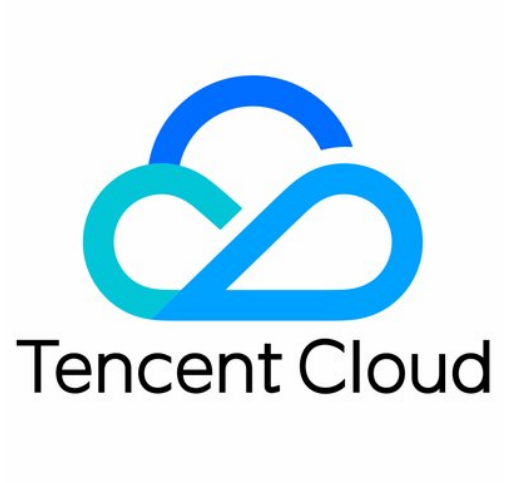
Serverless Cloud Function

Developer Tools

Product Documentation



Contents

Developer Tools
Serverless Web IDE

Last updated：2023-02-01 17:37:37

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

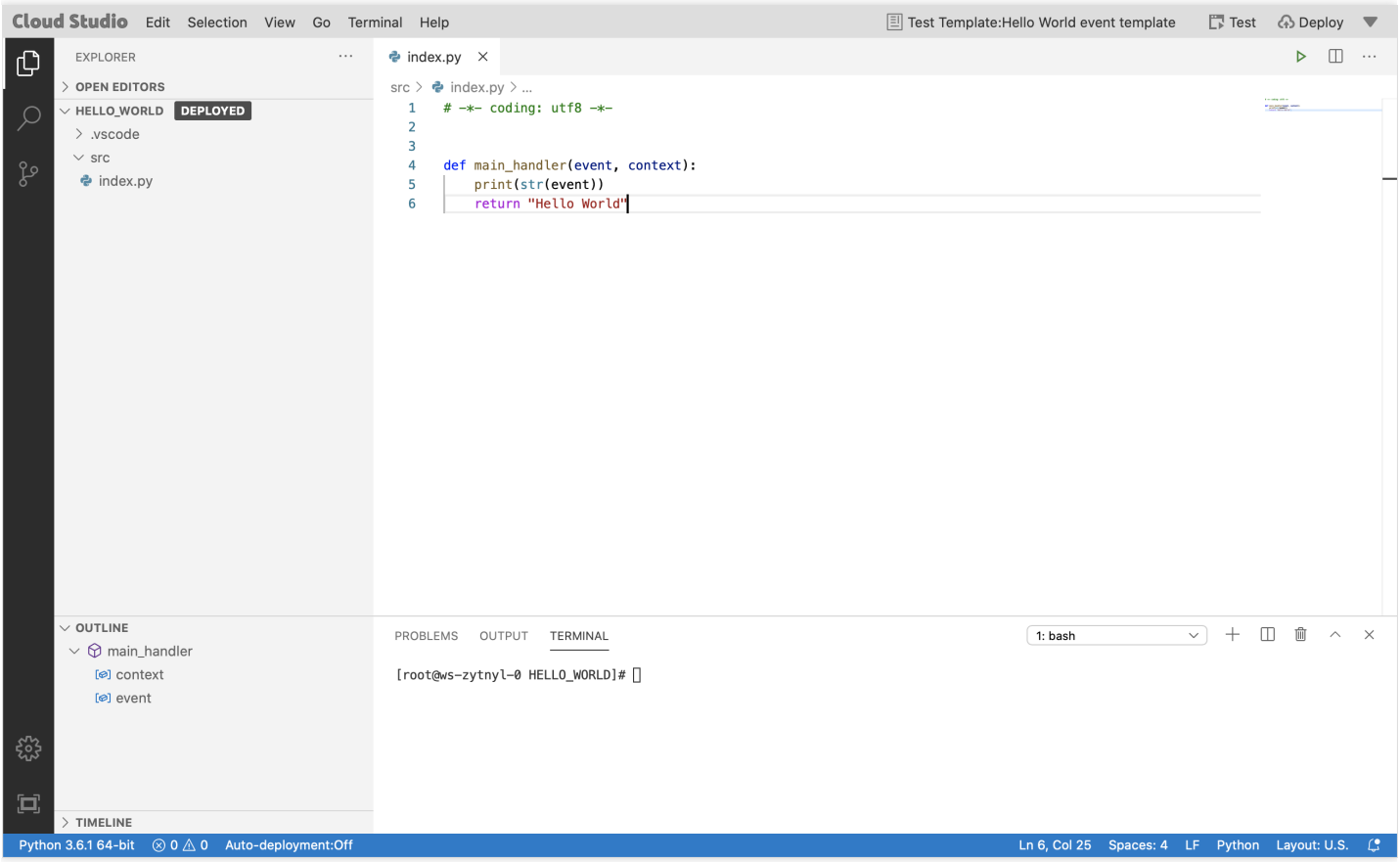
1. Log in to the SCF console and select **Functions** on the left sidebar.
2. In the function list, click a function name to enter the function details page.
3. On the **Function Management** page, select **Function Code** > **Online editing** to view and edit the function.

Note：
Online editing is not supported for Java and Go runtime environments currently. When they are used, only developed, compiled, and packaged ZIP packages or binary files can be uploaded. The SCF environment does

not provide Java and Go compiling capability. For more information, see Go Deployment Methods and Java Deployment Methods.

Overview

This document describes the Serverless Web IDE tool in detail as shown below in order from left to right:



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

Function Operations

In Serverless Web IDE, you can edit, deploy, and test function code. Common operations such as function testing and deployment and testing template selection are configured in the operation section in the top-right corner of the IDE as

shown below:

Function deployment

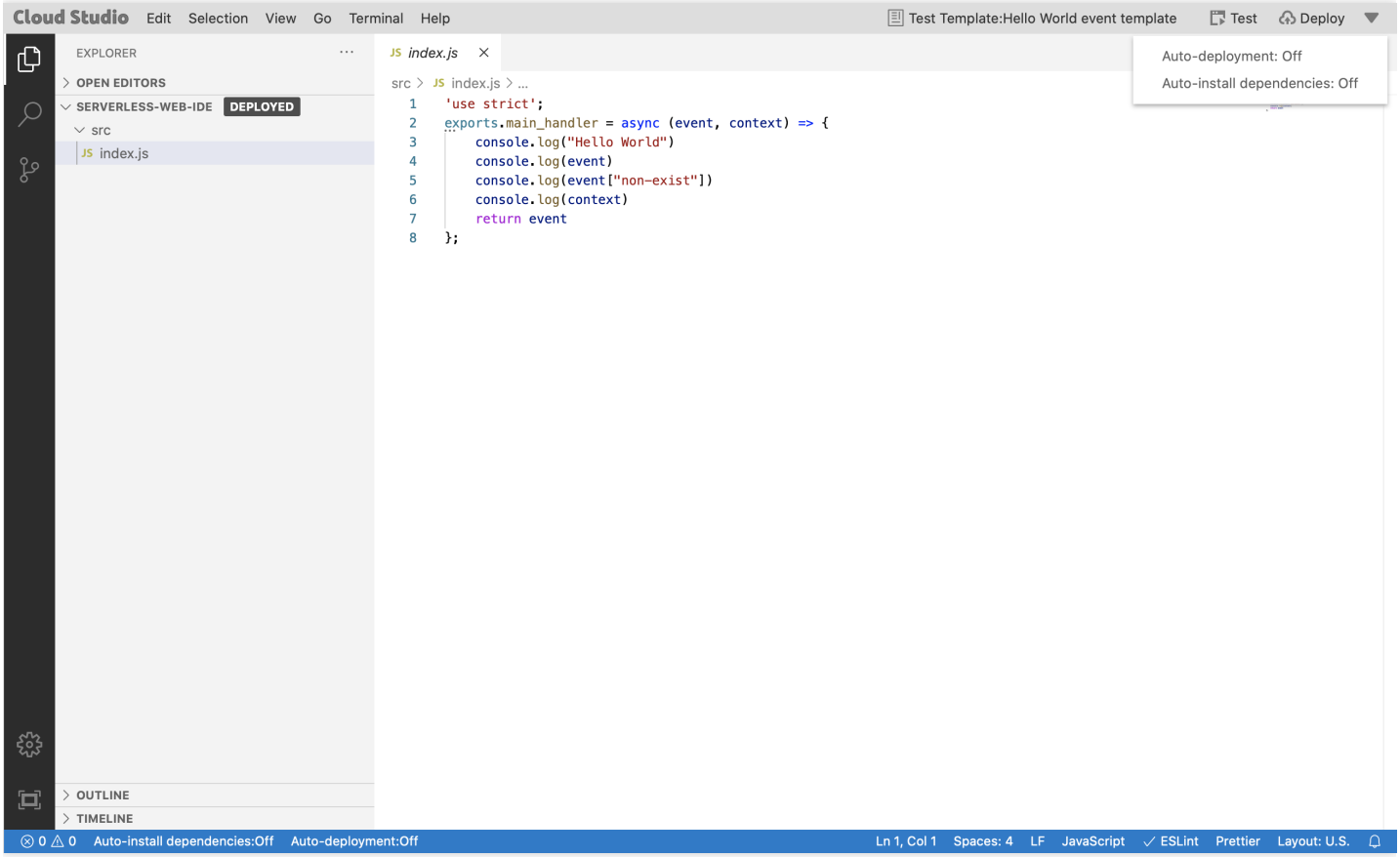
Serverless Web IDE enables you to deploy a function either manually or automatically and to install dependencies online.

