# Serverless 应用中心

# 操作指南

# 产品文档

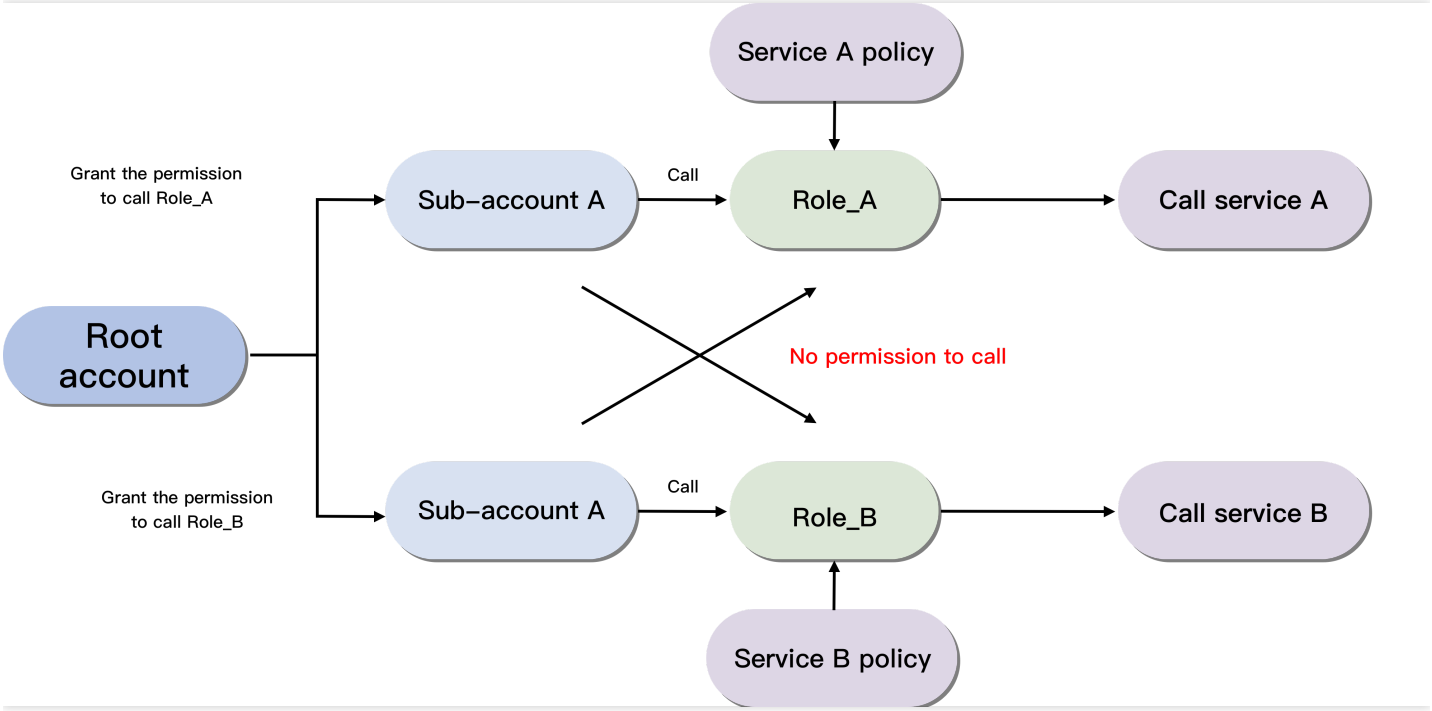
腾讯云

# 文档目录

# 操作指南
# 权限配置
# 指定操作角色配置

最近更新时间：2022-04-28 11:35:11

除了默认角色 SLS_QcsRole 外，主账号还可自行创建多个角色，并将它们分配给每个子用户，各子用户根据自身业务需求只具有相应角色所赋予的策略，达到权限收缩的目的，流程图如下：

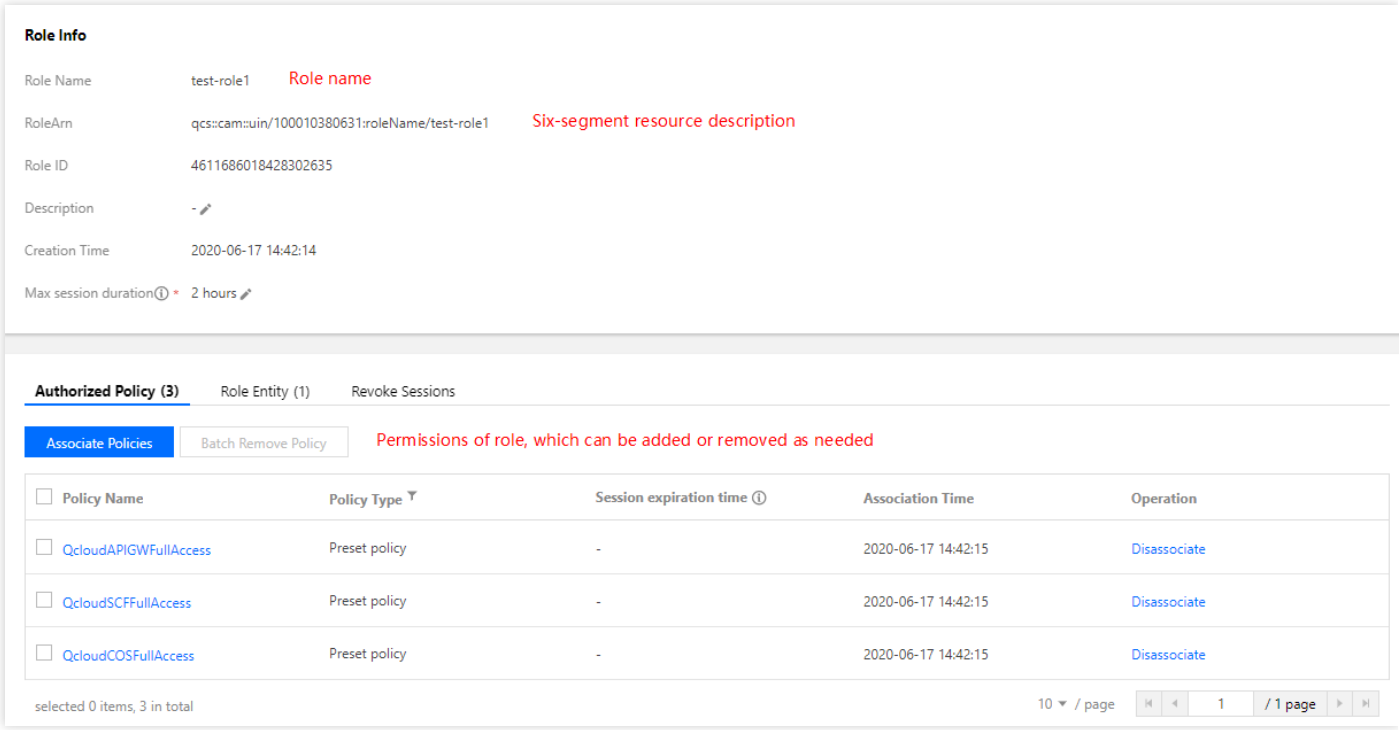



## 主账号配置过程

创建子账号配置角色，并为该角色赋予相应策略。以部署 API 网关触发的 SCF 函数为例：

### 创建子账号角色

1. 主账号登录 CAM 控制台，在左侧导航栏中，单击**角色**。
2. 在角色页面中，单击**新建角色**，并选择角色载体为**腾讯云产品服务**。
3. 在支持角色的服务中，选择**Serverless Framework (sls)**，单击**下一步**。
4. 选择预设策略 **QcloudCOSFullAccess**、 **QcloudAPIGWFullAccess**、**QcloudSCFFullAccess**，单击**下一步**。
5. 填写角色名称（如 test-role1），单击**完成**。
   单击角色名，可以查看配置完成后角色页面：




## 配置角色策略

1. 在左侧导航栏，单击**策略**进入策略管理页。
2. 在策略管理页，单击**新建自定义策略**，并选择**按策略语法创建**。
3. 在策略模板中，选择**空白模板**，单击**下一步**。
4. 填写策略名和内容，并单击**完成**。

绑定角色策略，其中 “resource” 参数填入需要给子账号绑定的角色六段式：

```
{
"version": "2.0",
"statement": [
{
"action": [
"cam:PassRole"
],
"resource": [
# 角色六段式（例："qcs::cam::uin/123456789:roleName/test-role1"）
],
"effect": "allow"
}
]
}
```

说明：
角色资源描述可以在角色信息页面获取。

### 关联子用户策略

1. 在左侧导航栏，单击**用户**>**用户列表**，进入用户列表页。
2. 选择需要授权的子用户，单击操作列的**授权**。
3. 从策略列表中筛选出创建的策略与预设策略**QcloudSLSFullAccess**，单击**确定**，将策略赋予目标子账号，完成角色绑定。
4. **（可选）**如认为 QcloudSLSFullAccess 权限过大，您也可以自己创建自定义策略，为指定资源赋予 SLS 调用权限，策略模板如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"sls:*"
],
"resource": [
#填入项目资源名称（例："qcs::sls:ap-guangzhou::appname/*"）
],
"effect": "allow"
}
]
}
```

说明：
项目资源描述严格按照 CAM 规范进行，您也可以对资源再进行进一步细化，具体到函数名称或者 stage 名称。

## 子账号配置过程

本地创建 Serverless 项目，在配置文件 serverless.yaml 中添加全局配置项 configRole，输入角色名称，后台通过权限检测后即可完成部署：

