome.
2. Enter `chrome://inspect/` in the address bar and navigate to it.
3. There are two methods to open DevTools, as illustrated below:




   3.1 (Recommended) Click on **Open dedicated DevTools for Node** under Devices.

   3.2 Select **inspect** under the specific Target in Remote Target #LOCALHOST.
   If you cannot open it or there are no Targets, please check whether `localhost:9229` or `localhost:9222` is configured in the Device's Configure. This configuration corresponds to the output when cloud debugging is enabled.

4. By opening the DevTools debugging tool through the **Open dedicated DevTools for Node** option, you can view the remote code by clicking on the **Sources** tab. The actual code of the function is located in the `/var/user/` directory.
   The code viewed in the **Sources** tab may be loading and will display more remote files as debugging progresses.
5. You can open files as needed and set breakpoints at specific locations within the file.
6. Triggering the function through any method, such as URL access, page triggering, command triggering, or interface triggering, will cause the remote environment to start running. It will pause at the set breakpoint, waiting for further execution.
7. Through the right toolbar of DevTools, you can control the continuation of the interrupted program, step-by-step execution, step in and out, as well as directly view current variables, or set variables to be tracked and viewed. For further use of DevTools, you can search for the DevTools user guide.

### Exiting in-cloud debugging

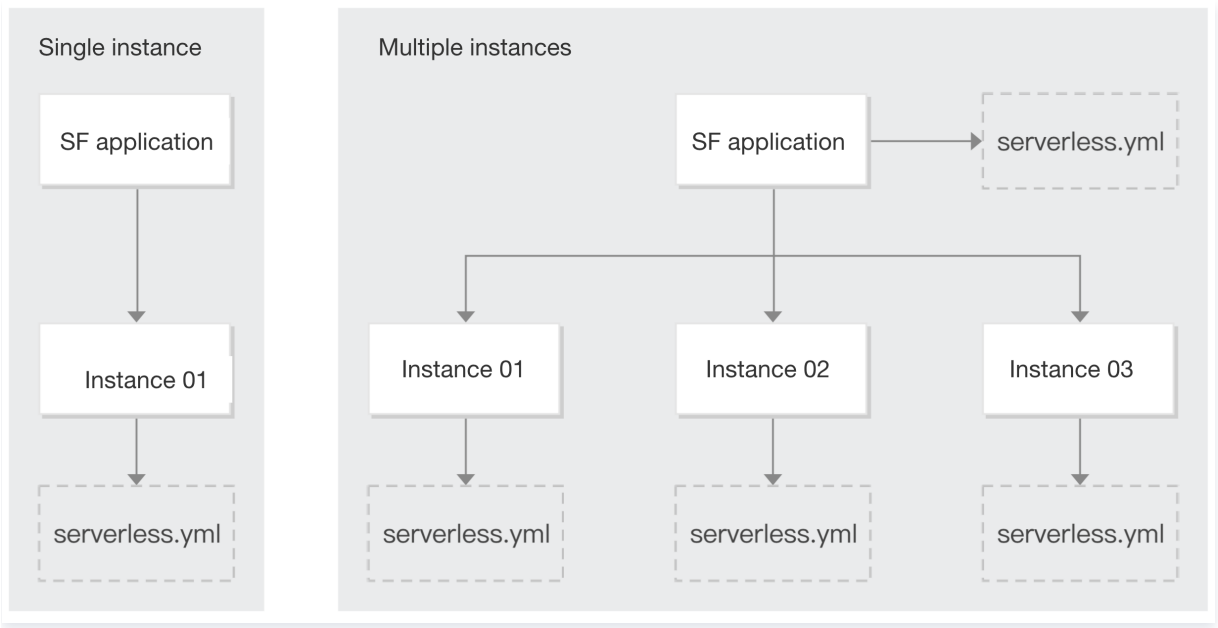
Upon exiting the developer mode, the in-cloud debugging feature will be automatically disabled.

# Project Application

Last updated: 2023-09-27 21:12:55

## Overview

The section Function Operations explains how to use the Serverless Cloud Framework to create cloud functions. From the perspective of the Serverless Cloud Framework itself, it deploys a Serverless single-instance application through the cloud function component. A Serverless application can consist of a single or multiple instances, with each component deployment corresponding to one instance. Each instance involves a serverless.yml file, which defines the parameters of the component. These parameters are used to generate the information of the instance during deployment. For instance, `region` defines the location of the resources. As shown in the figure below:




This article will help you understand single-instance and multi-instance applications, and manage your cloud function projects and resource orchestration according to actual scenarios.

## Single-instance applications

A single-instance application refers to a project that only introduces one component, and only one component instance is generated during deployment. The application name in a single-instance application is generated by default by the Serverless Cloud Framework.
**Applicable Scenarios**: The Serverless Cloud Framework is used solely as a CLI tool to create and update functions, with users needing to orchestrate and manage function resources independently.

## Multi-instance applications

A multi-instance application refers to a project that introduces multiple components, generating multiple component instances during deployment. Multi-instance applications require a fixed application name to ensure that all components are managed under a unified application.
**Applicable Scenarios**: Users need to organize and orchestrate multiple cloud function resources in a project through the Serverless Framework.

## Project Development

Serverless Cloud Framework provides a set of administrative mechanisms for resource orchestration, environment isolation, and grayscale release. In addition to creating SCF functions, Serverless Cloud Framework also provides a wealth of components for manipulating various Tencent Cloud services such as API Gateway, Tencent Cloud Object Storage (COS), and Cloud Access Management (CAM). With Serverless Cloud Framework for project development, you can focus on developing your business and improving your efficiency. For more information about project development, see Serverless Cloud Framework.

# List Of Supported Commands

Last updated: 2023-09-28 10:33:18

Serverless Application Center (SLS) is deployed based on Serverless Cloud Framework and supports the following CLI commands:

- `scf registry` : Lists available components.
- `scf registry publish` : Publishes components to the SLS component registry.
  `--dev` : Publishes components of the `@dev` version for development or testing.
- `scf init xxx` : Downloads the specified template from the component repository. After 'init', enter the name of the template you wish to download, for example "$ scf init fullstack".
  `scf init xxx --name my-app` : Customizes the project directory name.
  `--debug` : Lists log information during template download.
- `scf deploy` : Deploys a component instance in the cloud.
  `--debug` : Lists log information such as the deployment operations and the status output by `console.log()` during component deployment.
  `--inputs publish=true` : Publishes a new version during function deployment.
  `--inputs traffic=0.1` : During deployment, 10% of the traffic is switched to the $latest function version, while the remaining traffic is directed to the last published function version.