- **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, this feature is supported only for the Node.js runtime environment. After online dependency installation is enabled, dependencies will be installed automatically according to the configuration in `package.json` when the function is deployed. For more information, see 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 manual and **automatic deployment** and enable/disable online dependency installation by selecting from the drop-down list in the operation section in the top-right corner of the IDE. **Automatic deployment:**

Disabled indicates the manual deployment mode.

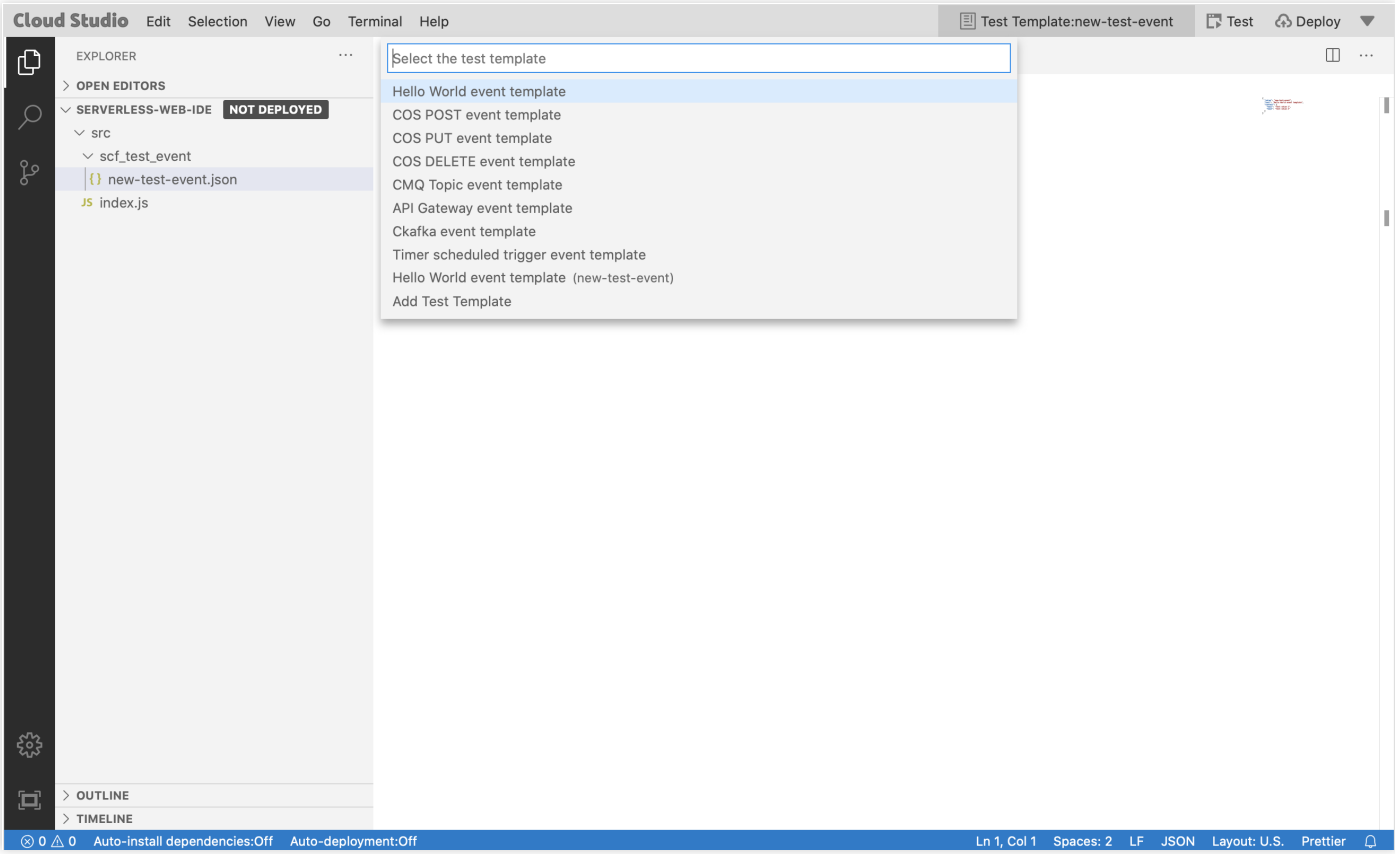


Function testing

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.

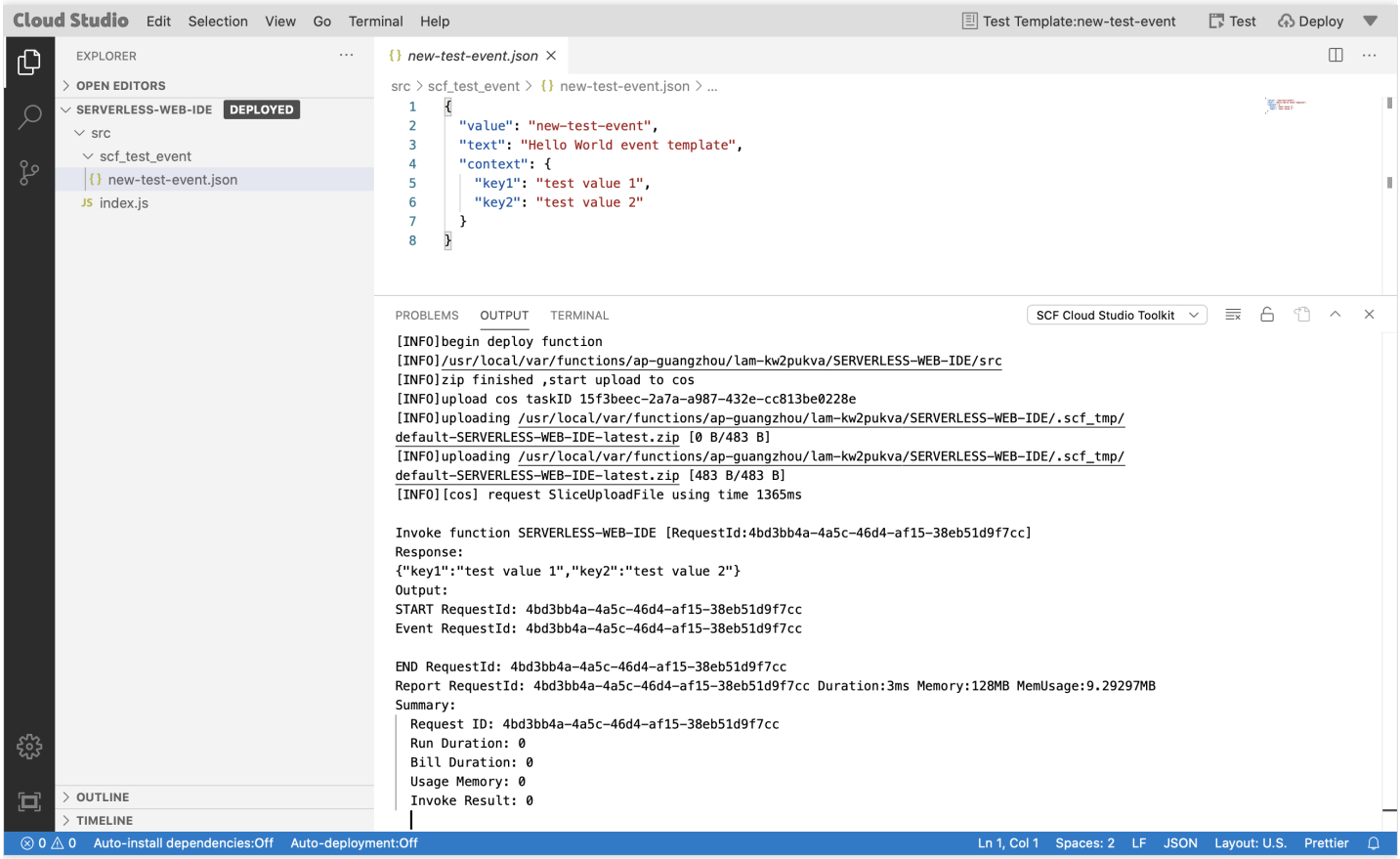
- **Select a test template**: Click **Test templates** in the operation section of the IDE to select a function testing triggering event.
- **Create a test template**: If existing test templates cannot meet your testing requirements, you can select **Create test template** from the test template drop-down list to customize a test event, which will be stored in JSON format in the scf_test_event folder in the `/src` root directory of the function and deployed to the cloud with the function

as shown below:



Viewing logs

You can view the function testing result in the output, including the returned data `Response` , the log `Output` , and the function execution summary `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:

- **Generating** `serverless.yml` : You can write the current configuration of the function into the `serverless.yml` configuration file and use the Serverless Cloud Framework command line tool for further development;
- **Discarding current modifications**: You can re-pull the function deployed in the cloud to overwrite the current workspace.