```
# serverless.yml
component: scf # (必填) 引用 component 的名称，当前用到的是 tencent-scf 组件
name: scfdemo # (必填) 该组件创建的实例名称
org: test # (可选) 用于记录组织信息，默认值为您的腾讯云账户 appid
app: scfApp # (可选) 该 SCF 应用名称
stage: dev # (可选) 用于区分环境信息，默认值是 dev
globalOptions:
configRole: test-role1 # (可选) 填入指定角色名称
inputs:
name: scfFunctionName
src: ./src
runtime: Nodejs10.15 # 云函数的运行时环境。除 Nodejs10.15 外，可选值为：Python2.7、Python3.6、Nodejs6.10、Nodejs8.9、PHP5、PHP7、Go1、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
```

注意：

- 如果不绑定角色，子账号默认使用 SLS_QcsRole 进行 SLS 部署，配置文件中无需填 configRole 参数。
- 一旦绑定角色，请仔细核对 yaml 文件中 configRole 名称，填错或不填都会报错，子账号只支持使用绑定的角色，无权使用其它角色。

## 子账号添加权限

子账号如果需要添加权限，需要将角色名称与需要添加的策略名称一起提供给主账号，主账号在 CAM 控制台 >**角色** 中完成权限添加。

# 账号和权限配置

最近更新时间：2023-06-25 14:47:29

本文为您介绍 Serverless Cloud Framework 的几种授权方式以及通过配置子账号权限进行实际操作演示。

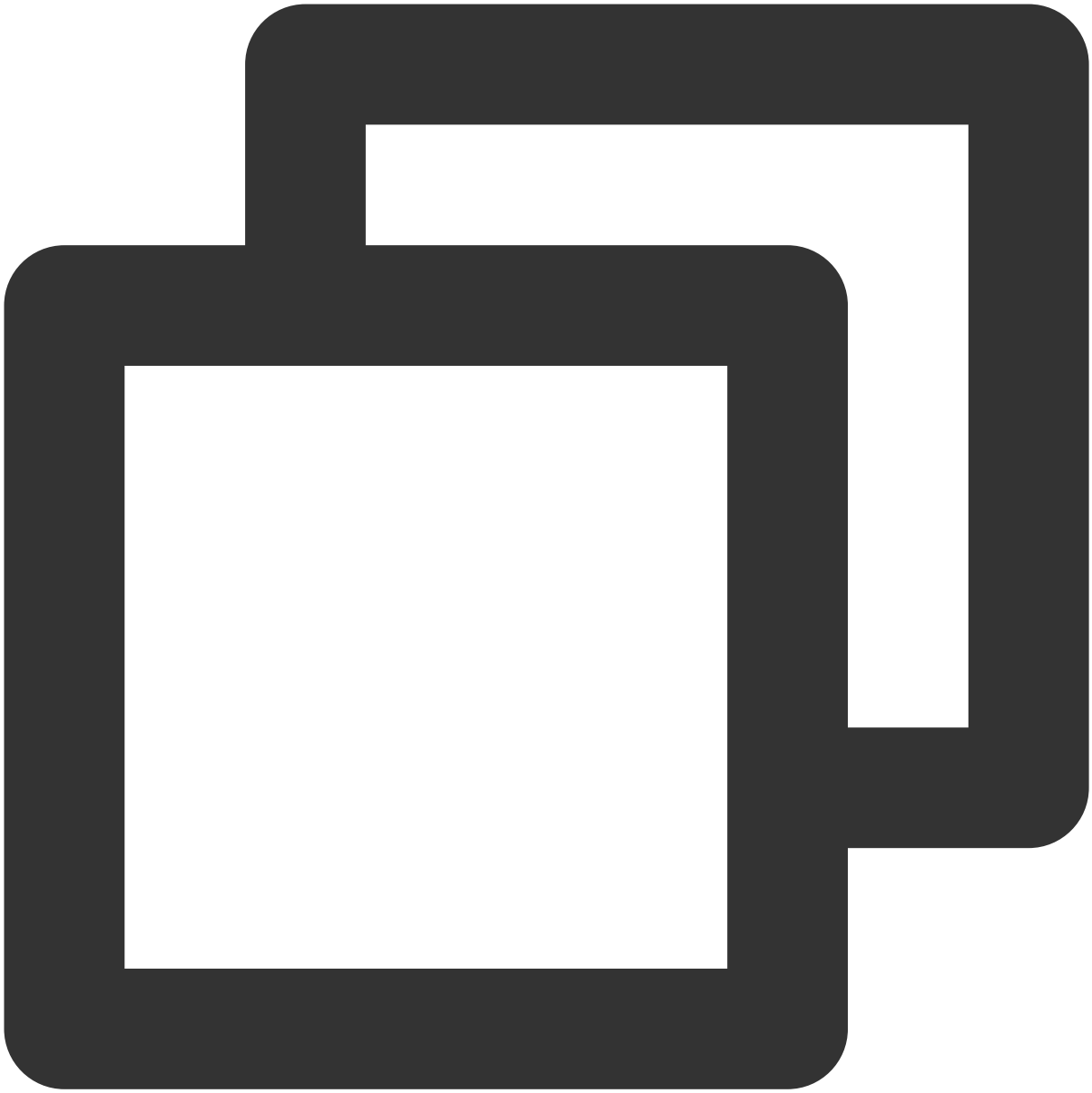
## 前提条件

Serverless Cloud Framework 帮助您将项目快速部署到**腾讯云 Serverless 应用中心**，因此在部署前，请确认您已经注册腾讯云账号。

## 授权方式

### 扫码一键授权

通过 `scf deploy` 进行部署时，您可以通过扫描二维码，一键授权并快速部署，扫码授权后，会生成临时密钥信息（过期时间为60分钟）写入当前目录下的 `.env` 文件中：



```
TENCENT_APP_ID=xxxxxx      #授权账号的 AppId
TENCENT_SECRET_ID=xxxxxx   #授权账号的 SecretId
TENCENT_SECRET_KEY=xxxxxx #授权账号的 SecretKey
TENCENT_TOKEN=xxxxx        #临时 token
```

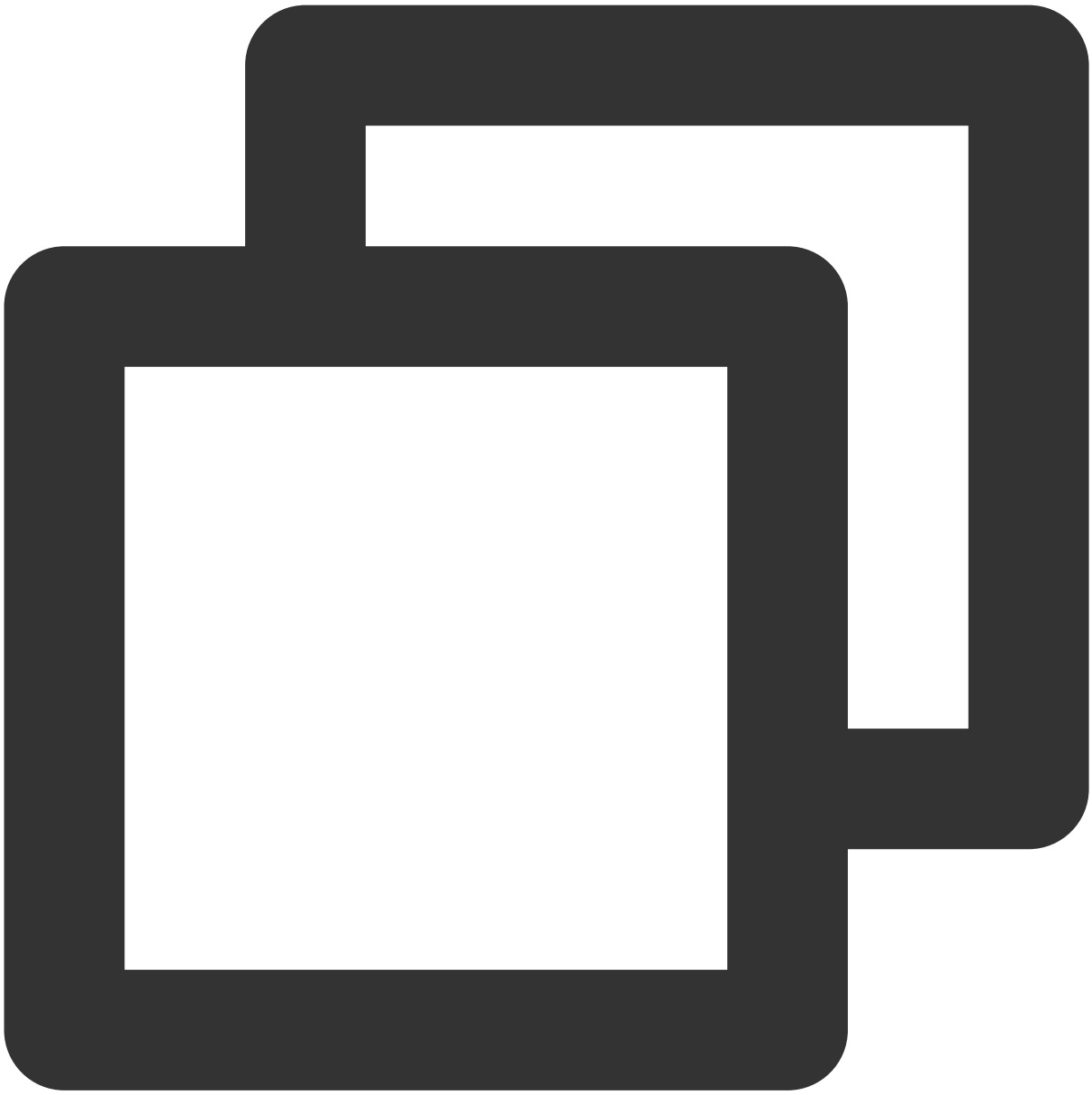
一键授权时获取的权限详情请参见 scf_QcsRole 角色权限列表。

**说明**

如果您的账号为**腾讯云子账号**，扫码部署前需要主账号先进行策略授权配置。配置详情请参见 子账号权限配置。

### 本地密钥授权

为了避免扫码授权过期进行重复授权，您可以采用密钥授权方式。在部署的根目录下创建 `.env` 文件，并配置腾讯云的 SecretId 和 SecretKey 信息：