  > **Description**
  > - The legacy command format `scf deploy --inputs.key=value` has been changed to `scf deploy --inputs key=value` since Serverless CLI v3.2.3. Legacy commands cannot be used in new versions of Serverless CLI. If you have upgraded Serverless CLI, please use the new commands.
  > - `scf` is short for the `serverless-cloud-framework` command.

- `scf remove` : Removes a component instance from the cloud.
  `--debug` : Lists log information such as the removal operations and the status output by `console.log()` during component removal.
- `scf info` : Gets and displays the information about a component instance.
  `--debug` : Lists more `state` values.
- `scf dev` : Enables the development mode ("DEV Mode") and automatically deploys changed information when component status changes are detected. In development mode, information such as execution logs, invocation information, and errors can be displayed on the CLI in real time. The development mode also supports in-cloud debugging for Node.js applications.
- `scf login` : Supports logging into a Tencent Cloud account and authorizing operations on associated resources by scanning a QR code with WeChat via the login command.

# SCF VS Code Plugin Getting Started

Last updated: 2023-09-27 21:20:37

## Overview

Tencent Serverless Toolkit for VS Code is a plugin for the VS Code (Visual Studio Code) IDE of Tencent Cloud's Serverless product. This plugin enhances your local Serverless project development and code debugging experience, and effortlessly deploys your project to the cloud.
You can refer to this document to start using the Tencent Serverless VS Code plugin.
With this VS Code plugin, you can:

- Retrieve the list of serverless functions from the cloud and trigger them to run in the cloud.
- Rapidly create serverless function projects locally.
- Develop, debug, and test your serverless function code locally.
- Trigger function execution using simulated events from COS, CMQ, CKafka, API Gateway, and other triggers.
- Upload function code to the cloud and update function configurations.

## Getting started

For plugin installation and usage, see Getting Started.

## Configuration Standards

The current plugin uses the syntax standards of the tencent-component in the Serverless Cloud Framework. For specific syntax standards, see Documentation.

## Troubleshooting

- If you encounter any issues during installation or use, refer to Common SCF Tool Issues for solutions. You can also submit an issue in the Tencent Serverless Toolkit for VS Code project to document and track the problems you encounter.
- Currently, the VS Code plugin does not support image-based functions and HTTP-triggered functions.

# Calling SDK Across Functions SDK For Node.js

Last updated: 2023-09-27 21:23:40

## Tencentcloud-Serverless-Nodejs SDK Overview

Tencentcloud-Serverless-Nodejs is a Tencent Cloud SCF SDK that integrates SCF business flow APIs to simplify the function invocation method. It can be used to invoke a function quickly from a local system, CVM instance, container, or function, eliminating your need to encapsulate public TencentCloud APIs.

## Features

Tencentcloud-Serverless-Nodejs SDK has the following features:

- It can invoke functions in a high-performance, low-latency manner.
- Enables quick invocation across functions after the required parameters are entered (it will obtain parameters in environment variables by default, such as `region` and `SecretId`).
- Supports access with private network domain names.
- Supports session keep-alive.
- It supports cross-region function invocation.

> **Note**
> Calling SDK across functions is only applicable to event-triggered functions. HTTP-triggered functions can be invoked by requesting the corresponding path of the function in the function code.

## Quick Start

### Development Preparations

- Development Environment
  Node.js v8.9 or higher is installed.
- Runtime Environment
  The tencentcloud-serverless-nodejs SDK is installed, compatible with Windows, Linux, and Mac operating systems.
- We recommend you use the Serverless Cloud Framework to quickly deploy local functions.

### Tencentcloud-Serverless-Nodejs SDK installation

#### Installation through npm (Recommended)

1. Based on your actual needs, select a directory path and create a new directory within it.
   For instance, create a project directory named `testNodejsSDK`, with the project path being `/Users/xxx/Desktop/testNodejsSDK`.
2. Enter the `testNodejsSDK` directory and run the following commands in sequence to install the Tencentcloud-Serverless-Nodejs SDK.

```
npm init -y
npm install tencentcloud-serverless-nodejs
```

   After installation, you can see `node_modules`, `package.json`, and `package-lock.json` in the `testNodejsSDK` directory.

#### Installing through source package

Go to the GitHub code hosting page to download the latest source package and install it after decompression.

#### Using SCF to install dependencies online

Use SCF online dependency installation and execute the following command in `package.json` to proceed with the installation.

```
{
  "dependencies": {
  "tencentcloud-serverless-nodejs":"*"
 }
}
```

## Mutual function invocation

### Sample

> **Note**
> - To implement mutual recursion of functions in different regions, you need to specify the region. For the naming rules, please see Region List.
> - If no region is specified, intra-region mutual function invocation will be used by default.
> - If no namespace is specified, `default` will be used by default.
> - The invoker function should have the public network access enabled.
> - If parameters such as secretId and secretKey are not manually passed in, the function needs to be bound to a role with SCF Invoke permissions (or including SCF Invoke, such as SCF FullAccess). Please refer to Creating a Function Execution Role.

1. Create a Node.js cloud function named "FuncInvoked" in the **Beijing** region, which will be used for **invocation**. The content of this cloud function is as follows:

```
'use strict';
exports.main_handler = async (event, context, callback) => {
  console.log("\n Hello World from the function being invoked\n")
  console.log(event)
  console.log(event["non-exist"])
  return event
};
```

2. Under the `testNodejsSDK` directory, create a new file named `index.js`. Input the following sample code to create a Node.js cloud function for **initiating invocation**.

```
const { SDK, LogType }  = require('tencentcloud-serverless-nodejs')
exports.main_handler = async (event, context) => {
 context.callbackWaitsForEmptyEventLoop = false
 const sdk = new SDK({
  region:'ap-beijing'
 }) // If running in a cloud function and bound to a role with SCF invocation privileges, it will default to the authentication
information in the environment variables.
 const res = await sdk.invoke({
  functionName: 'FuncInvoked',
  logType: LogType.Tail,
  data: {
   name: 'test',
   role: 'test_role'
  }
 })
 console.log(res)
 // return res
}
```

The main parameters can be obtained as follows:

- **region**: Region of the **invoked** function. Beijing region selected in step 1 is used as an example in this document.
- **functionName**: Name of the **invoked** function. The `FuncInvoked` function created in step 1 is used as an example in this

document.