IDE Operations

The commonly used commands, runtime environments, and preconfigured extensions in Serverless Web IDE are as follows:

Commands

Command	Version

Command	Version
python3	Python 3.7.12 `python3` follows the latest Python 3 version by default.
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 follows the latest Node.js version by default. Node.js 14.18, 12.16, and 10.15 are also installed in the environment. You can run the `n` command in the terminal to switch the version.
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
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

Tool	Version
Zip and 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 environments

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, and 5.6

Extensions

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

Quota Limits

- The IDE provides 5 GB of storage space for each user. If it is used up, you will not be able to perform write operations; therefore, clean it up in time. (Deleting functions will not clear the storage space of the IDE. You can back up your workspace changes and manually **reset the workspace**. You can also choose to switch to the old editor to avoid this restriction.)
- To ensure a smooth experience, we recommend you not open more than 3 functions on multiple browser pages at the same time.

Notes

Performing the following operations in the IDE may cause security risks such as data leakage. If you have to perform them, do so with caution:

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

FAQs

1. What should I do if an exception occurs during IDE loading?

If IDE cannot be normally started due to a workspace exception, you can click **Having problems?** in the top-right corner of the IDE and click **Reset Workspace** on the pop-up page to initialize the workspace.

2. What should I do if the function can be executed successfully in the terminal but fails to be executed after I click Test?

The online IDE terminal and the SCF cloud runtime environment are independent of each other. The execution result returned after you click **Test** is the actual execution result of the function.

3. What should I do if the result doesn't meet the expectation when I run a command in the terminal?

If you run a dependent package installation command, make sure that you run it under the `src` directory.

Check the command version before running a command. For the commands supported by default, see Common commands.

Serverless Cloud Framework Overview

Last updated：2022-10-20 14:41:07

Overview

Well-received in the industry, Serverless Cloud Framework allows you to deploy a complete and available serverless application framework without having to care about underlying resources. It features resource orchestrating, auto scaling, and event driving and covers the full development lifecycle from coding and debugging to testing and deploying, helping you quickly build serverless applications with the aid of Tencent Cloud resources.

SCF Component

Serverless Cloud Framework provides an SCF component, which can be used to quickly package and deploy SCF projects. You can familiarize yourself with and use the component by following the steps below:

1. Get started with Serverless Cloud Framework as instructed in Creating and Deploying Functions.
2. Learn how to use Serverless Cloud Framework to develop and debug SCF functions as instructed in Development Mode and In-cloud Debugging.
3. Learn how to perform project management and resource orchestration for multiple SCF functions as instructed in Project Application.

Best Practices

Serverless Cloud Framework provides the SCF component to implement resource creation and orchestration for SCF. In addition, it provides more encapsulated components and best practices for some typical use cases, such as Express framework support and website deployment. For more information, please see the Serverless Components project on GitHub.

Item	Description
Deploying static websites	Use the `Serverless Website` component to quickly host a static website.
Deploying Express.js applications	Use the `Serverless SCF` component to quickly construct an Express.js project.

Deploying full-stack websites with Vue + Express + PostgreSQL	Use Vue as the frontend and Express framework as the backend to deploy a serverless full-stack application through multiple Serverless Components.
Deploying Nuxt.js applications	Use the `Serverless Components Nuxt.js` component to quickly deploy an SSR project based on Nuxt.js.

Installation

Last updated：2023-10-30 10:35:22

You can install Serverless Cloud Framework through npm.

Installing via 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
```

Note：
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：2022-12-19 18:41:57

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

Prerequisites

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 by running `scf deploy` , you can scan the QR code for quick authorization and deployment. After you authorize by scanning the code, temporary key information will be generated (which will expire in 60 minutes) and 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 on the permissions obtained during quick authorization, see scf_QcsRole 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, see Sub-account Permission Configuration.

Authorizing with local key

To eliminate the need for repeated authorization due to information expiration in case of authorization by scanning the code, you can authorize with a key. Create an `.env` file in the root directory of the project to be deployed and configure the Tencent Cloud `SecretId` and `SecretKey` information:

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

You can get `SecretId` and `SecretKey` on the Manage API Key page.

> 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, see Sub-account Permission Configuration.

Configuring with permanent key

You can run the `scf credentials` command to quickly set the persistent storage of the global key information. This command must be configured under the created SCF project. Make sure that you have created a 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's `secretId`
--secretKey / -k (Required) Tencent Cloud CAM account's `secretKey`
--profile / -n {name} Authorization name, which is `default` by default
--overwrite / -o Overwrite the key with 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 through the specified 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 --o
verwrite
```

- **Delete global authorization information:**

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

- **View all current authorization information:**

```
$ scf credentials list
```

- **Deploy through global authorization information:**

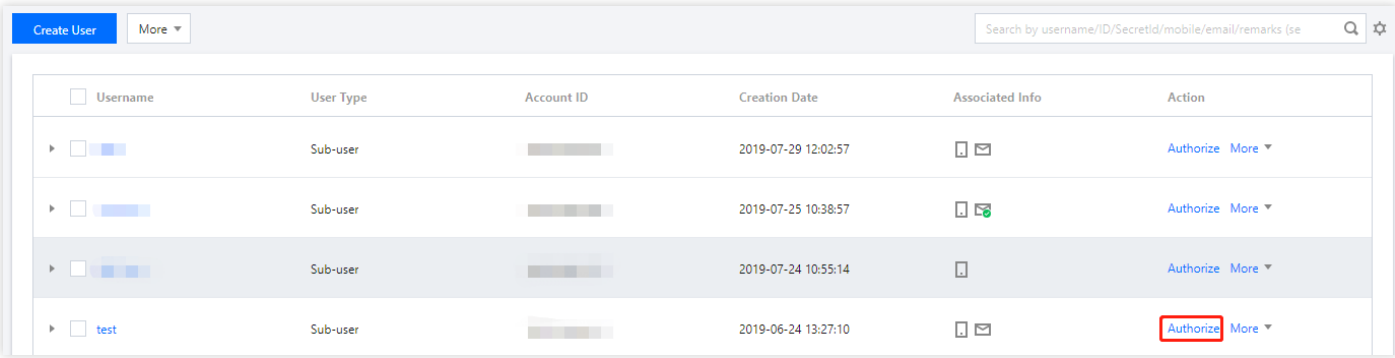
```
# Deploy through the default profile
$ scf deploy
# Deploy through the specified profile
$ scf deploy --profile newP
# Ignore global variables and scan the QR code for deployment
$ scf deploy --login
```

Sub-account Permission Configuration

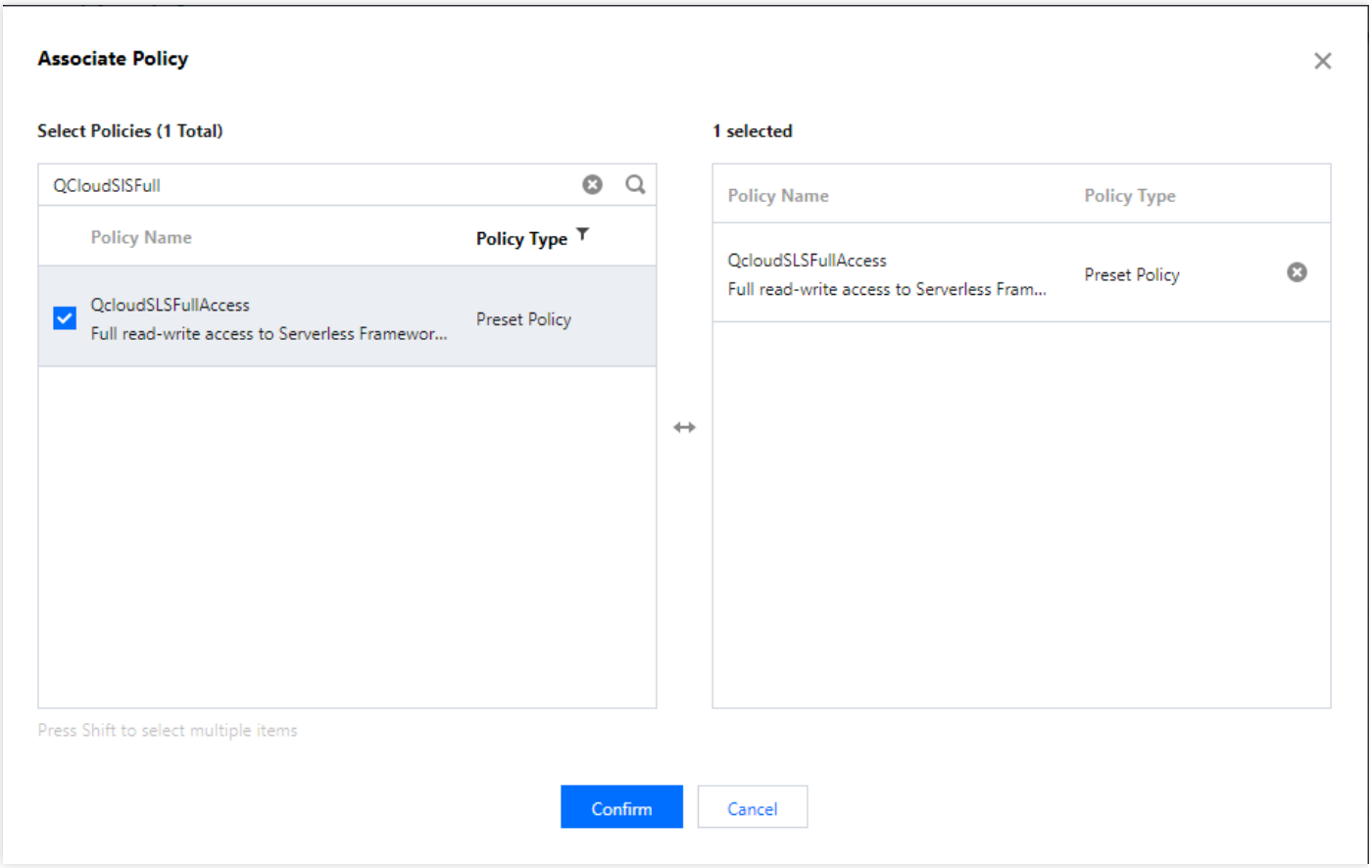
Configuration steps

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:

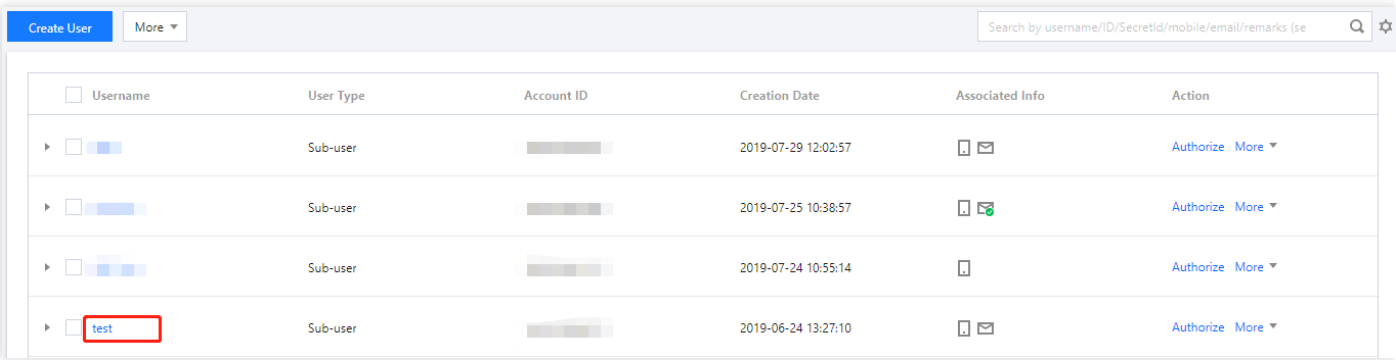
1. On the Users page in the CAM console, select the target sub-account and click **Authorize**.



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

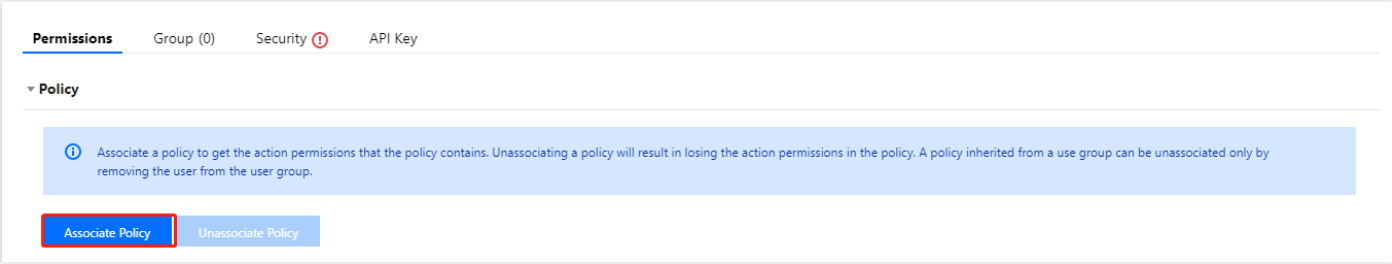


3. On the Users page in the CAM console, select the target sub-account and click the username to enter the user details page.

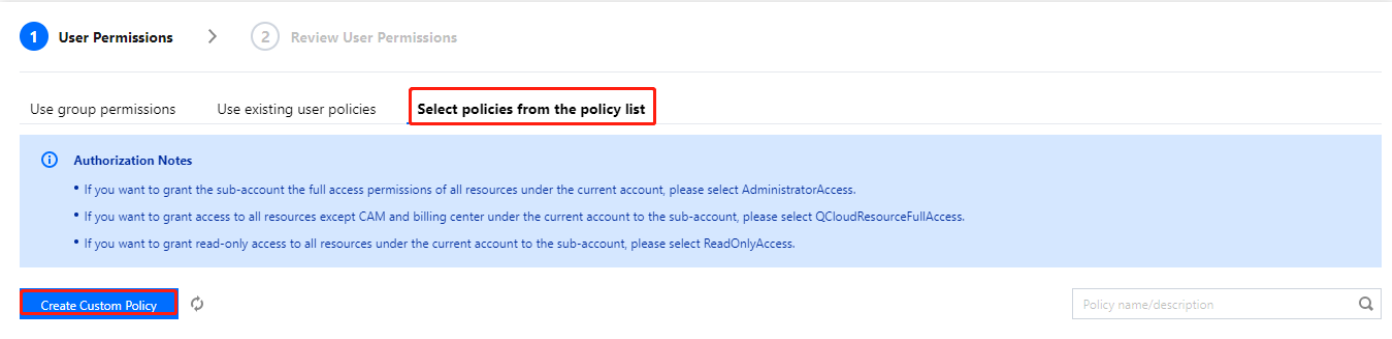


4. Click **Associate Policy**. On the policy adding page, click **Select policies from the policy list** > **Create Custom Policy**.

   Policy association page:



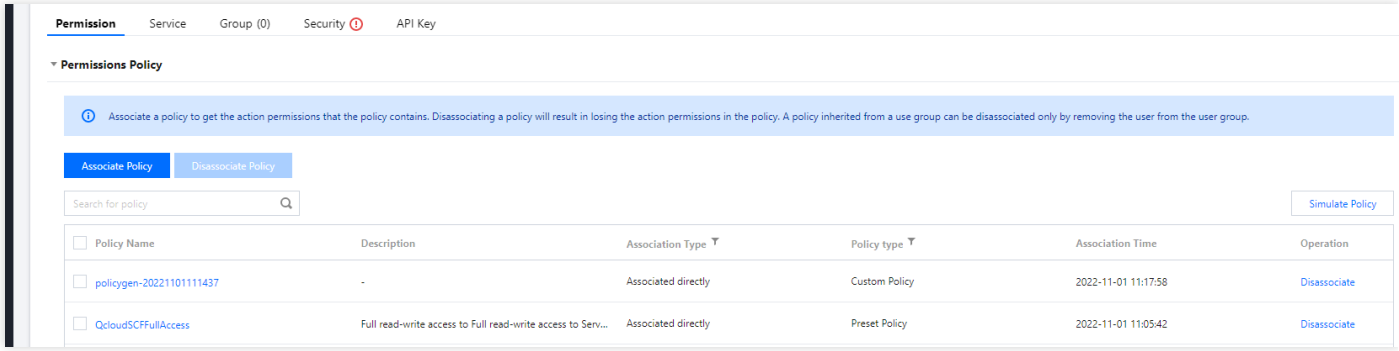
Policy creation page:



5. Click **Create by Policy Syntax** > **Blank Template** and enter the following content. Be 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, go back to the authorization page in step 4, search for the custom policy just created, and click **Next** > **OK** to grant the sub-account the operation permissions of `scf_QcsRole`. At this point, your sub-account should have a custom policy and a preset policy **QcloudscfFullAccess** and can use Serverless Framework normally.



> Note：
> In addition to the permission to call the default `scf_QcsRole` role, you can also grant the sub-account the permission to call a custom role and control the sub-account permissions with refined permission policies in the custom role. For more information, see Configuring Role for Specified Operation.

scf_QcsRole permission list

Policy	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 Cloud Monitor
QcloudslsFullAccess	Full access to SLS.
QcloudCDNFullAccess	Full access to CDN

Policy	Description
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：2022-11-01 10:58:16

> 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.

Overview

This document describes how to quickly create, configure, and deploy an SCF application in Tencent Cloud through Serverless Cloud Framework.

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.

Directions

Quick deployment

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

```
serverless-cloud-framework
```

Next, follow the interactive prompts to initialize the project. 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 cre
ate one? Yes
serverless-cloud-framework: Select the Serverless application you want to create:
scf-starter - quickly deploys an SCF function
react-starter - quickly deploys a React.js application
restful-api - quickly deploys a RESTful API to use Python + API Gateway
❯ scf-starter - quickly deploys an SCF function
vue-starter - quickly deploys a basic Vue.js application
website-starter - quickly deploys a static website
eggjs-starter - quickly deploys a basic Egg.js application
express-starter - quickly deploys a basic Express.js application

serverless-cloud-framework: Select the runtime of the application: scf-nodejs - q
uickly deploys an SCF function in Node.js
scf-golang - quickly deploys an SCF function in Go
❯ scf-nodejs - quickly deploys an SCF function in Node.js
scf-php - quickly deploys an SCF function in PHP
scf-python - quickly 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!
```

Select **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? Y
es
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 empty 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 deployment information

If you want to check the deployment status and resources of the application again, you can go to the folder where the project is successfully deployed and run the following command to view the corresponding information:

```
cd demo # Enter the project directory. 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 implements the quick configuration of the basic function information. All the configuration items supported by the SCF console can be configured in the `.yml` file (for more information, see SCF Configuration Information).
- `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：
> If you want to view the details during the removal process, you can add the `--debug` parameter.