```
# .env
TENCENT_SECRET_ID=xxxxxxxxxx #您账号的 SecretId
TENCENT_SECRET_KEY=xxxxxxxx #您账号的 SecretKey
```

SecretId 和 SecretKey 可以在 API 密钥管理 中获取 。

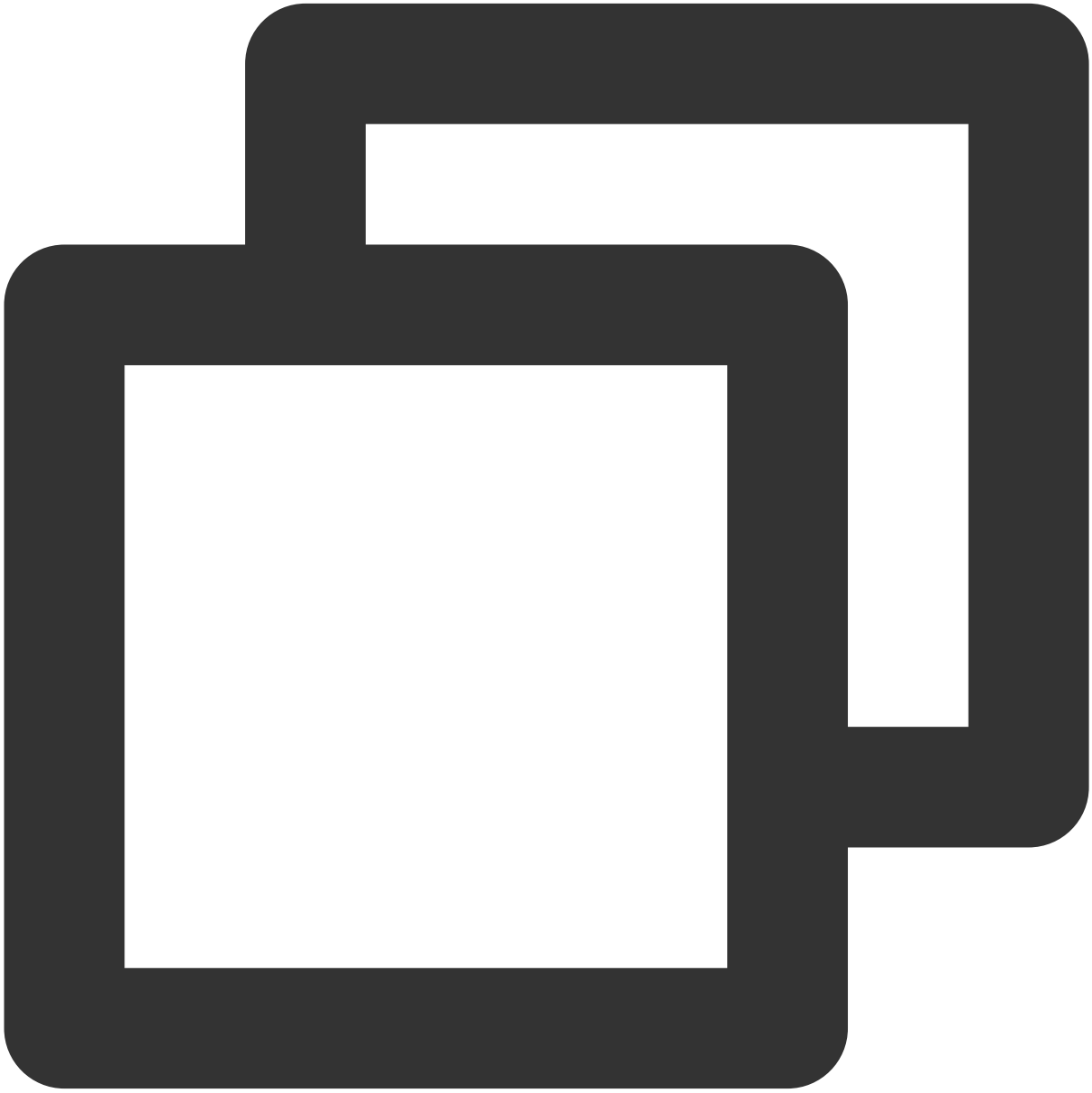
**说明**

为了账号安全性，密钥授权时建议使用**子账号**密钥。子账号必须先被授予相关权限才能进行部署。配置详情请参见子账号权限配置。

### 永久密钥配置

通过 `scf credentials` 指令，可以快速设置全局密钥信息永久保存。该指令必须在已经创建好的 scf 项目下进行配置，请确保您已经通过 `scf init` 或已经手动创建好您的带有 serverless.yml 的项目。

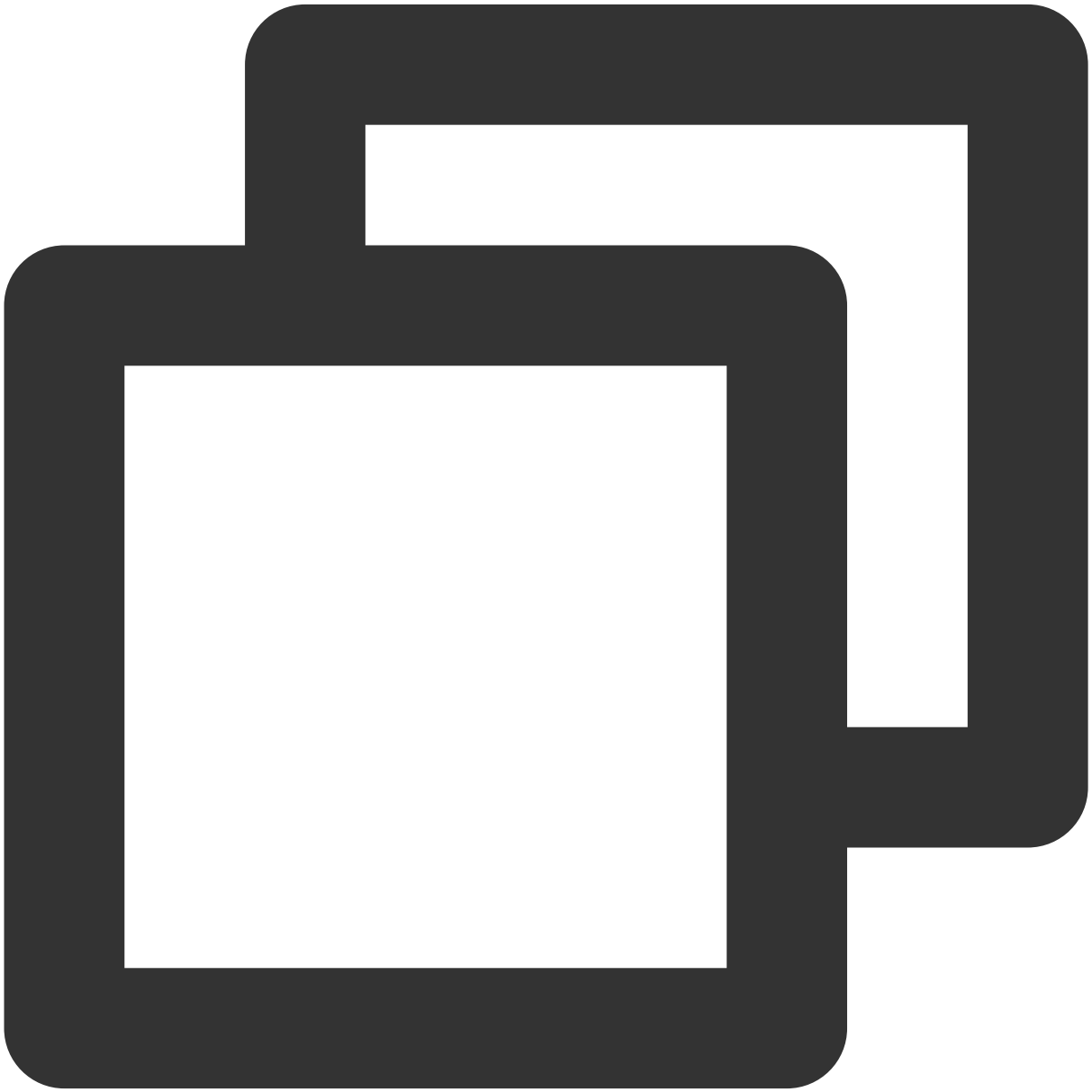
**全部指令参考如下：**



```
scf credentials    管理全局用户授权信息
  set                        存储用户授权信息
```

```
    --secretId / -i          （必填）腾讯云 CAM 账号 secretId
    --secretKey / -k         （必填）腾讯云 CAM 账号 secretKey
    --profile / -n {name}    授权名称，默认为 "default"
    --overwrite / -o         覆写已有授权名称的密钥
remove                   移除用户授权信息
    --profile / -n {name}    （必填）授权名称
list                     查看用户授权信息
```

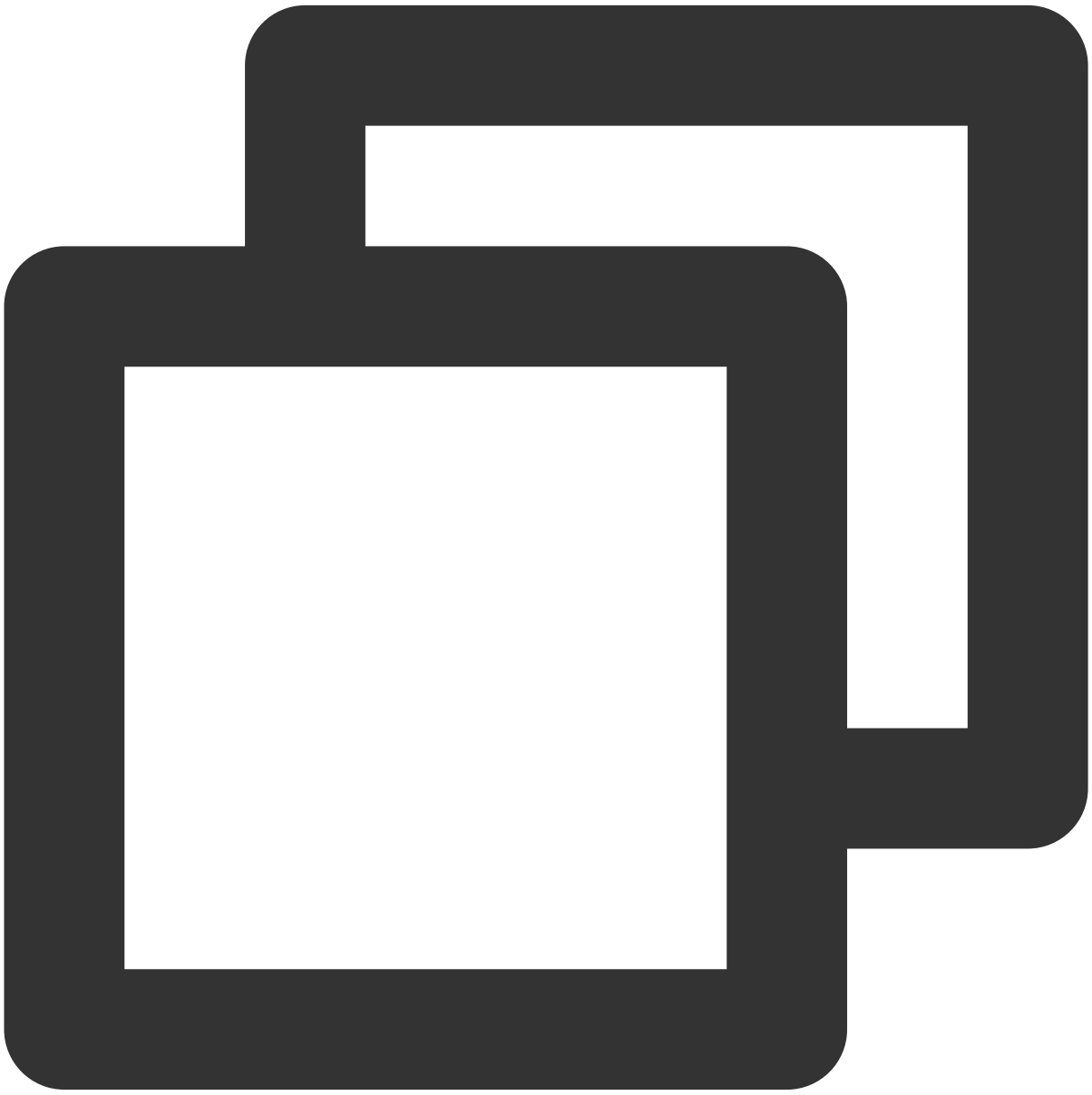
**配置全局授权信息：**