- **qualifier**: Version of the **invoked** function. If no version is specified, `$LATEST` will be used by default. For more information, see Viewing a Version.
- **namespace**: Namespace where the **invoked** function resides. If not specified, `default` will be used. For more information, see Namespace Management.
- **data**: Data passed to the **invoked** function. The invoked function can read this data from the event parameters.

3. Create an **invoking** Node.js function named "NodejsInvokeTest" in the **Chengdu** region. The main settings of the function are as follows:
   - Execution method: Select **index.main_handler**.
   - Code submission method: Select **Upload Local Zip Package**.
     Compress all files in the `testNodejsSDK` directory into a zip format and upload them to the cloud.
4. In the Serverless Console, click on the newly created function to enter the code editing page for function management, then click Test to run the function. The output results are as follows:

```
"Already invoked a function!"
```

## Local function invocation

### Sample

1. Create a **to-be-invoked** Node.js function named "FuncInvoked" in the region of **Beijing**. The content of the function is as follows:

```
'use strict';
exports.main_handler = async (event, context, callback) => {
  console.log("\n Hello World from the function being invoked\n")
  console.log(event)
  console.log(event["non-exist"])
  return event
};
```

2. In the `testNodejsSDK` directory, create a new file named `index.js`. This will serve as the **invoking** Node.js function. Enter the following example code:

```
const { SDK, LogType }  = require('tencentcloud-serverless-nodejs')
exports.main_handler = async (event, context) => {
 context.callbackWaitsForEmptyEventLoop = false
 const sdk = new SDK({
  region:'ap-beijing',
  secretId: 'AKxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxj',
  secretKey: 'WtxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxqL'
 }) // If running in a cloud function and bound to a role with SCF invocation privileges, it will default to the authentication
information in the environment variables.
 const res = await sdk.invoke({
  functionName: 'FuncInvoked',
  logType: LogType.Tail,
  data: {
   name: 'test',
   role: 'test_role'
  }
 })
 console.log(res)
 // return res
}
```

> **Note**

> SecretId and SecretKey: These refer to the ID and Key of the Cloud API. You can obtain or create the relevant keys by logging into the Access Management Console, and selecting **Access Keys** > **API Key Management**.

3. Go to the directory where the `index.js` file is located and run the following command to view the result.
   - On Linux or macOS, run the following command:

```
export NODE_ENV=development && node index.js
```

   - On Windows, run the following command:

```
set NODE_ENV=development && node index.js
```

   The output is as follows:

```
prepare to invoke a function!
{"key":"value"}
Already invoked a function!
```

# API List

## API Reference

- Init
- Invoke

### Init

You are recommended to run the `npm init` command to initialize the SDK before using it.

> **Note**
> - The `region`, `secretId`, and `secretKey` parameters can be passed in by using the initialization command.
> - After initialization is completed, the initialization configuration can be reused for future API calls.

**Parameter information:**

| Parameter | Required | Local Disk Types | Description |
|---|---|---|---|
| region | Not required | `String` | Regions |
| secretId | Not required | `String` | process.env.TENCENTCLOUD_SECRETID is used by default |
| secretKey | Not required | `String` | `process.env.TENCENTCLOUD_SECRETKEY` is used by default |
| token | Not required | `String` | `process.env.TENCENTCLOUD_SESSIONTOKEN` is used by default |

### Invoke

This API is used to invoke a function. Currently, sync invocation is supported.
**Parameter information:**

| Parameter | Required | Local Disk Types | Description |
|---|---|---|---|
| functionName | Supported | `String` | Function name |
| qualifier | Not required | `String` | Function version. Default value: $LATEST |
| data | Not required | `String` | Input parameter for function execution |
| namespace | Not required | `String` | Namespace. Default value: default |

| | | | |
|---|---|---|---|
| region | Not required | String | Regions |
| secretId | Not required | String | process.env.TENCENTCLOUD_SECRETID is used by default |
| secretKey | Not required | String | process.env.TENCENTCLOUD_SECRETKEY is used by default |
| token | Not required | String | process.env.TENCENTCLOUD_SESSIONTOKEN is used by default |

# SDK For Python

Last updated: 2023-09-27 21:25:07

## Tencentserverless SDK Overview

Tencentserverless is a Tencent Cloud SCF SDK that integrates SCF business flow APIs to make it easier to invoke SCF functions. It allows users to invoke a function quickly from a local system, CVM instance, container or function, eliminating the need to encapsulate APIs in a public cloud.

## Features

Tencentserverless SDK has the following features:

- It can invoke functions in a high-performance, low-latency manner.
- Enables quick invocation across functions after the required parameters are entered (it will obtain parameters in environment variables by default, such as `region` and `SecretId`).
- Supports access with private network domain names.
- Supports session keep-alive.
- It supports cross-region function invocation.
- Supports native invocation methods in Python.

> **Note**
> Calling SDK across functions is only applicable to event-triggered functions. HTTP-triggered functions can be invoked by requesting the corresponding path of the function in the function code.

## Quick Start

### Mutual function invocation

#### Sample

> **Note**
> - To implement mutual recursion of functions in different regions, you need to specify the region. For the naming rules, please see Region List.
> - If no region is specified, intra-region mutual function invocation will be used by default.
> - If no namespace is specified, `default` will be used by default.

1. Create a Python cloud function on the cloud, named "FuncInvoked" in the **Guangzhou** region. The function content is as follows:

```
# -*- coding: utf8 -*-

def main_handler(event, context):
 if 'key1' in event.keys():
    print("value1 = " + event['key1'])
 if 'key2' in event.keys():
    print("value2 = " + event['key2'])
 return "Hello World from the function being invoked"  #return
```

2. Create a Python cloud function on the cloud, named "PythonInvokeTest" in the **Chengdu** region. You can edit the PythonInvokeTest function in one of the following two ways, depending on your actual situation.