Continuous development

After the deployment is completed, Serverless Cloud Framework supports running different commands to help you implement continuous development, deployment, 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.

FAQs

What should I do if the wizard does not pop up by default when `serverless-cloud-framework` is entered?

Solution: Add the `SERVERLESS_PLATFORM_VENDOR=tencent` configuration item 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 `GLOBAL_ACCELERATOR_NA=true` configuration item 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：2022-10-20 15:33:47

Development Mode

Serverless Cloud Framework supports the development mode (`dev` mode). For projects in development mode, you can write their code and develop and debug them more easily, as you can continuously focus on the process from development to debugging while minimizing the interruptions caused by other tasks such as packaging and update.

Entering development mode

Under a project, you can run `scf dev` to enter the development mode as shown below:

> Currently, `scf dev` is supported by only the Node.js 10 & 12.16 runtime environment.

```
$ scf dev
serverless-cloud-framework
Dev Mode - Watching your Component for changes and enabling streaming logs, if su
pported...
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 fo
r 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
```

After you enter the development mode, the Serverless tool will output the deployed content and start continuous file monitoring. When a code file is updated, it will be automatically deployed again to sync the local file to the cloud.

Exiting development mode

You can press `Ctrl+C` to exit the development mode, and the following result will be returned:

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

In-cloud Debugging

You can enable in-cloud debugging for projects whose runtime environment is Node.js 10. You can use a debugging tool such as Chrome DevTools or VS Code Debugger to connect to the remote environment for debugging.

Notes

SCF in-cloud debugging is currently in beta test. You are recommended to try it out and share your questions and suggestions with us.
Before using SCF in-cloud debugging, you need to note the following:

- In-cloud debugging uses an actually running SCF instance for debugging.
- Because of the randomness of event triggering, if there are multiple instances, an event may be triggered on a random instance. Therefore, not all requests can hit the debugging instance and trigger debugging.
- When debugging is paused at a breakpoint:
  - If it stops running for a long period of time and there is no return, the trigger such as API gateway may prompt timeout.
  - If the instance is still in countdown status and continues running until the execution is completed after debugging completion, the total consumed time will be recorded as the function execution duration.
- The maximum duration of a single execution from triggering of instance execution to debugging completion is 900 seconds. If the debugging is interrupted for over 900 seconds, the execution will be forcibly ended, and 900 seconds will be used as the function execution duration for statistics and measurement.
- The debugging capability on the current version will set the function timeout period to 900 seconds. If you exit debugging properly, the timeout period will be reset to a normal value. If you forcibly end debugging or exit debugging exceptionally, the function timeout period will fail to be set to a normal value. In this case, you can deploy the function again (on the CLI) or manually edit it (in the console) to adjust the timeout configuration.

Enabling in-cloud debugging

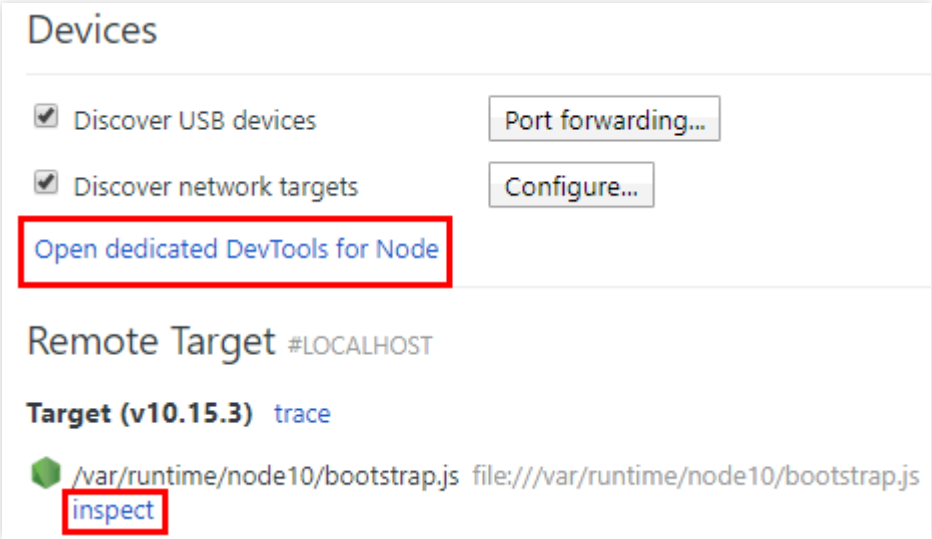
When you enter the development mode, if the project is a function whose runtime environment is Node.js 10 or above, in-cloud debugging will be automatically enabled and debugging information will be output.
For example, when you enable the development mode, if the output result contains information similar to the following content, in-cloud debugging has been enabled for this project.

```
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 fo
r Node] to debug your code.
```

Using Chrome DevTools

The following steps are used as an example to describe how to use DevTools in Chrome to connect to a remote environment for debugging:

1. Start the Chrome browser.
2. Enter `chrome://inspect/` in the address bar to access it.
3. You can open DevTools in two ways as shown below:



4. (Recommended) Click **Open dedicated DevTools for Node** under "Devices".
5. Select **inspect** under a specific target in "Remote Target #LOCALHOST".
   If you cannot open the target or there are no targets, please check whether configuration of `localhost:9229` or `localhost:9222` exists in "Configure" under "Devices", which corresponds to the output after in-cloud debugging is enabled.
6. In DevTools opened after you click **Open dedicated DevTools for Node**, you can click the **Sources** tab to view the remote code. The actual code of the function is in the `/var/user/` directory.
   On the **Sources** tab, the code that you want to view may be loaded. More remote files will be displayed as the debugging proceeds.
7. Open a file as needed and set a breakpoint at the specified position in it.
8. If you trigger the function in any means such as URL access, page, command, or API, the remote environment will start running and be interrupted at the breakpoint to wait for further operations.

9. On the tool bar on the right of DevTools, you can continue the execution of an interrupted program or perform other operations such as step-over, step-into, and step-out on it. You can also directly view the current variables or set the variables that you want to track. For more information on how to use DevTools, please see the DevTools user guide.

Exiting in-cloud debugging

When you exit the development mode, in-cloud debugging will be disabled automatically.

List of Supported Commands

Last updated：2023-05-04 18:11:11

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, from the component registry, a template specified by entering the template name after `init` , 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` : Switches 10% of the traffic to the `$latest` function version during deployment and switches the rest of the traffic 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 `serverless-cloud-framework` .

`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 in to the Tencent Cloud account and authorizing operations on associated resources by using the `login` command.

Account and Permission Configuration

Last updated：2021-12-06 14:32:22

Currently, Serverless Framework can deploy a project properly only with the relevant role permissions in the `SLS_QcsRole` role under the account. This role contains the policies for products that are used in deployment with Serverless Framework. You can configure the permissions for a root account or sub-account.

Root Account Permission Configuration

Currently, you can grant permissions by configuring the account key. As the root account has the permissions to create roles and bind policies, you can associate it with `SLS_QcsRole` for Serverless Framework access in the following way:

Authorization through account key configuration

If you want to configure persistent environment variables/key information so that you do not need to deploy them by scanning the code every time, you can create a `.env` file under the project directory and save the `SecretId` and `SecretKey` information.

```
# .env
TENCENT_SECRET_ID=123 // Your `SecretId`
TENCENT_SECRET_KEY=123 // Your `SecretKey`
```

Serverless Framework will check whether the user is in Mainland China by default during deployment. If your development environment is outside Mainland China and you want to use Serverless Framework in the Mainland China edition, you can add the following configuration in the `.env` file to start the Mainland China edition by default, which provides an interactive quick deployment process (for more information, please see Getting Started).

```
# .env
TENCENT_SECRET_ID=123
TENCENT_SECRET_KEY=123
SERVERLESS_PLATFORM_VENDOR=tencent
```

Note：

- If you don't have a Tencent Cloud account yet, please sign up first.
- If you already have a Tencent Cloud account, you can get `SecretId` and `SecretKey` in API Key Management.

Sub-account Permission Configuration

If you want to grant a sub-account the permission to deploy by scanning code, you need to ensure that the sub-account has permissions to create roles and bind role policies. You can add the preset policy `QcloudCamRoleFullAccess` or `QcloudCamSubaccountsAuthorizeRoleFullAccess` to the sub-account.