```
# 通过默认 profile 名称配置授权信息
$ scf credentials set --secretId xxx --secretKey xxx
```

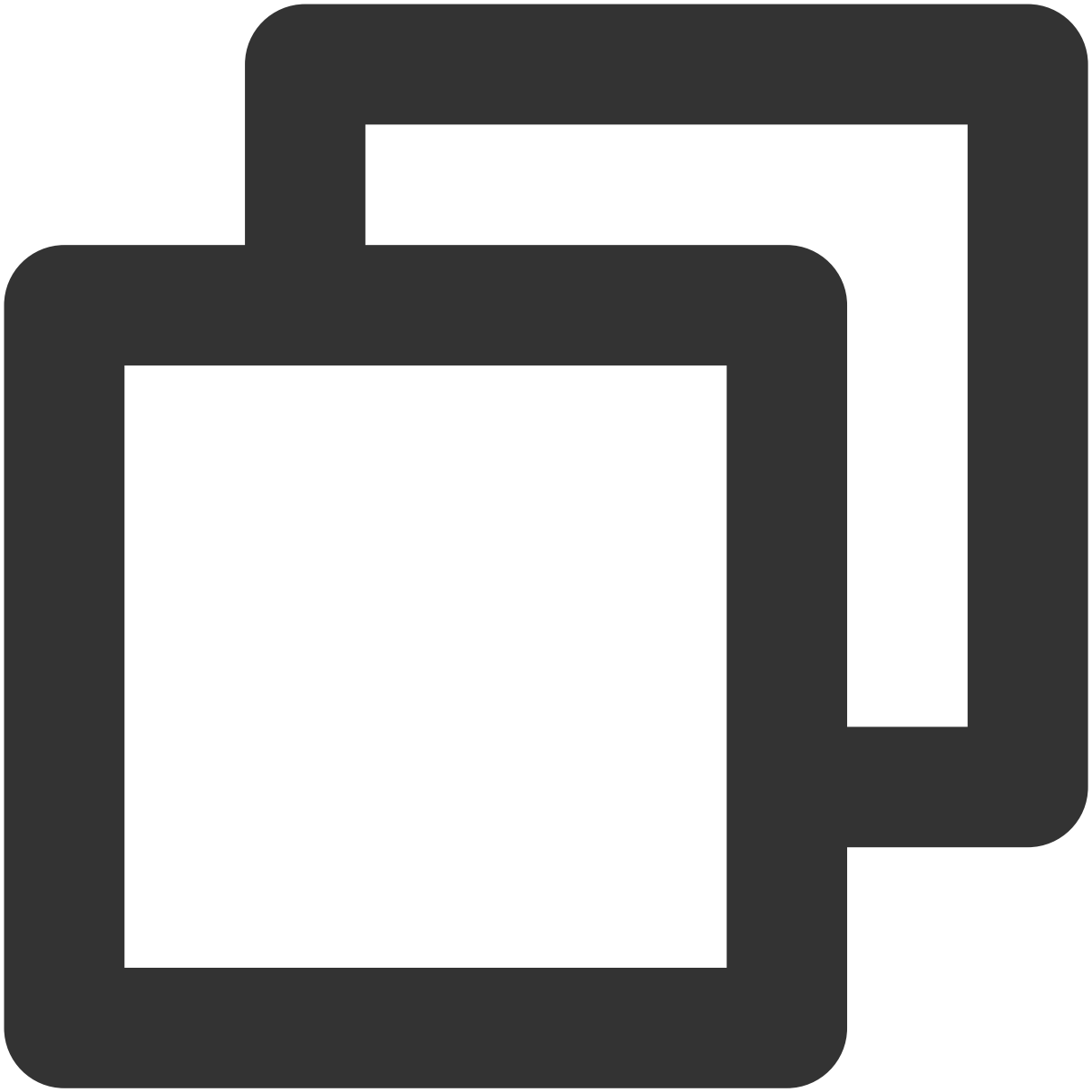
```
# 通过指定 profile 名称配置授权信息
$ scf credentials set --secretId xxx --secretKey xxx --profile profileName1

# 更新指定 profile 名称里的授权信息
$ scf credentials set --secretId xxx --secretKey xxx --profile profileName1 --overw
```

**删除全局授权信息：**



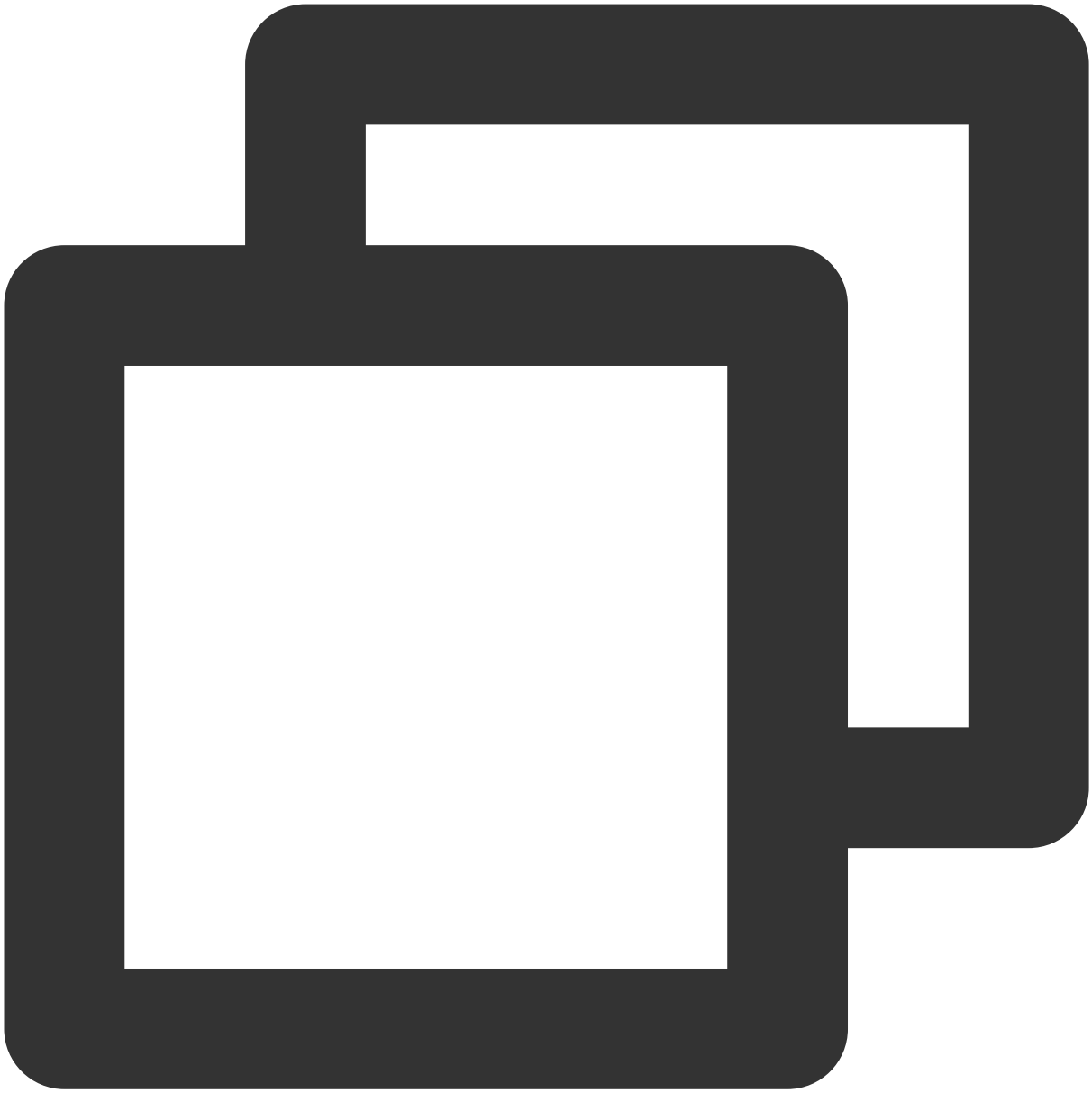
```
$ scf credentials remove --profile profileName1
```

**查看当前所有授权信息：**