- Method 1. If you don't need to invoke the function frequently, you can use the following sample code:

```
from tencentserverless import scf
from tencentserverless.scf import Client
from tencentserverless.exception import TencentServerlessSDKException
```

```
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException

def main_handler(event, context):
    print("prepare to invoke a function!")
    try:
        data = scf.invoke('FuncInvoked',region="ap-guangzhou",data={"a": "b"})
        print (data)
    except TencentServerlessSDKException as e:
        print (e)
    except TencentCloudSDKException as e:
        print (e)
    except Exception as e:
        print (e)
    return "Already invoked a function!" # return
```

The output is as follows:

```
"Already invoked a function!"
```

- Method 2. If you need to invoke the function frequently, you can choose to connect and trigger it through `Client` by using the following sample code:

```
# -*- coding: utf8 -*-

from tencentserverless import scf
from tencentserverless.scf import Client
from tencentserverless.exception import TencentServerlessSDKException
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException

def main_handler(event, context):
    #scf = Client(region="ap-guangzhou") # To connect using this method, please enable the "Execution Role" feature in the function configuration, and select an execution role with function invocation permissions.
    scf = Client(secret_id="AKIxxxxxxxxxxxxxxxxxxxxxxxxggB4Sa", secret_key="3vZzxxxxxxxxxxxaeTC", region="ap-guangzhou", token="") # To connect using this method, please replace the secret_id and secret_key with your own. This pair of keys needs to include the permission to invoke the function.
    print("prepare to invoke a function!")
    try:
        data = scf.invoke('FuncInvoked',data={"a":"b"})
        data = scf.FuncInvoked(data={"a": "b"}) # To use the native Python invocation method, you need to initialize it through Client first.
        print (data)
    except TencentServerlessSDKException as e:
        print (e)
    except TencentCloudSDKException as e:
        print (e)
    except Exception as e:
        print (e)
    return "Already invoked a function!" # return
```

The output is as follows:

```
"Already invoked a function!"
```

> **Note**
> Secret_id and secret_key refer to the ID and Key of the Cloud API. You can obtain or create the relevant keys by logging into the Access Management Console, selecting **Cloud API Keys**, and then API Key Management.

## Local function invocation

### Development Preparations

- Development Environment:
  Python 2.7 or Python 3.6 should be installed.
- Runtime Environment:
  The tencentserverless SDK should be installed, and it supports Windows, Linux, and Mac operating systems.

> **Note**
> For local function invocation, you must complete the above preparations. We recommend you develop the function locally and then upload it to the cloud and use mutual function invocation for debugging.

### Installation through pip (recommended)

Run the following command to install Tencentserverless SDK for Python.

```
pip install tencentserverless
```

### Installing through source package

Download the latest source code package from the Github repository. After decompressing the source code package, run the following commands in sequence to install.

```
cd tencent-serverless-python-master
python setup.py install
```

### Configuring Tencentserverless SDK for Python

Run the following command to upgrade Tencentserverless SDK for Python.

```
pip install tencentserverless -U
```

Run the following command to view the information of Tencentserverless SDK for Python.

```
pip show tencentserverless
```

### Sample

1. Create a Python cloud function on the cloud, named "FuncInvoked" in the **Guangzhou** region. The function content is as follows:

```
# -*- coding: utf8 -*-

def main_handler(event, context):
 if 'key1' in event.keys():
    print("value1 = " + event['key1'])
 if 'key2' in event.keys():
    print("value2 = " + event['key2'])
 return "Hello World from the function being invoked"  #return
```

2. Create a local file named `PythonInvokeTest.py` with the following content:

```
# -*- coding: utf8 -*-

from tencentserverless import scf
from tencentserverless.scf import Client
from tencentserverless.exception import TencentServerlessSDKException
```

```
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException

def main_handler(event, context):
 print("prepare to invoke a function!")
 scf = Client(secret_id="AKIxxxxxxxxxxxxxxxxxxxxxxxxggB4Sa", secret_key="3vZzxxxxxxxxxxxaeTC", region="ap-
guangzhou", token="") # Replace with your own secret_id and secret_key
 try:
    data = scf.invoke('FuncInvoked',data={"a":"b"})
    # data = scf.FuncInvoked(data={"a":"b"})
    print (data)
 except TencentServerlessSDKException as e:
    print (e)
 except TencentCloudSDKException as e:
    print (e)
 except Exception as e:
    print (e)
 return "Already invoked a function!" # return

main_handler("","")
```

Go to the directory where the `PythonInvokeTest.py` file is located and run the following command to view the result.

```
python PythonInvokeTest.py
```

The output is as follows:

```
prepare to invoke a function!"Hello World form the function being invoked"
```

# API List

## API Reference

- Client (class)
- invoke (method)
- TencentserverlessSDKException (class)

### Client

**Method:**

- **__init__**

  **Parameter information:**

| Parameter | Required | Local Disk Types | Description |
| --- | --- | --- | --- |
| region | Not required | String | Region, which is the same as the region of the function invoking the API and is Guangzhou for local invocations by default. |
| secret_id | Not required | String | User `SecretId`, which is obtained from the function's environment variable by default and is required for local debugging. |
| secret_key | Not required | String | User `SecretKey`, which is obtained from the function's environment variable by default and is required for local debugging. |
| token | Not required | String | User `token`, which is obtained from the function's environment variable by default. |

- **invoke**

**Parameter information:**

| Parameter | Required | Local Disk Types | Description |
|---|---|---|---|
| function_name | Supported | String | Function name. |
| qualifier | Not required | String | Function version. Default value: $LATEST. |
| data | Not required | Object | Input parameter for function execution, which must be an object that can be processed by `json.dumps`. |
| namespace | Not required | String | Namespace. Default value: default. |

### invoke

This is used to invoke a function. Currently, only sync invocation is supported.
**Parameter information:**

| Parameter | Required | Local Disk Types | Description |
|---|---|---|---|
| region | Not required | `String` | Region, which is the same as the region of the function invoking the API and is Guangzhou for local invocations by default. |
| secret_id | Not required | `String` | User `SecretId`, which is obtained from the function's environment variable by default and is **required for local debugging**. |
| secret_key | Not required | `String` | User `SecretKey`, which is obtained from the function's environment variable by default and is **required for local debugging**. |
| token | Not required | `String` | User `token`, which is obtained from the function's environment variable by default. |
| function_name | Supported | `String` | Function name. |
| qualifier | Not required | `String` | Function version. Default value: $LATEST. |
| data | Not required | `String` | Input parameter for function execution, which must be an object that can be processed by `json.dumps`. |
| namespace | Not required | `String` | Namespace. Default value: default. |