You can also add `SLS_QcsRole` by using the root account in the CAM Console to grant access to Serverless Framework resources. The role entity is `sls.cloud.tencent.com`, which includes the following policy permissions:

- QcloudCDNFullAccess
- QcloudTCBFullAccess
- QcloudSLSFullAccess
- QcloudSSLFullAccess
- QcloudCKafkaFullAccess
- QcloudMonitorFullAccess
- QcloudVPCFullAccess
- QcloudCOSFullAccess
- QcloudAPIGWFullAccess
- QcloudSCFFullAccess

After the creation is successful, the root account needs to bind the following two policies to the sub-account:

1. Call permission policy of a specified role
2. API permission policy of Serverless Framework

Granting sub-account permission to call specified role

1. On the CAM User List page, select the target sub-account and click the username to enter the user details page.
2. Click **Associate Policy**. On the policy adding page, click **Select policies from the policy list**.
3. Click **Create Custom Policy** > **Create by Policy Syntax** > **Blank Template** and enter the following content. Be sure to replace the role parameter with your own `uin` (account ID):

```
{
"version": "2.0",
"statement": [
{
"action": [
"cam:PassRole"
```

```
],
"resource": [
"qcs::cam::uin/000000000000:roleName/SLS_QcsRole"
],
"effect": "allow"
}
]
}
```

4. Click **OK** to grant the sub-account the permission to manipulate SLS_QcsRole.

Granting sub-account permission to use APIs of Serverless Framework

Two authorization methods are provided below for your reference:

Method 1. Grant the sub-account permission to manipulate all Serverless Framework resources

1. On the CAM User List page, select the target sub-account and click the username to enter the user details page.
2. Click **Associate Policy**. On the policy adding page, click **Select policies from the policy list**.
3. Search for and associate with `QcloudSLSFullAccess` and click **Next**.
4. Click **OK** to grant the sub-account the permission to manipulate all Serverless Framework resources.

   The policy syntax is as follows:

```
{
"version": "2.0",
"statement": [
{
"action": [
"sls:*"
],
"resource": "*",
"effect": "allow"
}
]
}
```

Method 2. Grant the sub-account permission to manipulate specific Serverless Framework resources

You can allow a sub-account to manipulate only specific Serverless Framework resources in the following steps:

1. On the CAM User List page, select the target sub-account and click the username to enter the user details page.
2. Click **Associate Policy**. On the policy adding page, click **Select policies from the policy list**.
3. Click **Create Custom Policy**, create a custom policy based on the policy syntax, and associate it to the user. The sample policy syntax is as shown below:

```
{
"version": "2.0",
"statement": [
{
"action": [
"sls:*"
],
"resource": "qcs::sls:ap-guangzhou::appname/${appname}/stagename/${stagename}",
"effect": "allow"
}
]
}
```

After the configuration is completed, the sub-account will have the permission to manipulate serverless applications only under `${appname}` and `${stagename}` .

Creating and Deploying Function

Last updated：2021-12-06 14:32:22

Operation Scenarios

This document describes how to use the SCF component provided by Serverless Framework to quickly create and deploy an SCF project.

Prerequisites

You have installed Serverless Framework as instructed in Installing Serverless Framework.

Directions

Creating function directory

1. Run the following command on the command line to create a directory and enter it (this document uses `tencent-scf` as an example):

```
mkdir tencent-scf && cd tencent-scf
```

2. Run the following commands in sequence to quickly create an SCF application:

```
serverless create --template-url https://github.com/serverless-components/tencent-scf/tree/v2/example
```

```
cd example
```

After the application is created successfully, its directory structure is as follows

```
|- src
| └── index.py
└── serverless.yml
```

Deploying function

1. Enter the directory where `serverless.yml` is and run the following command to deploy the function:

```
serverless deploy
```

2. Log in to your Tencent Cloud account and grant applicable permissions. If you want to configure persistent environment variables or key information, please do so as instructed in Account Configuration.
After the function is deployed successfully, you can view the URL provided by the gateway trigger of the corresponding function in the command line output and access the URL in a browser to view the function deployment result.

> If you want to view more information on the deployment process, you can run the `sls deploy --debug` command to view the real-time log information during the deployment process (`sls` is an abbreviation for the `serverless` command).

Configuring deployment

The SCF component supports "zero" configuration deployment, that is, it can be deployed directly with the default values in the configuration file. Nonetheless, you can also modify more optional configuration items as needed to further customize the project to be deployed.
The following is the description of the SCF component configuration file `serverless.yml` . For more information, please see Full Configuration and Configuration Description.

```
# 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
org: test # Organization information, which is optional. The default value is the `appid` of your Tencent Cloud account
app: scfApp # SCF application name, which is optional
stage: dev # Information for identifying environment, which is optional. The defa ult value is `dev`
inputs:
name: scfFunctionName
src: ./src
runtime: Nodejs10.15 # Runtime environment of function. Valid values: Python2.7, Python3.6, Nodejs6.10, Nodejs8.9, Nodejs10.15, 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 Framework
environment: release
endpoints:
- path: /index
method: GET
```

After updating the fields in the configuration file, run the `serverless deploy` or `serverless` command again to update the configuration to the cloud.

Subsequent Operations

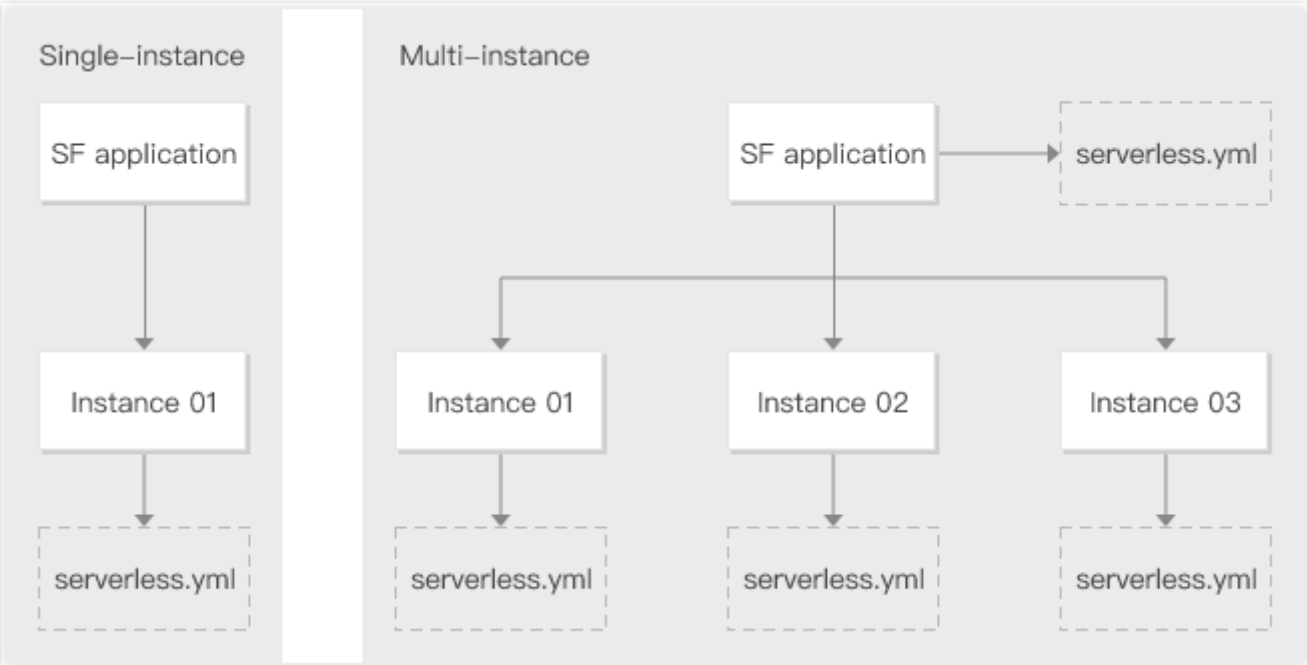
After deploying the function, you can use the development and debugging capabilities provided by the component to re-develop the project into a production-ready application.

Project Application

Last updated：2023-05-04 18:15:28

Overview

The Creating and Deploying Function document describes how to create an SCF function by using Serverless Cloud Framework. For Serverless Cloud Framework, this operation deploys a single-instance serverless application through the SCF component. A serverless application can consist of one or multiple instances, and each component deployment corresponds to one instance. Each instance involves a `serverless.yml` file as shown below, which defines certain parameters of the component. Such parameters are used to generate the instance information during deployment. For example, `region` defines the region where the resources are located.



This document describes single-instance and multi-instance applications and uses actual scenarios as examples to show how to perform project management and resource orchestration for SCF.

Single-instance applications

In the project of a single-instance application, only one component is imported, and only one component instance will be generated during deployment. The name of the single-instance application is generated by Serverless Cloud Framework by default.

Use cases: Serverless Cloud Framework is used only as a CLI tool to create and update functions, and you need to orchestrate and manage function resources by yourself.

Multi-instance applications

In the project of a multi-instance application, multiple components are imported, and multiple component instances will be generated during deployment. You need to enter a fixed name for the multi-instance application to ensure that all components are managed under the same application.

Use cases: you need to organize and orchestrate multiple SCF resources in the project through 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.

Calling SDK Across Functions
Node.js SDK

Last updated：2023-03-14 15:54:10

Tencentcloud-Serverless-Nodejs SDK Overview

Tencentcloud-Serverless-Nodejs is a Tencent Cloud SCF SDK that integrates SCF APIs to simplify the function invocation method. It can invoke a function from a local system, CVM instance, container, or another cloud function, eliminating the need for you to encapsulate TencentCloud APIs.

Features

Tencentcloud-Serverless-Nodejs SDK has the following features:

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

> 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.

Getting Started

Development Preparations

- Development environment
  Node.js 8.9 or a higher version has been installed.
- Running environment
  Windows, Linux, or macOS with tencentcloud-serverless-nodejs SDK have been installed.

- We recommend you use the Serverless Cloud Framework to quickly deploy local functions.