```
$ scf credentials list
```

**通过全局授权信息部署：**



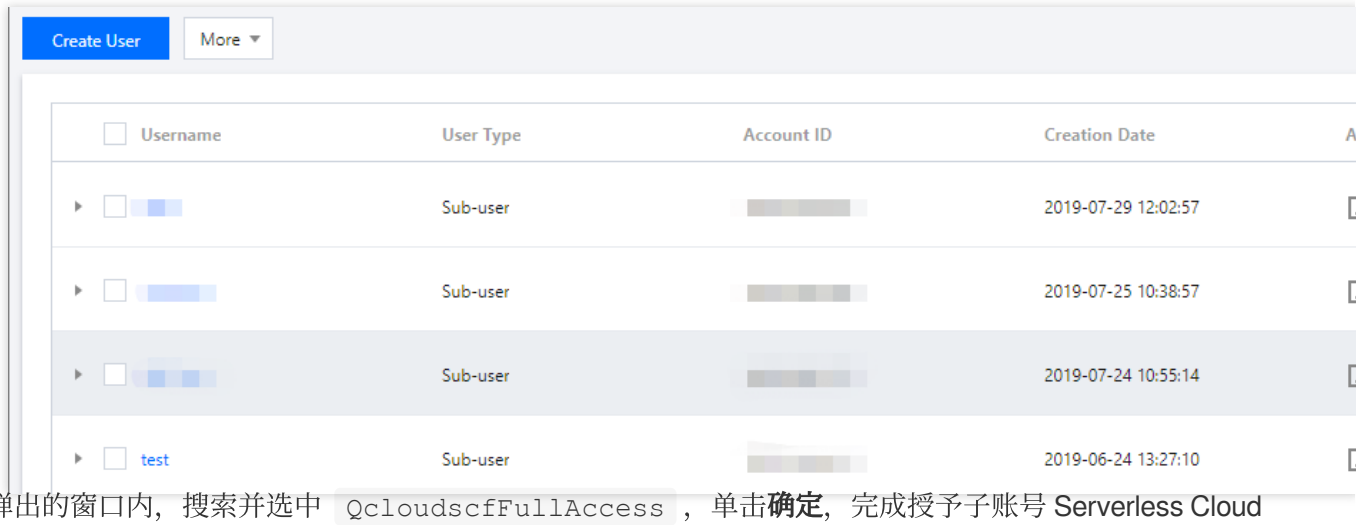
```
# 通过默认 profile 部署
$ scf deploy
# 通过指定 profile 部署
$ scf deploy --profile newP
# 忽略全局变量，扫码部署
$ scf deploy --login
```

## 子账号权限配置

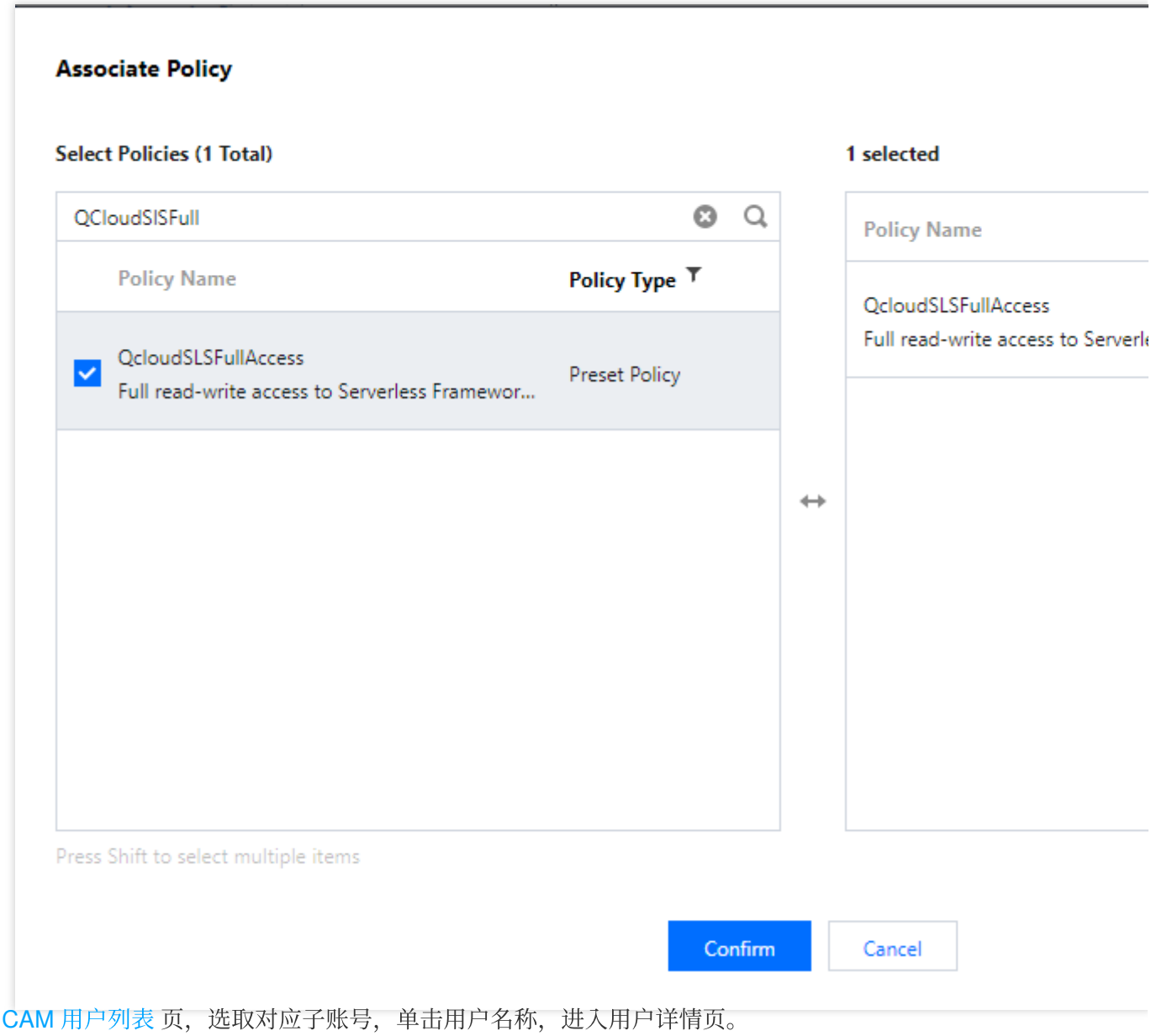
## 配置步骤

如果您的操作账号为腾讯云子账号，没有默认操作权限，则需要**主账号（或拥有授权操作的子账号）**进行如下授权操作：

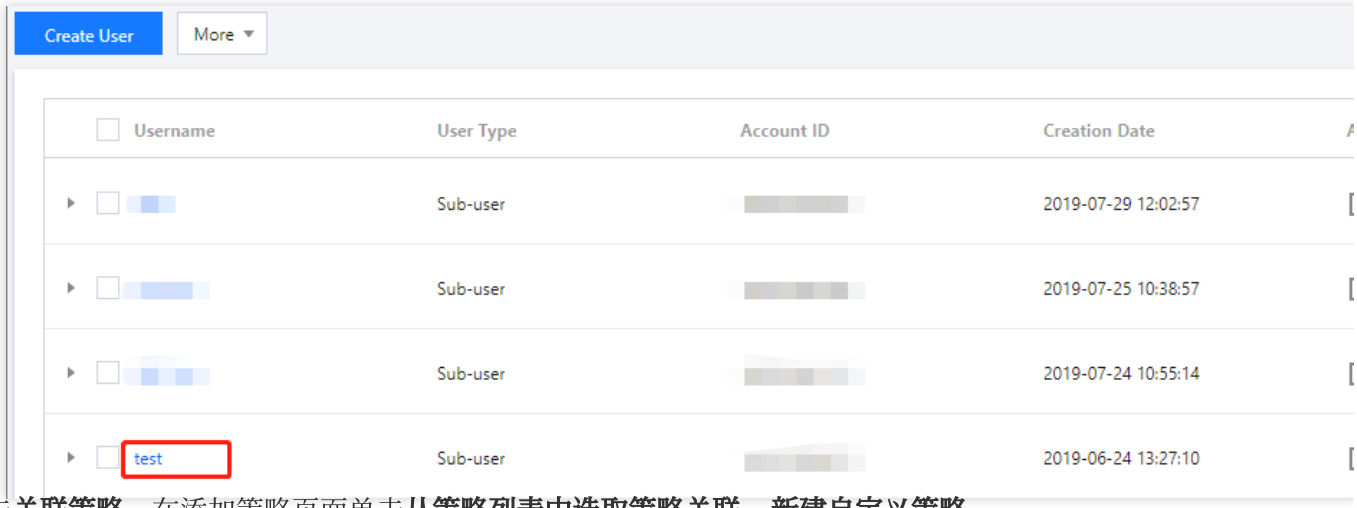
1. 在 CAM 用户列表 页，选取对应子账号，单击**授权**。




2. 在弹出的窗口内，搜索并选中 `QcloudscfFullAccess` ，单击**确定**，完成授予子账号 Serverless Cloud Framework 所有资源的操作权限。

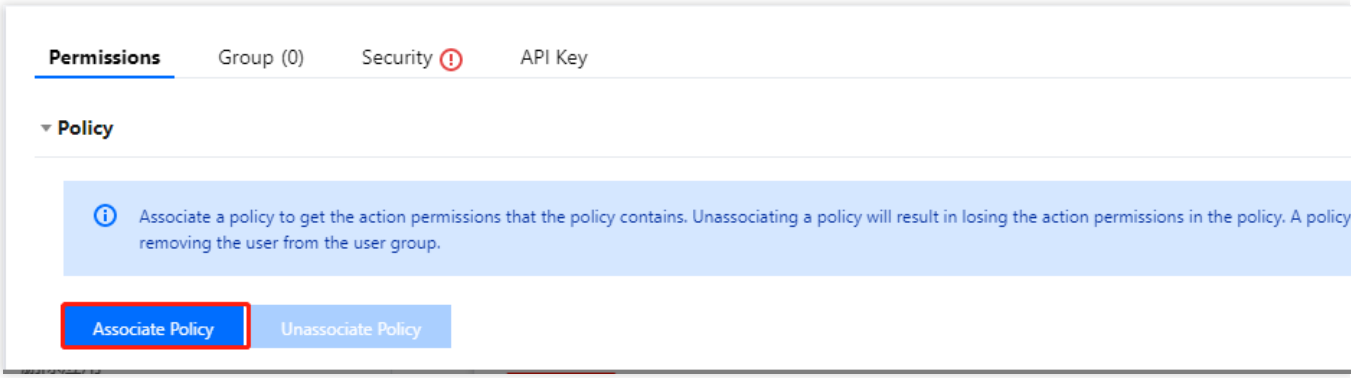



3. 在 CAM 用户列表 页，选取对应子账号，单击用户名称，进入用户详情页。

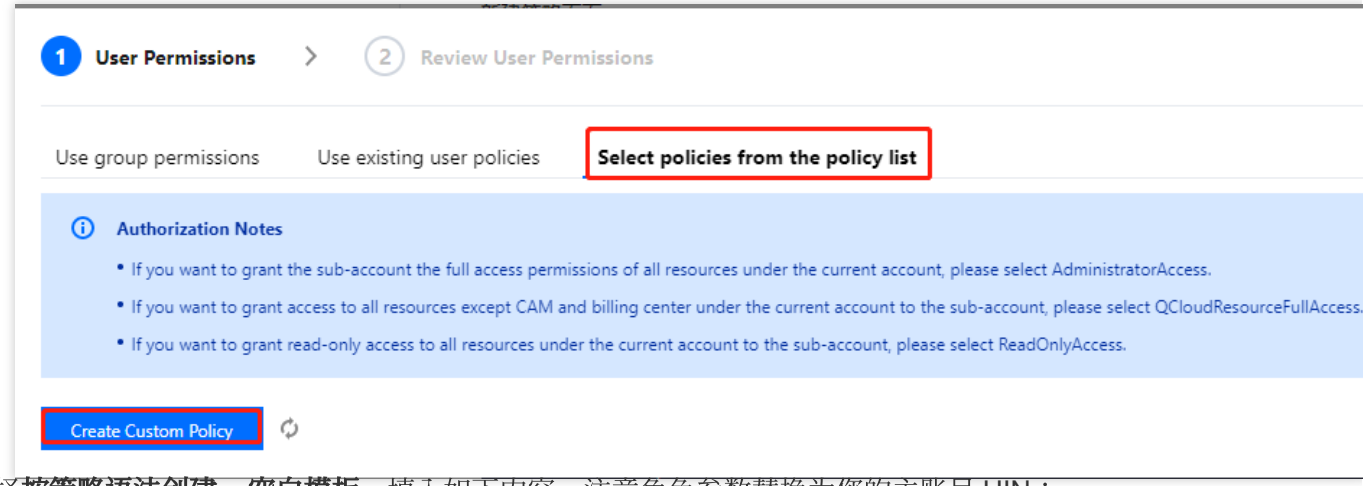



4. 单击**关联策略**，在添加策略页面单击**从策略列表中选取策略关联 > 新建自定义策略**。

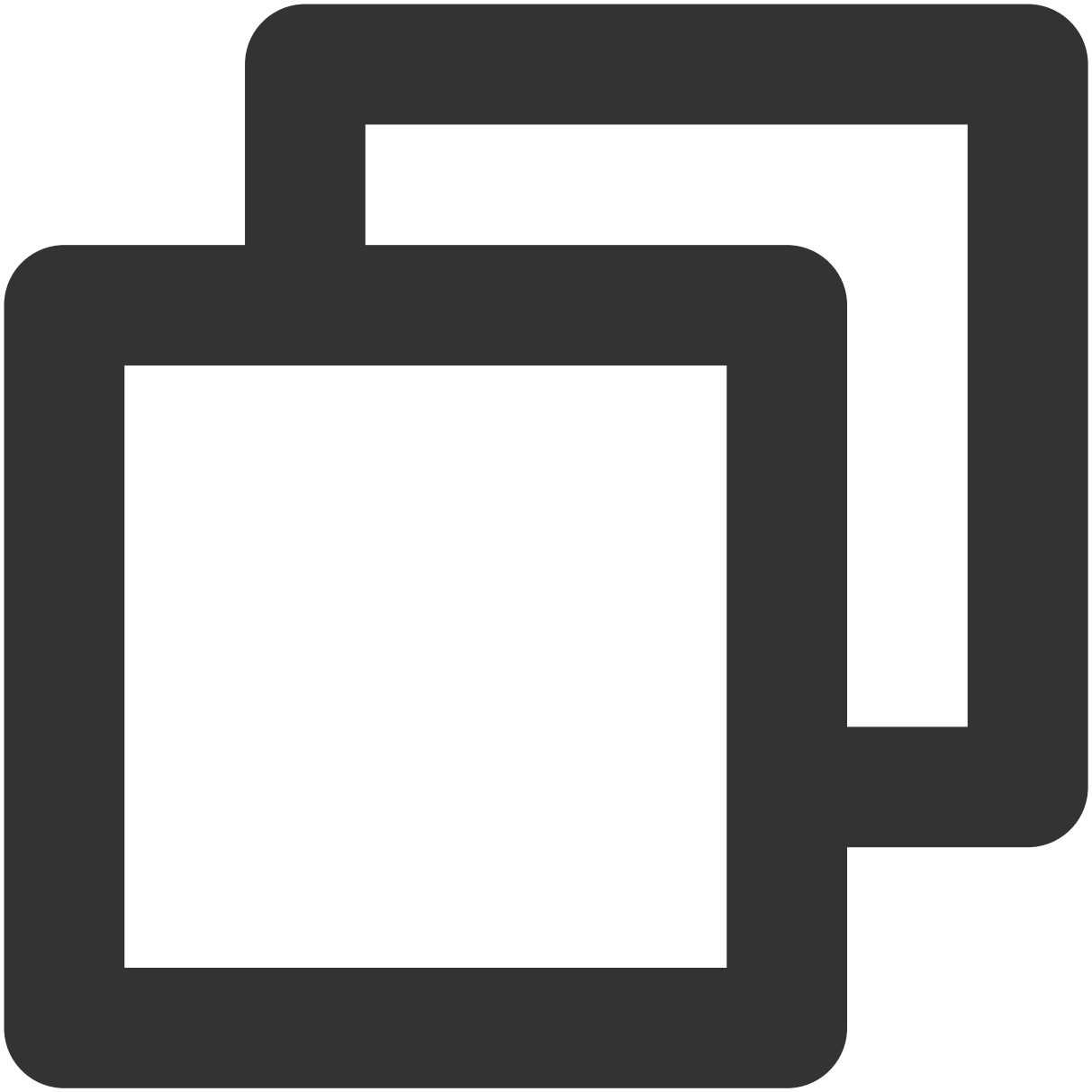
关联策略页面：




新建策略页面：



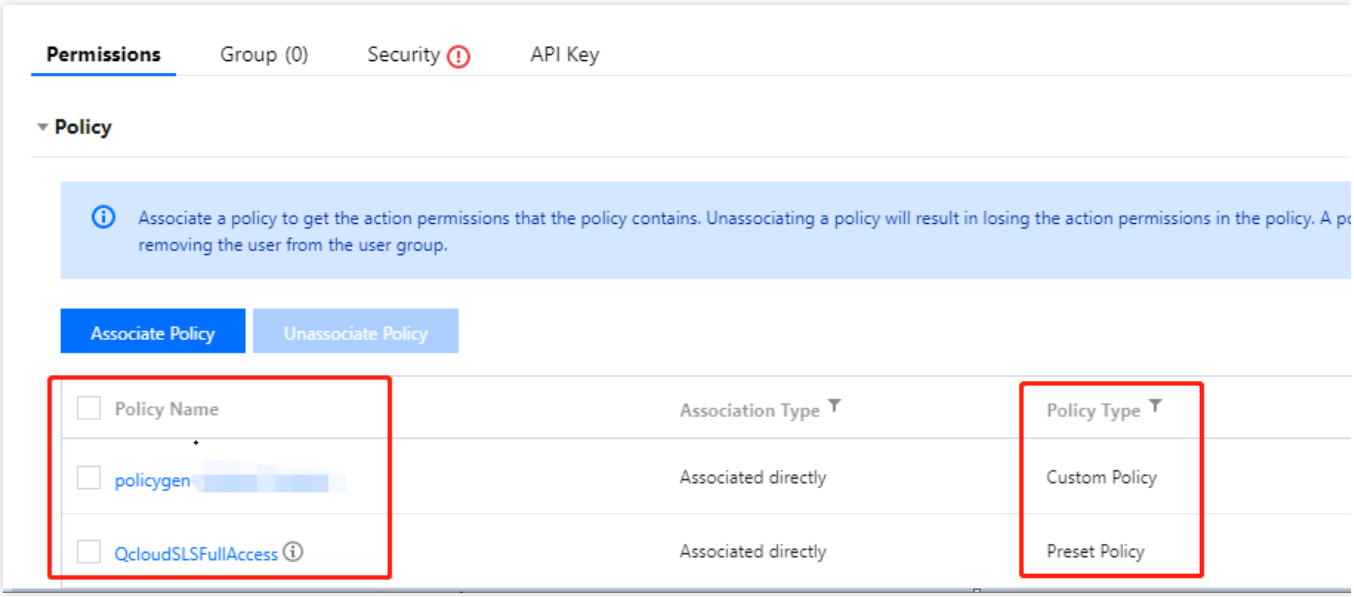

5. 选择**按策略语法创建** > **空白模板**，填入如下内容，注意角色参数替换为您的主账号 UIN：