### TencentserverlessSDKException

**Attributes:**

- **[code]**
- **[message]**
- **[request_id]**
- **[response]**
- **[stack_trace]**

**Methods and descriptions:**

| Method Name | Description |
|---|---|
| get_code | Returns error code |
| get_message | Returns error message |
| | |

| | |
|---|---|
| get_request_id | Returns `RequestId` |
| get_response | Returns `response` |
| get_stack_trace | Returns `stack_trace` |

# Third-Party Tools
# Malagu Framework Overview

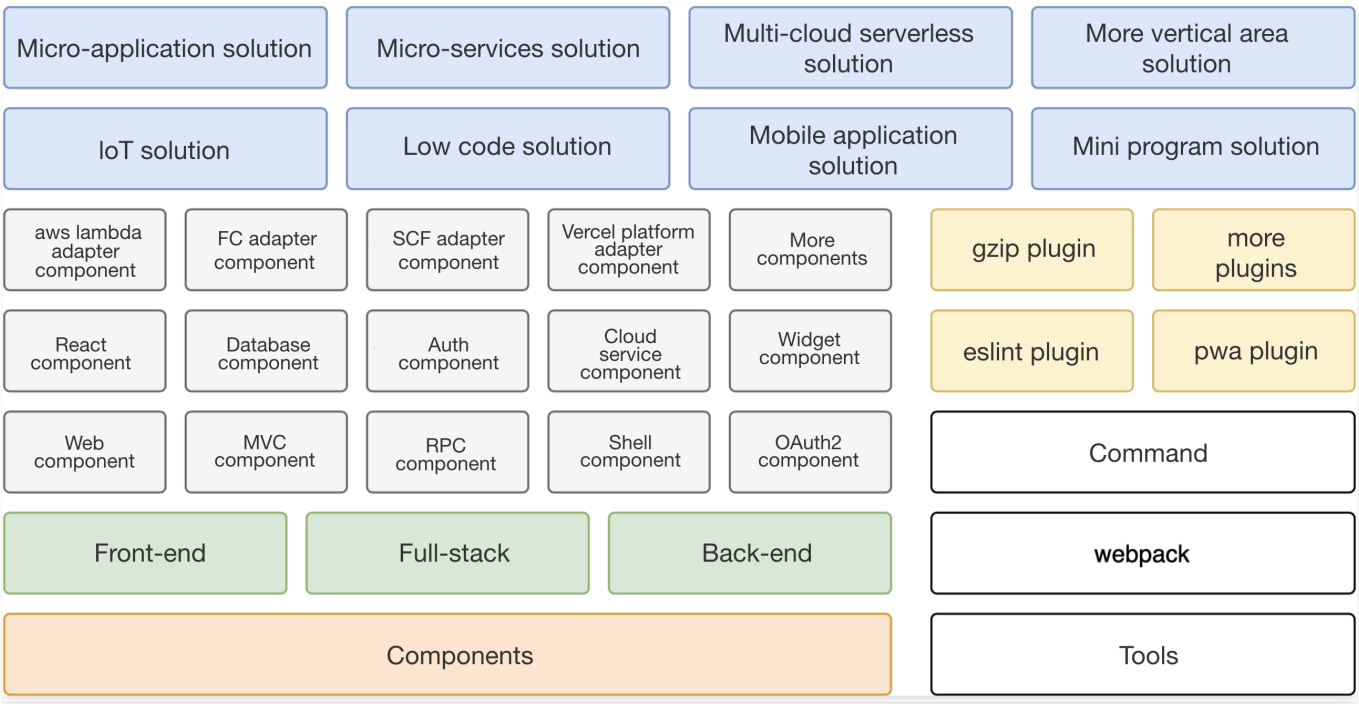
Last updated: 2023-09-27 21:27:09

**Description**
Malagu is a third-party development tool, for which Tencent Cloud cannot provide official support at the moment. If you have any questions or feedback, you are welcome to visit the Malagu Community to discuss or contribute to the community through issues.

## What is Malagu?

Aka the M framework, Malagu is a serverless-first, componentized, platform-independent progressive application framework based on TypeScript. It uses the same programming language and IoC design to develop frontend, backend, and frontend/backend integrated applications. It combines object-oriented programming (OOP), aspect-oriented programming (AOP), and other elements and draws on many design ideas of Spring Boot.
On the backend, Malagu abstracts a set of APIs to facilitate adaptation to any platforms (SCF, AWS Lambda, Vercel, etc.) and basic frameworks (Express, Koa, Fastify, etc.). It is an upper-layer framework independent of such platforms and basic frameworks.
In serverless scenarios, Malagu is used to develop projects by application. An application generally includes multiple APIs. If the application is large, it should be split into small microapplications or microservices. Just like the principle of granularity breakdown in the microservice architecture, reasonable granularity breakdown enables better application management. The framework will guarantee the execution performance of one application in one function.
For more information, please see Malagu Framework.

## Malagu Architecture Diagram




## Why Malagu?

**Firm belief that serverless is the future**

Serverless is a new-generation cloud computing engine. It is developed to replace the traditional cloud service framework. The core idea of serverless is to enable developers to focus on the business code with no need to care about servers.

### Serverless status quo

Currently, all cloud vendors and communities are vigorously promoting and preaching the Serverless concept. Through Serverless, business solutions can be implemented quickly, at a low cost and high quality. The industry generally believes that Serverless = FaaS + BaaS, and in the future, it may take other forms. Regardless of how the form changes, the core concept of Serverless remains the same. The key to the development experience of Serverless lies in the development experience of FaaS. However, the current development experience of FaaS is not ideal and has many pain points. Some of these pain points may be difficult to solve in the short term at the FaaS base level, while others may be more reasonable to solve at the tool and framework level. For example, cold start, CICD, microservices, database access, local development debugging and running, platform non-locking, etc. In the future, there will be more and more Serverless First development frameworks, not just resource orchestration operation and maintenance tools. These will be combined with Serverless First development frameworks to provide more advanced Serverless development platforms or low-code platforms.