Installing the tencentcloud-serverless-nodejs SDK

Installing via npm (recommended)

1. Select the directory path according to your actual needs and create a directory under it.
   For example, you can create a project directory named `testNodejsSDK` in the `/Users/xxx/Desktop/testNodejsSDK` path.
2. Enter the `testNodejsSDK` directory and run the following commands in sequence to install tencentcloud-serverless-nodejs SDK.

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

After installation, you will be able to see `node_modules` , `package.json` , and `package-lock.json` in the `testNodejsSDK` directory.

Installing via the 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

To install dependencies online with SCF, run the following command in `package.json` :

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

Mutual Recursion

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, functions will invoke one another within the same region.
- 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 containing `SCF Invoke`, such as `SCF FullAccess`). For more information, please see Roles and Policies.

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. Create an `index.js` file in the `testNodejsSDK` directory and enter the following sample code to create an **invoking** Node.js function.

```
const { SDK, LogType } = require('tencentcloud-serverless-nodejs')
exports.main_handler = async (event, context) => {
context.callbackWaitsForEmptyEventLoop = false
const sdk = new SDK({
region:'ap-beijing'
}) // If you bind and run in SCF an execution role with SCF invocation permiss
ions, the authentication information in the environment variable will be used
by default
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 described below:

- **region**: the region of the **invoked** function. The Beijing region selected in step 1 is used as an example in this document.
- **functionName**: the name of the **invoked** function. The `FuncInvoked` function created in step 1 is used as an example in this document.
- **qualifier**: the version of the **invoked** function. If no version is specified, `$LATEST` will be used by default. For more information, please see Viewing a Version.
- **namespace**: the namespace of the **invoked** function. If no namespace is specified, `default` will be used by default.
- **data**: the data passed to the **invoked** function, which can be read from the `event` input parameter.

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 **Local zip package upload**.
  Compress all files in the `testNodejsSDK` directory to ZIP format and upload them to the cloud.

4. In the SCF console, click the function just created, go to **Function management** > **Edit codes**. Click **Test** to run the function. The result should be as follows:

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

Invoking a function locally

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. Create an `index.js` file in the `testNodejsSDK` directory as an **invoking** Node.js function and enter the following sample 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: 'AKxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxj',
secretKey: 'WtxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxqL'
}) // If you bind and run in SCF an execution role with SCF invocation permiss
ions, the authentication information in the environment variable will be used
by default
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: Secret ID and secret key of TencentCloud API. You can obtain them or create new ones by logging into the CAM Console and selecting **Access Key** > **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

We recommend you run the `npm init` command to initialize the SDK before using it.

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

Parameter information:

Parameter Name	Required	Type	Description
region	No	`String`	Region
secretId	No	`String`	process.env.TENCENTCLOUD_SECRETID is used by default
secretKey	No	`String`	`process.env.TENCENTCLOUD_SECRETKEY` is used by default
token	No	`String`	process.env.TENCENTCLOUD_SESSIONTOKEN is used by default

Invoke

This is used to invoke a function. Currently, sync invocation is supported.

Parameter information:

Parameter Name	Required	Type	Description
functionName	Yes	String	Function name
qualifier	No	String	Function version. Default value: $LATEST
data	No	String	Input parameter for function execution
namespace	No	String	Namespace, which is `default` by default.
region	No	String	Region
secretId	No	String	process.env.TENCENTCLOUD_SECRETID is used by default
secretKey	No	String	`process.env.TENCENTCLOUD_SECRETKEY` is used by default
token	No	`String`	process.env.TENCENTCLOUD_SESSIONTOKEN is used by default

SDK for Python

Last updated：2022-12-28 14:47:58

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:

- Invokes 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.
- Supports cross-region function chaining.
- 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.

Getting Started

Mutual function invocation

Samples

> Note：

- To make functions in different regions invoke each other, regions must be specified. For the naming convention, see Common Params.
- 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 an invoked Python function in the cloud named `FuncInvoked` in **Guangzhou** region with the following content:

```
# -*- 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 an invoking Python function in the cloud named `PythonInvokeTest` in **Chengdu** region. You can edit it as needed in the following two methods.

- 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 TencentC
loudSDKException

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 TencentC
loudSDKException

def main_handler(event, context):
#scf = Client(region="ap-guangzhou") # To use this method to establish a `Clie
nt` connection, enable the "execution role" feature in the function configurat
ion and select an execution role with the function invocation permission.
scf = Client(secret_id="AKIxxxxxxxxxxxxxxxxxxxxxxxggB4Sa",secret_key="3vZzxxxxx
xxxxxxaeTC",region="ap-guangzhou",token=" ")# To use this method to establish
a `Client` connection, replace `secret_id` and `secret_key` in the sample code
with your actual `secret_id` and `secret_key`. This key pair needs to contain
the function invocation permission.
print("prepare to invoke a function!")
try:
data = scf.invoke('FuncInvoked',data={"a": "b"})
# data = scf.FuncInvoked(data={"a": "b"}) # To use Python's native invocation
method, perform initialization 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` : TencentCloud API key ID and key, which can be obtained or created in **TencentCloud API Key** > **API Key Management** in the **CAM console**.

Local function invocation

Preparations for development

- Development environment
  Python 2.7 or Python 3.6 has been installed.
- Running environment
  Windows, Linux, or macOS with Tencentserverless SDK installed.

> 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
```

Installation through source package

Go to GitHub to download the latest source package and install it by running the following commands after decompression.

```
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
```

Samples

1. Create an invoked Python function in the cloud named `FuncInvoked` in **Guangzhou** region with the following content:

```
# -*- 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 TencentCl
oudSDKException
def main_handler(event, context):
print("prepare to invoke a function!")
scf = Client(secret_id="AKIxxxxxxxxxxxxxxxxxxxxxxxggB4Sa",secret_key="3vZzxxxxxx
xxxxxaeTC",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 Name	Required	Type	Description
region	No	`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	No	`String`	User `SecretId`, which is obtained from the function's environment variable by default and is **required for local debugging**.
secret_key	No	`String`	User `SecretKey`, which is obtained from the function's environment variable by default and is **required for local debugging.**
token	No	`String`	User `token`, which is obtained from the function's environment variable by default.

- **invoke**

Parameter information:

Parameter Name	Required	Type	Description

function_name	Yes	`String`	Function name.
qualifier	No	`String`	Function version. Default value: $LATEST.
data	No	`Object`	Input parameter for function execution, which must be an object that can be processed by `json.dumps`.
namespace	No	`String`	Namespace. Default value: default.

invoke

This is used to invoke a function. Currently, only sync invocation is supported.

Parameter information:

Parameter	Required	Type	Description
region	No	`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	No	`String`	User `SecretId`, which is obtained from the function's environment variable by default and is **required for local debugging**.
secret_key	No	`String`	User `SecretKey`, which is obtained from the function's environment variable by default and is **required for local debugging**.
token	No	`String`	User `token`, which is obtained from the function's environment variable by default.
function_name	Yes	`String`	Function name.
qualifier	No	`String`	Function version. Default value: $LATEST.
data	No	`String`	Input parameter for function execution, which must be an object that can be processed by `json.dumps`.
namespace	No	`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
Accessing Database

Last updated：2021-10-28 11:55:56

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.

Directions

1. The framework provides a built-in template `database-app`. You can run the following command to quickly initialize a template application related to database operations:

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

2. After the initialization is completed, you only need to change the database connection to the connection in the current actual environment. You can also install the `@malagu/typeorm` component directly in the project by running the following command:

```
yarn add @malagu/typeorm
# Or, run `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
- Multiple

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
```

Database Operation

The following sample uses the RESTful style to implement APIs.

> Note：
> You can also use the RPC style for implementation, and these two styles are similar.

```
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<Ent
```

```
ity>|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>|EntitySchem
a<Entity>|string, name?: string): MongoRepository<Entity> {
...
}
export function getCustomRepository<T>(customRepository: ObjectType<T>, name?: st
ring): T {
...
}
export function pushEntityManager(name: string, entityManager: EntityManager): vo
id {
...
}
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 classes and methods. If it is added to a class and a method at the same time, the final configuration will be to use the configuration of the method to merge the class, which has a higher priority than the class. The decorator configuration options are as follows:

```
export interface TransactionalOption {
name?: string; // In case of multiple data source connections, specify the data s
ource connection name, which is `default` by default
isolation?: IsolationLevel; // Database isolation level
propagation?: Propagation; // Transaction propagation behavior. Valid values: Req
uired, RequiresNew. Default value: Required
readOnly?: boolean; // Read-only mode, i.e., not to start transaction. Transactio
```

```
n is started by default
}
```

Below is a sample:

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

@Transactional and OrmContext

According to the configuration of the decorator, Malagu starts (or does not start) a transaction before invoking a method and hosts the EntityManager in the OrmContext context. OrmContext is fetched to the framework to assist with the EntityManager that has started a transaction, where the repository is created by the managed EntityManager. In order to get the EntityManager correctly, please make sure that the configured name of the decorator is the same as that of the EntityManager to be obtained through OrmContext. If you don't specify a name, the default value will be `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 starting transactions, but we recommend you add the `@Transactional` decorator to the method and configure `readonly` to `true`, so that the framework can create an EntityManager that does not start transactions and maintain a uniform code style. Below is a sample:

```
@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**: data source connection you want to bind to the entity class, which is `default` by default.