```
{
 "version": "2.0",
 "statement": [
     {
         "action": [
             "cam:PassRole"
         ],
         "resource": [
             "qcs::cam::uin/${填入账号的 uin}:roleName/scf_QcsRole"
         ],
         "effect": "allow"
```

```
        },
        {
            "resource": [
                "*"
            ],
            "action": [
                "name/sts:AssumeRole"
            ],
            "effect": "allow"
        }
    ]
}
```

6. 完成自定义策略配置后，回到第 4 步的授权页面，搜索刚刚创建的自定义策略，单击**下一步> 确定**，即可授予子账号 scf_QcsRole 的操作权限，此时，您的子账号应该拥有一个自定义策略和一个 **QcloudscfFullAccess** 的预设策略，可以完成 Serverless Framework 的正常使用。




**说明**

除了授权调用默认角色 scf_QcsRole 外，也可给子账号授权调用自定义角色。通过自定义角色中的细粒度权限策略，达到权限收缩的目的。详情请参见 指定操作角色配置。

## scf_QcsRole 角色权限列表

| 策略 | 描述 |
| --- | --- |
| QcloudCOSFullAccess | COS（对象存储）全读写访问权限 |
| QcloudSCFFullAccess | SCF（云函数）全读写权限 |
| QcloudSSLFullAccess | SSL 证书（SSL）全读写访问权限 |
| | |

| | |
|---|---|
| QcloudTCBFullAccess | TCB（云开发）全读写权限 |
| QcloudAPIGWFullAccess | APIGW（API 网关）全读写权限 |
| QcloudVPCFullAccess | VPC（私有网络）全读写权限 |
| QcloudMonitorFullAccess | Monitor（云监控）全读写权限 |
| QcloudslsFullAccess | sls（Serverless Cloud Framework）全读写权限 |
| QcloudCDNFullAccess | CDN（内容分发网络）全读写权限 |
| QcloudCKafkaFullAccess | CKafka（消息队列 CKafka）全读写权限 |
| QcloudCodingFullAccess | CODING DevOps 全读写访问权限 |
| QcloudPostgreSQLFullAccess | 云数据库 PostgreSQL 全读写访问权限 |
| QcloudCynosDBFullAccess | 云数据库 CynosDB 全读写访问权限 |
| QcloudCLSFullAccess | 日志服务（CLS）全读写访问权限 |
| QcloudAccessForscfRole | 该策略供 Serverless Cloud Framework（sls）服务角色（scf_QCSRole）进行关联，用于 scf 一键体验功能访问其他云服务资源。包含访问管理（CAM）相关操作权限 |

# 访问管理配置

最近更新时间：2021-03-29 15:51:15

## 访问管理（CAM）简介

访问管理（Cloud Access Management，CAM）是腾讯云提供的一套 Web 服务，用于帮助客户安全地管理腾讯云账户的访问权限，资源管理和使用权限。通过 CAM，您可以创建、管理和销毁用户（组），并通过身份管理和策略管理控制哪些人可以使用哪些腾讯云资源。

Serverless 应用产品支持**资源级别授权**。您可以通过策略语法给子账号单个资源的管理的权限，详细请参考 授权方案示例。

## 可授权的资源类型

Serverless 应用产品支持资源级授权，您可以指定子账号拥有特定资源的接口权限。支持资源级授权的接口列表如下：

| API 名 | API 描述 | 资源六段式示例 |
| --- | --- | --- |
| SaveInstance | 保存 Component 的 Instance 信息 | `qcs::sls:${Region}:uin/:appname/${AppName}/stagename` |
| GetInstance | 获取 Component 的 Instance 信息 | `qcs::sls:${Region}:uin/:appname/${AppName}/stagename` |
| ListInstances | 获取 Component 的 Instances 列表信息 | `qcs::sls:${Region}:uin/:appname/${AppName}/stagename` |
| RunComponent | 运行一个 Component 实例 | `qcs::sls:${Region}:uin/:appname/${AppName}/stagename` |
| RunFinishComponent | 运行完毕一个 | `qcs::sls:${Region}:uin/:appname/${AppName}/stagename` |

| | | |
|---|---|---|
| | Component 实例 | |

# 授权方案示例

### 资源六段式说明

| 参数 | 是否必填 | 说明 |
|---|---|---|
| qcs | 是 | 是 qcloud service 的简称，表示是腾讯云的云资源。 |
| project_id | 是 | 描述项目信息，仅为了兼容 CAM 早期逻辑。 |
| service_type | 是 | 产品简称，Serverless Framework 的简称为 `sls` 。 |
| region | 是 | 地域信息，如 bj 等，参考 地域列表。 |
| account | 否 | 资源拥有者的主帐号信息，如 uin/164256472。如果为空，表示创建策略的 CAM 用账号。 |
| resource | 是 | 各产品的具体资源详情，Serverless Framework 的为 `qcs::sls:${Region}:uin/:appname/${AppName}/stagename/${Sta` |

### 相关案例

您可以通过主账号登录 访问管理控制台，对 Serverless Framework 进行权限配置和管理。当前 Serverless Framework 提供了**全读写权限**和**只读访问权限**两种预设策略：

#### 全读写权限

- 授权一个子账户以 Serverless Framework（SLS）全读写访问权限。
- 策略名称：QcloudSLSFullAccess

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