> **How do we address these challenges?**
> We can take a different approach and try to solve these problems from the development framework level (as it turns out, these problems can be solved through the development framework). So, the new choice is whether to adopt a traditional framework or need a completely new one? If we choose to create a new framework, should we choose a specific programming language or a general-purpose one?

### Why a new framework?

While developers can generally accept the development experience of traditional frameworks that have been used for many years, applications developed with traditional frameworks often encounter various difficult problems when migrating to a Serverless environment. These problems are often closely related to the underlying design of the framework. Although some issues can be resolved or mitigated through the framework's extension capabilities, the practical result is that the threshold for framework modification is high, the effect is not ideal, and it requires hacking, which is not elegant.

When you use a traditional framework in Serverless, you may feel that although the application can run, you may have concerns when it is truly applied to the production level. Of course, with the continuous development of the underlying technology of the Serverless platform, the situation of traditional frameworks in Serverless scenarios will also be greatly improved. To achieve the best state, unilateral changes are often not enough, and the framework also needs to reasonably adapt to Serverless scenarios. Just like the frontend UI framework needs Mobile First, the browser provides responsive support, and the frontend UI framework provides relevant adaptation. Therefore, we need a brand new, Serverless First development framework to greatly leverage the advantages of Serverless and make the Serverless development experience inherit and even surpass the traditional development experience.

### Why a specific programming language?

Currently, there are many language-independent Serverless tools or frameworks in the open-source community, such as Funcraft, Serverless Cloud Framework, Vercel, etc. These general-purpose language Serverless tools can indeed provide a good experience at the operational level and can form universal standards. However, the development experience in application code development, debugging, and running may not be ideal. Each programming language has its unique aspects in development, debugging, and running, and it is difficult for general-purpose language Serverless tools to achieve a unified development experience while still being very good. Only by choosing a specific programming language can the experience of development, debugging, and running be maximized.

### Why TypeScript?

Serverless makes it much easier to get started with backend development and greatly reduces the learning costs for frontend developers to develop backend applications based on serverless. In the future, more and more frontend developers will become full-stack developers. TypeScript can be used to develop both frontend and backend applications, so it is very friendly to frontend and full-stack developers.
Frontend architecture is a kind of Serverless architecture. For instance, the frontend browser needs to load frontend code for execution, and the Serverless scenario also needs to load user code for execution. Therefore, many frontend solutions are naturally suitable for Serverless scenarios. For example, the frontend reduces code volume and decreases code deployment and

cold start time through packaging, compression, and Tree Shaking. Similarly, this optimization solution is also applicable to Serverless scenarios. Therefore, choosing TypeScript is equivalent to directly owning ready-made solutions that have been polished by countless real scenarios.

In addition, TypeScript is very close to Java, so Java developers can easily switch to the TypeScript technology stack.

### Value of Malagu

Malagu is a serverless-first, extensible, componentized progressive application framework based on TypeScript. It shields the underlying details of different serverless platforms and most of the challenges in serverless scenarios. It is developed and improved based on real business scenarios and provides solutions usable at the production level. Moreover, it offers multi-cloud vendor-independent solutions.

## How to Use Malagu

The Malagu framework consists of a series of components, each of which is a node module. You can choose the appropriate components according to your business scenario. You can also develop your own components based on the component mechanism. For the convenience of fast development, Malagu provides a command line tool that has built-in out-of-the-box templates for different use cases. You can quickly create your applications through the command line tool.

1. Run the following command to install the command line tool.

```
npm install -g @malagu/cli
```

2. Initialize a project.

```
malagu init project-name
cd project-name # Enter the root directory of the project
```

3. Run tests locally.

```
malagu serve            # Start the application, default port is 3000
```

4. Open a browser and access `http://localhost:3000/` .

# Quick Start

Last updated: 2023-09-27 21:27:34

We can deploy applications to Tencent Cloud SCF using the @malagu/scf-plugin component. Adhering to the principle of convention over configuration, it requires no initial setup and is ready to use out of the box.

## Cloud Resource

The adapter component has a default deployment rule, which can be overwritten. When running a deployment task, it will use the SDK provided by the platform to create the required cloud resource according to the deployment rule. If it finds that the cloud resource already exists, it will update the resource differentially. **It always creates or updates cloud resources in the most secure way possible**; for example, if a custom domain name is configured, it will attempt to create or update the custom domain name resource.
The adapter component deploys an application into a function, which means that one application corresponds to one function. If the application is large, it should be split into small microapplications or microservices. Just like the principle of granularity breakdown in the microservice architecture, reasonable granularity breakdown enables better application management. The framework will guarantee the execution performance of one application in one function.

## Environmental Isolation

In the Malagu framework, a configuration attribute `stage` is provided to represent the environment. In the deployment rules agreed upon by the @malagu/cloud-plugin component, the `mode` attribute maps to the `stage` attribute. By default, three environments are provided: testing, pre-release, and production. The expression rules are as follows:

```
stage: "${'test' in mode ? 'test' : 'pre' in mode ? 'pre' : 'prod' in mode ? 'prod' : cliContext.prod ? 'prod' : 'test'}" # test, pre, prod
```

The `stage` value rule is as follows:

- **Test**: Testing environment, when the `mode` attribute includes the `test` mode, or `mode` does not include `test`, `pre`, `prod`, and the command line parameter `-p,--prod` is not specified.
- **Pre**: Pre-release environment, when the `mode` attribute includes the `pre` mode.
- **Prod**: Production environment, when the `mode` attribute includes the `prod` mode, or the command line parameter `-p,--prod` is specified.