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

Below is a sample:

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

Tools

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

Getting Started

Last updated：2021-10-28 11:55:56

You can use the `@malagu/scf-adapter` component to deploy applications in SCF. Based on the principle of convention over configuration, the component can be used out of the box with zero configuration required.

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.

Environment Isolation

Malagu provides the `stage` configuration attribute to represent the environment. In the deployment rule agreed by the `@malagu/scf-adapter` component, the `mode` attribute is used to map the `stage` attribute. Three environments are provided by default: testing, prerelease, and production. The expression rule is as follows:

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

The `stage` value rule is as follows:

- **test**: test environment, i.e., when the `mode` attribute contains the `test` mode, or `mode` does not contain `test` , `pre` , and `prod` and the command line parameter `-p,--prod` is not specified.
- **pre**: prerelease environment, i.e., when the `mode` attribute contains the `pre` mode.
- **prod**: production environment, i.e., when the `mode` attribute contains the `prod` mode, or the command line parameter `-p,--prod` is specified.

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

```
# Deploy to the test environment
malagu deploy -m test # Or use `malagu deploy`
# Deploy to the prerelease environment. You can also skip deploying to the prerel
ease environment and deploy directly to the production environment
malagu deploy -m pre
# Deploy 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:
faas-adapter:
alias:
name: ${stage}
```

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

```
malagu:
faas-adapter:
apiGateway:
release:
environmentName: "${stage == 'pre' ? 'prepub' : stage == 'prod' ? 'release' : sta
ge}"
```

Deployment Mode

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

- **http**: deployment mode based on API Gateway + HTTP-triggered function. During the deployment process, cloud resources such as API gateways, 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 API Gateway + event-triggered function. During the deployment process, cloud resources such as API gateways, namespaces, and functions are created or updated.

```
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-adapter` component.

Custom deployment type

```
mode:
- htpp # Valid values: http, timer, api-gateway. Default value: http
```

Custom namespace

```
malagu:
faas-adapter:
namespace:
name: xxxx # The default value is `default`
```

> Note：
> Other namespace attributes can be configured in a similar way.

Custom function name

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

Note：
Other function attributes can be configured in a similar way.

Attribute Configuration

```
malagu:
faas-adapter:
type:
namespace:
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:
```

Overview

Last updated：2021-10-28 11:55:56

Note：
Malagu is a third-party development tool with no Tencent Cloud official support unavailable currently. If you have any questions or feedback, please go to the Malagu community for discussion and contribution by using issues.

Malagu Overview

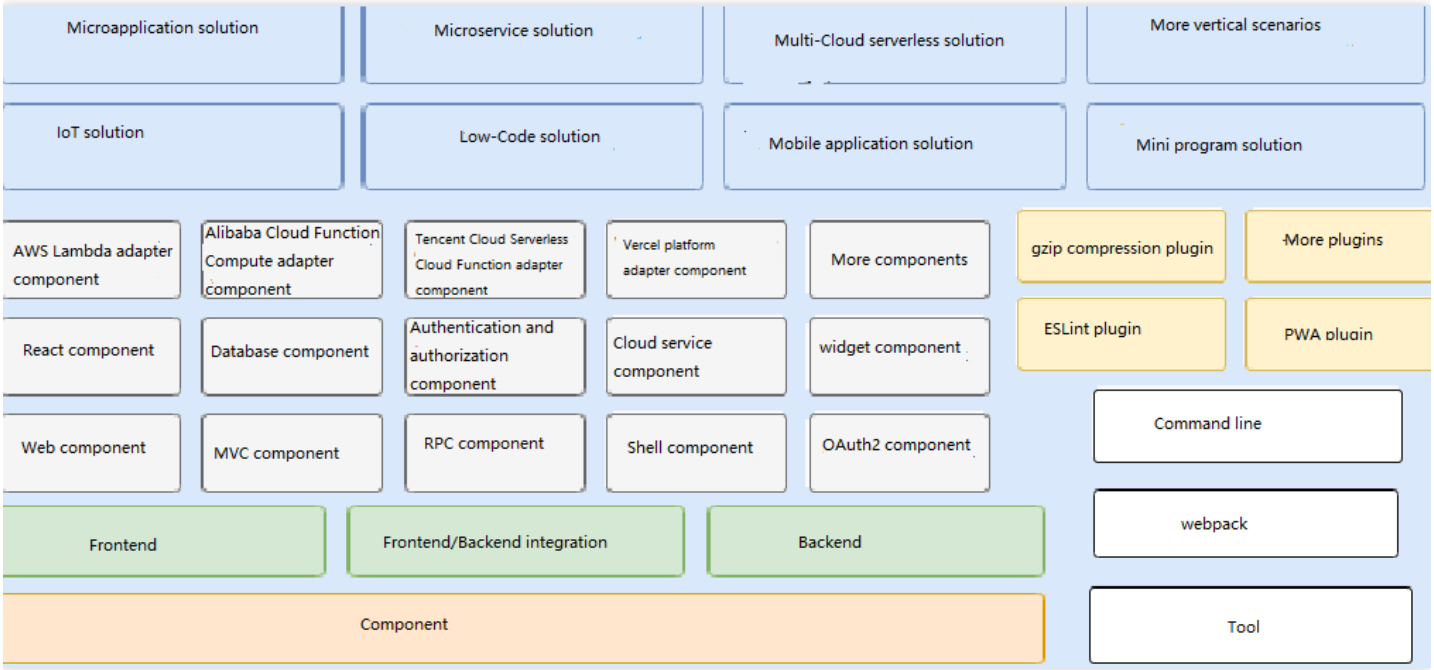
Aka the M framework, Malagu is a serverless-first, componentized, platform-independent progressive application framework based on TypeScript. It uses the same programming language and IoC design to develop frontend, backend, and frontend/backend integrated applications. It combines object-oriented programming (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?

Show All

Firm

展开&收起

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

展开&收起

Currently, all cloud vendors and communities are vigorously promoting and advocating the concept of serverless, through which commercial solutions can be implemented with speed and quality at low costs. It is widely acknowledged in the industry that serverless is the combination of FaaS and BaaS, and it may evolve into other forms in the future. However, no matter how its form changes, the core philosophy of serverless will remain the same. Serverless development experience is subject to the development experience of FaaS, which, however, is not quite satisfactory at present and has many challenges. Some challenges may be hard to overcome at the FaaS underlying layer in the near future, and some may be better to solve at the tool or framework level. Such challenges include cold start, CI/CD, microservice, database access, local development, debugging, and execution, and platform-

independency. More and more serverless-first development frameworks will emerge, which not only are resource orchestration and OPS tools, but also provide more advanced serverless or low-code development platforms.

How to solve such challenges?
You can try solving such problems from the perspective of development framework (which has been proven effective). Then, you should decide whether to use a traditional framework or select a new framework and whether to use a specific or general programming language if you choose a new framework.

Why

展开&收起

With many years of usage of traditional frameworks, most developers can tolerate their development experience. However, when you need to migrate an application developed in a traditional framework to a serverless environment, you will usually encounter various difficult problems, which are generally related to the framework's underlying design. Although you can use the framework's extension capabilities to solve or mitigate some problems, practices show that the threshold for framework transformation is very high, the effect is unsatisfactory, and hacking is required, making the solution less graceful.

If you use a traditional framework in serverless, although your application can run in it, you may still have worries that the application may not be able to run normally in the production environment. Of course, as the underlying technologies of the serverless platform are continuously advanced, the use of traditional frameworks in serverless scenarios are also improved greatly. However, to achieve the optimal status, changes to the application alone may not be enough, and the framework also needs to adapt to serverless scenarios reasonably. Just like when the frontend UI framework tries to be mobile-first, although browsers offer responsive support, the framework also requires adaption. Therefore, a new serverless-first development framework is desired to give full play to the strengths of serverless and make the serverless development experience inherit and even excel the traditional development experience.

Why

展开&收起

Currently, open source communities have many language-neutral serverless tools and frameworks, such as Funcraft, Serverless Framework, and Vercel. Such tools do provide an acceptable experience and can form general standards in terms of OPS, but may be unsatisfactory in terms of the experience of application code development, debugging, and execution. Each programming language has its special benefits in aspects such as development, debugging, and execution, so it is hard for language-neutral serverless tools to achieve an excellent performance while delivering a unified development experience. You can enjoy an ultimate programming experience only by selecting a specific language.

Why

展开&收起

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.

Its frontend architecture is a serverless-like architecture. For example, a frontend browser needs to load frontend code for execution, and user code also needs to be loaded in a serverless scenario for execution. Therefore, many frontend solutions are natively suitable for serverless scenarios. For example, the frontend can reduce the code size, deployment time, and cold start time through packaging, compression, and tree shaking. Similarly, such optimized solutions are also suitable for serverless scenarios. As a result, if you select TypeScript, you can get direct access to many solutions proven and polished by countless real-world use cases.
In addition, TypeScript is similar to Java, so Java developers can easily switch to its technology stack.

Value

展开&收起

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 commands to install the relevant command line tool.

```
$ npm install -g @malagu/cli # Install Malagu command line tool
$ malagu init project-name # Use the `malagu init` command to select a template and initialize a template application
$ cd project-name # Enter the root directory of the application
$ malagu serve # Start the application. The default port is 3000
```

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