```
]
}
```

**只读访问权限**

- 授权一个子账户以 Serverless 应用（SLS）只读写访问权限。
- 策略名称：QcloudSLSReadOnlyAccess

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

# 子账号资源管理

- 子账号可以访问和管理主账号为子账号授权的资源。
- 子账号有创建资源和财务权限时，可以自行购买资源，按正常流程购买，主账号进行扣费。

# yml 文件规范

最近更新时间：2021-03-05 15:25:51

Serverless Framework 通过项目配置文件 `serverless.yml` 完成应用的类型识别与资源配置，本地开发完成后的项目，必须先配置 yml 文件，才可以通过运行 `sls deploy` 命令，将 serverless.yml 中的配置文件和 inputs 中指定参数或代码目录会都被传入 Serverless Components 部署引擎中，从而完成云端部署。

## 基本信息

一个基本的 `serverless.yml` 文件里，第一层配置字段为以下内容：

```
#应用组织信息（可选）
app: '' # 应用名称。留空则默认取当前组件的实例名称为app名称。
stage: '' # 环境名称。默认值是 dev。建议使用 ${env.STAGE} 变量定义环境名称

#组件信息
component: scf # （必选）组件名称，在该实例中为 scf
name: scfdemo # （必选）组件实例名称。

#组件参数配置，根据每个组件，实现具体的资源信息配置
inputs:
```

## 详细配置

在 `inputs` 字段里，根据每个组件创建的云上资源，会进行对应的信息配置，此处以 云函数 SCF 组件 为例，input 字段内的二级目录如下：

```
inputs:
name: xxx # 云函数名称，默认为 ${name}-${stage}-${app}
src: ./src # 项目代码路径，默认写法，新建特定命名的 COS Bucket 并上传
handler: index.main_handler #入口
runtime: Nodejs10.15 # 运行环境 默认 Nodejs10.15
region: ap-guangzhou # 函数所在区域
description: This is a function in ${app} application.
environment: # 环境变量
variables: # 环境变量对象
TEST: value
layers: #layer配置
- name: scfLayer # layer名称
```

```
version: 1 # 版本
events: # 触发器配置
- timer: # 定时触发器
parameters:
cronExpression: '*/5 * * * * * *' # 每5秒触发一次
enable: true
```

# 全量配置列表

目前 Serverless Framework 各个组件的全量配置信息列表如下：

### 基础组件

| 组件名称 | 全量配置 |
| --- | --- |
| SCF 组件 | SCF - serverless.yml 全量配置 |
| Website 组件 | Website - serverless.yml 全量配置 |
| API 网关组件 | API 网关 - serverless.yml 全量配置 |
| VPC 组件 | VPC - serverless.yml 全量配置 |
| COS 组件 | COS - serverless.yml 全量配置 |
| PostgreSQL 组件 | PostgreSQL - serverless.yml 全量配置 |
| CynosDB 组件 | CynosDB - serverless.yml 全量配置 |
| CDN 组件 | CDN - serverless.yml 全量配置 |
| Layer 组件 | Layer - serverless.yml 全量配置 |

### 框架组件

| 组件名称 | 全量配置 |
| --- | --- |
| Express 组件 | Express - serverless.yml 全量配置 |
| Koa 组件 | Koa - serverless.yml 全量配置 |
| Egg 组件 | Egg - serverless.yml 全量配置 |
| Next.js 组件 | Next.js - serverless.yml 全量配置 |
| | |

| | |
|---|---|
| Nuxt.js 组件 | Nuxt.js - serverless.yml 全量配置 |
| Flask 组件 | Flask - serverless.yml 全量配置 |
| Django 组件 | Django - serverless.yml 全量配置 |
| Laravel 组件 | Laravel - serverless.yml 全量配置 |
| ThinkPHP 组件 | ThinkPHP - serverless.yml 全量配置 |

# 项目结构

最近更新时间：2022-10-21 15:51:37

Serverless Cloud Framework 基于 Serverless 组件 完成应用的部署，对于本地项目结构没有强制的规定，但为了便于管理与部署，我们推荐您采用以下几种目录结构组织您的应用：

## 单函数应用

对于单函数的应用，您可以将您的业务代码放置在 src 目录中，并在 `serverless.yml` 配置文件里引用这个目录，实现项目与配置文件的分开管理，示例如下：

```
.
├── serverless.yml # 配置文件
├── src
| ├── package.json # 依赖项文件
| └── index.js # 入口函数
└── .env # 环境变量文件
```

## 多函数/多资源应用

Serverless Cloud Framework 不仅支持单函数的部署，对于多函数的项目也可以实现应用层级的统一部署，对于每一个函数，需要配置对应的配置文件，因此建议目录结构如下：

```
.
├── package.json # 依赖项文件
├── function1
| ├── serverless.yml # 函数1配置文件
| └── index1.js # 入口函数1
├── function2
| ├── serverless.yml # 函数2配置文件
| └── index2.js # 入口函数1
└── .env # 环境变量文件
```

在这种结构下，您只需要在根目录下执行 `scf deploy` ，Serverless Cloud Framework 会自动帮您遍历目录下所有的 yml 配置文件，完成资源的部署。

同时，如果您在函数项目中引入了其它云端资源的创建，也可以采用相同的目录组织方式：

```
.
├── package.json # 依赖项文件
├── src
│ ├── serverless.yml # 函数配置文件
│ └── index1.js # 入口函数
├── cos
│ └── serverless.yml # 对象存储cos桶配置文件
├── db
│ └── serverless.yml # 数据库配置文件
└── .env # 环境变量文件
```

# 本地调试

最近更新时间：2022-04-28 11:54:32

## 操作场景

通过 Serverless Framework 的本地调试能力，您可以在本地的模拟环境中运行代码，发送模拟测试事件，并获取到函数代码的运行日志等信息。

## 前提条件

系统中已安装好 Node.js 环境。

注意：

- 当前命令仅支持 Node.js 和 Python runtime。为保证部署云端和本地运行的结果一致，建议本地安装的 runtime 版本和云端版本保持一致。例如，在云端使用 Node.js 12.x，则本地建议也安装 Node.js 12.x 版本。
- 目前只有 SCF 组件支持本地调试。
- 本地调试仅支持事件类型函数，Web 类型函数请参照 云端测试 进行测试。

## 使用方式

通过 `sls invoke local` 命令完成本地触发运行。Serverless Framework 命令行工具将依据指定的函数模板配置文件，在本地的指定目录中运行相应代码，并通过指定的触发事件，实现在本地云函数模拟环境中运行。

相关命令如下：

```
invoke local .................. 调用本地函数
--function / -f ............函数名称(只能指定统一目录下 yml 里配置的函数名称)
--data / -d ................要传递给调用函数的序列化Event数据(String)
--path / -p ................要传递给调用函数 Event 的 json 文件所在路径
--context ..................要传递给调用函数的序列化 Context 数据(String)
--contextPath / -x .........要传递给调用函数 Context 的 json 文件所在路径
--env / -e .................覆盖环境变量信息，例如 --env VAR1=val1 --env VAR2=val2
--config / -c ..............Path to serverless config file
```

# 操作步骤

下文以 Node.js 为例，指导您如何进行本地调试：

1. 执行以下命令，初始化示例代码。

```
sls init scf-nodejs && cd scf-nodejs
```

2. 在目录下创建测试事件模板 `test.json` 。示例如下：

```
{
"value": "test",
"text": "Hello World事件模板",
"context": {
"key1": "test value 1",
"key2": "test value 2"
}
}
```

3. 创建 `.env` 文件，输入您的永久密钥。示例如下：

```
# .env
TENCENT_SECRET_ID=xxxxxxxxxx #您账号的 SecretId
TENCENT_SECRET_KEY=xxxxxxxx #您账号的 SecretKey
```

说明：
您也可以通过扫码部署，获取临时密钥自动生成配置文件。
4. 执行以下命令，在本地查看调用结果。

```
sls invoke local -p xxxx.json
```

示例如下：

```
# sls invoke local -p test.json
Hello World
{
value: 'test',
text: 'Hello World事件模板',
```

```
context: { key1: 'test value 1', key2: 'test value 2' }
}
undefined
{}
-----------------------------------------------
Serverless: 调用成功

{
value: "test",
text: "Hello World事件模板",
context: {
key1: "test value 1",
key2: "test value 2"
}
}
```

# 构建应用

最近更新时间：2022-04-28 12:02:32

## 操作场景

完成 Serverless Framework 安装后，您可以参考本文档初始化项目模板，并构建多组件应用。

## 前提条件

已 安装 Serverless Framework。

## 操作步骤

### 初始化项目模板

可以通过以下指令，快速初始化一个示例项目，并在此基础上进行改造开发：