You can choose different deployment environments by specifying `mode`:

```
# Deployment to the testing environment
malagu deploy -m test # Or use malagu deploy

# Deployment to the pre-release environment, you can also skip the pre-release environment deployment and directly deploy to the production environment.
malagu deploy -m pre

# Deployment to the production environment
malagu deploy -m prod
```

## Isolation Level

**The isolation level of environments can be controlled.** You can use accounts to isolate environments by using different configuration files for different environments and configuring different accounts for different configuration files. Similarly, you can also use regions and service aliases to isolate environments. The framework isolates environments by service alias by default. The isolation methods can be used together.
Association of the `stage` attribute value with the service alias (the following is the default rule and does not need to be configured):

```
malagu:
  cloud:
```

```
  alias:
    name: ${stage}
```

Association with the API Gateway environment (the following is the default rule and does not need to be configured):

```
malagu:
  cloud:
    apiGateway:
      release:
        environmentName: "release"
```

## Deployment mode

The adapter component defines the deployment mode through the `mode` attribute. Supported deployment modes include:

- **HTTP**: Deployment mode based on custom API Gateway and HTTP-triggered Functions. During the deployment process, cloud resources such as API Gateway, namespaces, and functions are created or updated.
- **timer**: deployment mode based on timer trigger + event-triggered function. During the deployment process, cloud resources such as timer triggers, namespaces, and functions are created or updated.
- **API-Gateway**: Deployment mode based on custom API Gateway and event-triggered functions. During the deployment process, cloud resources such as API Gateway, namespaces, and functions are created or updated.
- **API-Gateway-Basic**: Deployment mode based on default API Gateway and event-triggered functions. During the deployment process, cloud resources such as the default API Gateway are created or updated. In this mode, the number of API Gateway calls is not charged.

```
mode:
  - http
```

## Custom Deployment Rule

You can overwrite the default deployment rule with a custom rule of the same name.

### Default rule

The default rule is defined in the malagu-remote.yml configuration file of the @malagu/scf-plugin component.

### Custom deployment type

```
mode:
- http # The default value is http, currently supporting http, timer, api-gateway, api-gateway-basic.
```

### Custom namespace

```
malagu:
  cloud:
    namespace:
    name: xxxx # The default value is default
```

> **Description**
> Other namespace attributes can be configured in a similar way.

### Custom function name

```
malagu:
  cloud:
    function:
      name: xxxx # The default value is ${pkg.name}
```

**Description**
Other function attributes can be configured in a similar way.

## Attribute Configuration

```
malagu:
  cloud:
    namespace:
    name:
      description:
    function:
      name: ''
      namespace:
      handler:
      publish:
      l5Enable:
        type:
      codeSource:
        description:
      memorySize:
      timeout:
      runtime:
      role:
      clsLogsetId:
      ClsTopicId:
      env:
      vpcConfig:
        vpcId:
        subnetId:
      layers:
    name:
        version:
      deadLetterConfig:
        type:
    name:
      filterType:
      publicNetConfig:
        PublicNetStatus:
          eipConfig:
            eipStatus:
    alias:
    name:
      functionName:
      namespace:
        description:
      routingConfig:
        additionalVersionWeights:
          version:
          weight:
        addtionVersionMatchs:
          version:
          key:
          method:
          expression:
    apiGateway:
      usagePlan:
    name:
      environment:
      desc:
      maxRequestNum:
      maxRequestNumPreSec:
```

```
      strategy:
    name:
      environmentName:
      strategy:
    api:
    name:
      serviceTimeout:
    protocol:
      desc:
      authType:
      enableCORS:
      businessType:
      serviceScfFunctionName:
      serviceWebsocketTransportFunctionName:
      serviceScfFunctionNamespace:
      serviceScfFunctionQualifier:
      serviceWebsocketTransportFunctionNamespace:
      serviceWebsocketTransportFunctionQualifier:
      isDebugAfterCharge:
      serviceScfIsIntegratedResponse:
      isDeleteResponseErrorCodes:
      responseSuccessExample:
      responseFailExample:
      authRelationApiId:
      userType:
      oauthConfig:
        publicKey:
        tokenLocation:
        loginRedirectUrl:
      responseErrorCodes:
        code:
        msg:
      desc:
        convertedCode:
        needConvert:
      requestConfig:
        ApiRequestConfig:
    path:
        method:
      requestParameters:
    name:
      desc:
        position:
        type:
        defaultValue:
        required:
      RequestParameter:
    service:
      exclusiveSetName:
    name:
    protocol:
      description:
      netTypes:
      ipVersion:
      setServerName:
      appIdType:
    release:
      environmentName:
      desc:
  customDomain:
    name:
    isDefaultMapping:
    certificateId:
```

```
  protocol:
  netType:
  pathMappingSet:
    path:
    Environment:
```

# Accessing Database

Last updated: 2023-09-27 21:27:57

The Malagu framework can be easily integrated with third-party database operation frameworks, such as Sequelize and TypeORM. Malagu's component mechanism increases the extensibility of third-party libraries and supports attribute configuration for out-of-the-box use.
Currently, Malagu offers integration with TypeORM libraries. You can configure the database connection information through the framework configuration file. In addition, Malagu is serverless-first, so it features best practice adaption to serverless scenarios during integration with TypeORM. In addition, it draws on the Spring transaction management mechanism to provide non-intrusive transaction management capabilities and support transaction propagation behaviors.

## How to Use

1. Install the Malagu software. For detailed installation steps, please refer to Using Malagu.
2. The framework provides a built-in template, database-app. By executing the following command, you can quickly initialize a template application for database operations.

```
malagu init demo database-app
```

3. Upon completion of the initialization, simply adjust the database connection configuration to match the current actual environment. You can also directly install the `@malagu/typeorm` component in the project by executing the following command.

```
yarn add @malagu/typeorm
# Alternatively, execute npm i @malagu/typeorm
```

## Configuring Data Source Connection

The data source connection configuration in Malagu is similar to that in TypeORM, with slightly different configuration form and location. In order to keep the configuration method for third-party libraries consistent with that for framework components, the framework adapts the original configuration method of TypeORM to that for framework components during integration with TypeORM. For more information on TypeORM data source connection configuration, please see Connection Options.

### Single data source

If the data source connection name is not set, it will be `default` by default.

```
# malagu.yml
backend:
  malagu:
    typeorm:
      ormConfig:
        - type: mysql
          host: localhost
          port: 3306
          synchronize: true
          username: root
          password: root
          database: test
```

### Multiple data sources

In order to distinguish between different data source connections, you need to set a name for each connection. There can be one and only one connection with no name set, and its name will be `default` by default. When OrmContext APIs are used, the data source connection names will be used.

```
# malagu.yml
backend:
  malagu:
    typeorm:
      ormConfig:
        - type: mysql
          host: localhost
          port: 3306
          synchronize: true
          username: root
          password: root
          database: test
        - type: mysql
            name: 'datasource2'
          host: xxxx
          port: 3306
          synchronize: true
          username: root
          password: root
          database: test
```

## Database Operation

The following sample uses the RESTful style to implement APIs.

> **Note**
> You may also implement using the RPC style, which is similar to the RESTful style.

```
import { Controller, Get, Param, Delete, Put, Post, Body } from '@malagu/mvc/lib/node';
import { Transactional, OrmContext } from '@malagu/typeorm/lib/node';
import { User } from './entity';

@Controller('users')
export class UserController {
    @Get()
    @Transactional({ readOnly: true })
    list(): Promise<User[]> {
        const repo = OrmContext.getRepository(User);
        return repo.find();
    }
    @Get(':id')
    @Transactional({ readOnly: true })
    get(@Param('id') id: number): Promise<User | undefined> {
        const repo = OrmContext.getRepository(User);
        return repo.findOne(id);
    }
    @Delete(':id')
    @Transactional()
    async reomve(@Param('id') id: number): Promise<void> {
        const repo = OrmContext.getRepository(User);
        await repo.delete(id);
    }
    @Put()
    @Transactional()
    async modify(@Body() user: User): Promise<void> {
        const repo = OrmContext.getRepository(User);
        await repo.update(user.id, user);
    }
    @Post()
```

```
    @Transactional()
    create(@Body() user: User): Promise<User> {
        const repo = OrmContext.getRepository(User);
        return repo.save(user);
    }
}
```

## Database Context

In Malagu, TypeORM's transactions are managed by the framework, which provides the `@Transactional` decorator for how the framework initiates, propagates, commits, and rolls back transactions before and after execution methods. Plus, the framework puts the managed EntityManager objects in the database context for easy use by the business code. In addition, you can also manually manage database transactions and create EntityManager objects.
The database context is implemented based on the request context, so it is also at the request level. It mainly provides methods to get EntityManager and Repository objects:

```
export namespace OrmContext {
    export function getEntityManager(name = DEFAULT_CONNECTION_NAME): EntityManager {
        ...
    }
    export function getRepository<Entity>(target: ObjectType<Entity>|EntitySchema<Entity>|string, name?: string):
Repository<Entity>  {
            ...
    }
    export function getTreeRepository<Entity>(target: ObjectType<Entity>|EntitySchema<Entity>|string, name?: string):
TreeRepository<Entity>  {
            ...
    }
    export function getMongoRepository<Entity>(target: ObjectType<Entity>|EntitySchema<Entity>|string, name?: string):
MongoRepository<Entity>  {
        ...
    }
    export function getCustomRepository<T>(customRepository: ObjectType<T>, name?: string): T {
        ...
    }
    export function pushEntityManager(name: string, entityManager: EntityManager): void {
        ...
    }
    export function popEntityManager(name: string): EntityManager | undefined {
        ...
    }
}
```

## Transaction Management

Malagu provides the `@Transactional` decorator to define the behaviors of transactions in a declarative manner. It decides the opening, propagation, commit, and rollback behaviors of transactions according to the decorator's declaration.

### @Transactional

The `@Transactional` decorator can be added to both classes and methods. If both are added, the final configuration uses the method's configuration to merge the class, with the method's configuration taking precedence. The decorator configuration options are as follows:

```
export interface TransactionalOption {
    name?: string;              // In the case of multiple data source connections, specify the data source connection name, default
is 'default'.
    isolation?: IsolationLevel; // Database isolation level
    propagation?: Propagation;  // The propagation behavior of the transaction, supports 'Required' and 'RequiresNew', default
is 'Required'.
    readOnly?: boolean;         // Read-only, does not initiate a transaction, default is to initiate a transaction.
```

```
}
```

Sample:

```
@Put()
@Transactional()
async modify(@Body() user: User): Promise<void> {
  const repo = OrmContext.getRepository(User);
    await repo.update(user.id, user);
}
```

## @Transactional and OrmContext

The Malagu framework initiates a transaction (or possibly does not initiate) prior to method invocation based on the decorator's configuration, and delegates the EntityManager to the OrmContext context. The EntityManager, which has had a transaction initiated by the framework, is obtained through the OrmContext, where the Repository is created through the delegated EntityManager. To correctly obtain the EntityManager, please ensure that the name configured in the decorator is consistent with the name of the EntityManager to be obtained through the OrmContext. If no name is specified, the default is 'default'.
After the method is executed, the framework automatically determines whether to commit or roll back the transaction according to the method execution. If the method execution is exceptional, the transaction will be rolled back; otherwise, it will be committed.
If the method has nested invocations to another method with the `@Transactional` decorator, the configuration of transaction propagation behavior determines whether to reuse the transaction of the upper-layer method or start a new one.

## Database query

In most cases, database queries do not require the initiation of a transaction. However, it is recommended to add the `@Transactional` decorator to the method and set the readonly configuration to true. This allows the framework to create an EntityManager that does not initiate a transaction, thereby maintaining a uniform code style. An example is as follows:

```
@Get()
@Transactional({ readOnly: true })
list(): Promise<User[]> {
  const repo = OrmContext.getRepository(User);
    return repo.find();
}
```

## Transaction propagation behavior

Transaction propagation behaviors determine how transactions are propagated between different methods that require transactions. Currently, two transaction propagation behaviors are supported:

```
export enum Propagation {
    Required, RequiresNew
}
```

- **Required**: a transaction needs to be started. If the upper-layer method has already started one, it will be reused; otherwise, a new one will be started.
- **RequiresNew**: no matter whether the upper-layer method has started a transaction, a new transaction will be started.

> **Note**
> When a transaction is propagated in different methods, please make sure that the methods are invoked synchronously. Below is a sample:

```
...
@Transactional()
async foo(): Promise<void> {
  ...
```

```
  await bar();  // await must be added
}
....

...
@Transactional()
async bar(): Promise<void> {
  ...
}
```

## Binding Entity Class

The framework provides the `autoBindEntities` method for binding entity classes, which is generally invoked in the module entry file and contains the following two parameters:

- **entities**: entity class you defined.
- **name**: The data source you wish to bind with the entity class. By default, it binds with 'default'.

```
export function autoBindEntities(entities: any, name = DEFAULT_CONNECTION_NAME) {
}
```

Sample:

```
import { autoBindEntities } from '@malagu/typeorm';
import * as entities from './entity';
import { autoBind } from '@malagu/core';

autoBindEntities(entities);
export default autoBind();
```

## Category

| Tool | Description |
| --- | --- |
| DEFAULT_CONNECTION_NAME | The default database connection name is `default`. |
| autoBindEntities | Binds entity class. |