```
sls init scf-demo
```

通过该指令，我们在本地快速构建了一个基本的函数应用，目录结构如下：

```
.
├── serverless.yml # 配置文件
└── src
└── index.js # 入口函数
```

进入该目录，可以在示例模板的基础上进行项目的开发。

> 说明：
> `sls init` 支持快速初始化多个项目模板，请通过 `sls registry` 查看所有支持的项目模板。

### 构建多组件应用

Serverless Framework 提供了多个基础资源组件，用户可以通过不同组件的结合使用，快速完成云端资源的创建与部署，无需在控制台手动操作（参考 基础组件列表与配置方式）。

此处以部署一个使用 COS 触发器触发的函数项目为例，教您如何在项目中引入多个组件，并快速完成部署，步骤如下：

1. 调整项目目录结构，新建 COS 文件夹，并在该目录下完成 COS 组件的配置文件 `serverless.yml` 的编写，调整后的目录结构：

```
.
├── src
│ ├── serverless.yml # 函数配置文件
│ └── index.js # 入口函数
├── cos
│ └── serverless.yml # 对象存储 cos 桶配置文件
└── .env # 环境变量文件
```

COS 组件的 yml 文件示例如下，全量配置文件可参考 COS 组件全量配置。

```
app: appDemo
stage: dev
component: cos
name: cosdemo
inputs:
bucket: my-bucket
region: ap-guangzhou
```

2. 修改 SCF 项目的 yml 配置文件，在触发器配置部分按以下语法引用 COS 组件的部署结果：

```
app: appDemo
stage: dev
component: scf
name: scfdemo
inputs:
...
events:
- cos: # cos触发器
parameters:
bucket: ${output:${stage}:${app}:cosdemo.bucket}
```

> 注意：
> 同一个项目内部署多个组件实例时，需要保证每个项目的 `app` 、 `stage` 参数相同，否则无法成功引用。

3. 在项目根目录下，执行 `sls deploy`，即可完成 COS 桶的创建，并将 COS 组件的输出作为 SCF 组件的输入完成触发器的配置。

## 变量引用说明

`serverless.yml` 支持多种方式引用变量：

- **Serverless 基本参数引用**
  在 `inputs` 字段里，支持直接引用 Serverless 基本参数配置信息，引用语法为：`${org}`、`${app}`。

- **环境变量引用**
  在 `serverless.yml` 中，可以直接通过 `${env}` 的方式，直接引用环境变量配置（包含 .env 文件中的环境变量配置，以及手动配置在环境中的变量参数）。
  例如，通过 `${env:REGION}`，引用环境变量 REGION。

- **引用其它组件输出结果**
  如果希望在当前组件配置文件中引用其他组件实例的输出信息，可以通过如下语法进行配置：`${output:[app]:[stage]:[instance name].[output]}`

示例 yml：

```
app: demo
component: scf
name: rest-api
stage: dev
inputs:
name: ${stage}-${app}-${name} # 命名最终为 "acme-prod-ecommerce-rest-api"
region: ${env:REGION} # 环境变量中指定的 REGION= 信息
vpcName: ${output:prod:my-app:vpc.name} # 获取其他组件中的输出信息
vpcName: ${output:${stage}:${app}:vpc.name} # 上述方式也可以组合使用
```

# 云端调试

最近更新时间：2022-06-02 17:13:17

## 开发模式

开发模式是为处于开发状态下的项目可以更便捷的进行代码编写、开发调试而设计的。在开发模式中，用户可以持续地进行开发-调试的过程，尽量减少打包、更新等其他工作的干扰。

### 进入开发模式

在项目下执行 `serverless dev` 命令，可以进入项目的开发模式。

示例如下：

```
$ sls dev
serverless ⚡ 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-b77xtibo
subDomain: service-b77xtibo-1253970226.gz.apigw.tencentcs.com
environment: release
url: http://service-b77xtibo-1253970226.gz.apigw.tencentcs.com/release/
scf:
functionName: express_component_6r6xkh60k
runtime: Nodejs10.15
namespace: default
express-api-demo › Watching
```

在进入 dev 模式后，Serverless 工具将输出部署的内容，并启动持续文件监控；在代码文件有修改的情况下，将自动再次进行部署，将本地文件更新到云端。

再次部署并输出部署信息：

```
express-api-demo › Deploying ...
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 ----------------------
21:11:31 - express-api-demo - deployment
region: ap-guangzhou
apigw:
serviceId: service-b7dlqkyy
subDomain: service-b7dlqkyy-1253970226.gz.apigw.tencentcs.com
environment: release
url: http://service-b7dlqkyy-1253970226.gz.apigw.tencentcs.com/release/
scf:
functionName: express_component_uo5v2vp
runtime: Nodejs10.15
namespace: default
```

注意：
当前 `serverless dev` 仅支持 Node.js 10 运行环境，后续将支持 Python、PHP 等运行环境的实时日志。

### 退出开发模式

在开发模式下，通过 Ctrl+C 可以退出开发模式（dev 模式）。

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

## 云端调试：Node.js 10+

针对 Runtime 为 Node.js 10+ 的项目，可以通过开启云端调试，并使用针对 Node.js 的调试工具来连接云端调试，例如 Chrome DevTools、VS Code Debugger。

### 开启云端调试

在按如上方案进入开发模式时，如果是 Runtime 为 Node.js 10及以上版本的函数，会自行开启云端调试，并输出调试相关信息。

例如在开启开发模式时，如果有如下输出，则代表已经启动云函数的云端调试：

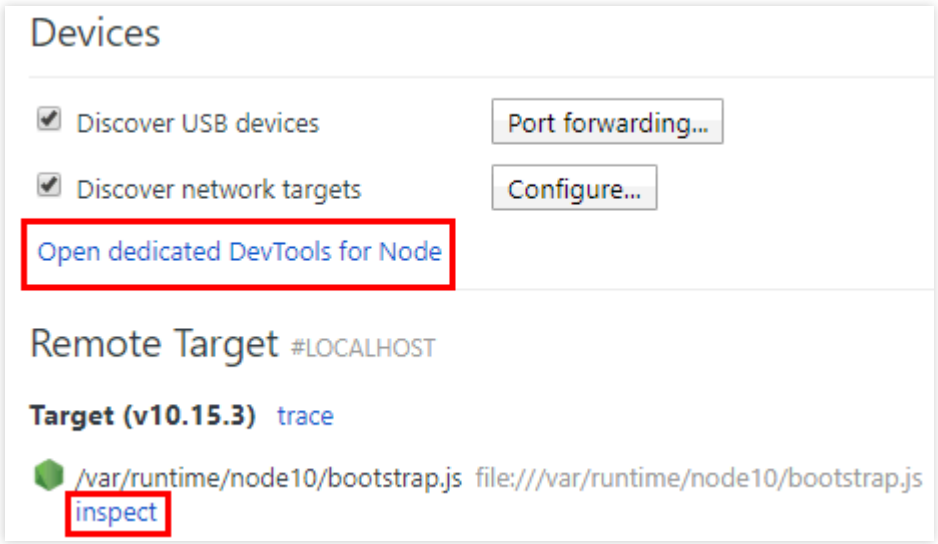
```
Debugging listening on ws://127.0.0.1:9222.
For help see https://nodejs.org/en/docs/inspector.
```

```
Please open chorme, and visit chrome://inspect, click [Open dedicated DevTools fo
r Node] to debug your code.
```

### 使用调试工具 Chrome DevTools

以下步骤说明如何使用 Chrome 浏览器的 DevTools 工具来连接远程环境并进行调试：

1. 启动 Chrome 浏览器。
2. 在地址栏中输入 `chrome://inspect/` 并访问。
3. 可通过以下两种方式打开 DevTools。如下图所示：




4. （推荐）单击 Devices 下的**Open dedicated DevTools for Node**。
5. 选择 Remote Target #LOCALHOST 中具体 Target 下的**inspect**。
   如果无法打开或者没有 Target，请检查 Device 的 Configure 中是否已有 `localhost:9229` 或 `localhost:9222` 的配置，该配置对应开启云端调试时的输出。
6. 通过选择**Open dedicated DevTools for Node**方式打开的 DevTools 调试工具，可单击**Sources**页签看远端代码。函数的实际代码在 `/var/user/` 目录下。
   在**Sources**页签中查看的代码可能处于加载中，会随着调试进行而展示出更多远端文件。
7. 可按需打开文件，在文件的指定位置设置断点。
8. 通过任意方式，例如 URL 访问、页面触发、命令触发、接口触发等方式触发函数，会使得远端环境开始运行，并会在设置了断点的位置中断，等待进一步的运行。
9. 通过 DevTools 的右侧工具栏，可以控制中断的程序继续执行、单步执行、步入步出等操作，也可以直接查看当前变量，或设定需跟踪查看的变量。DevTools 的进一步使用可以搜索查询 DevTools 使用说明文档。

### 关闭云端调试

在退出开发模式时，将会自动关闭云端调试功能。

# 命令调试

Serverless Framework 云函数组件支持 `invoke` 命令触发云函数进行调试。对于 `sls deploy` 部署成功的云函数，进入项目目录，执行函数调用命令如下：

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

说明：

- `invoke` 命令必须在该函数部署的 serverless.yml 文件同目录下执行。
- `clientContext` 为触发函数时传递的 json 字符串。可以根据 触发事件模板 的 json 字符串格式模拟不同触发事件。

# 部署应用

最近更新时间：2021-01-15 15:30:53

## 操作场景

完成本地项目开发后，您可以快速部署应用、查看部署信息并进行函数调试。

## 前提条件

已完成本地项目开发（参考 项目开发）。

## 操作步骤

### 快速部署

您可以通过 Serverless Framework，快速将项目部署到云端，具体操作如下:

```
sls deploy
```

输入该指令后，Serverless Framework CLI 将会为您完成以下操作：

#### 1. 扫码授权

通过扫描二维码，完成一键授权，授权后 CLI 工具会将生成临时密钥信息写入当前目录下的 `.env` 文件里，临时密钥的有效时间 2 小时，失效后，部署的时候会引导您重新扫码鉴权。

如果不想重复扫码，您也可以将永久密钥配置在项目目录下的 `.env` 文件中：

```
# .env
TENCENT_SECRET_ID=xxxxxxxxxx #您账号的 SecretId
TENCENT_SECRET_KEY=xxxxxxxx #您账号的 SecretKey
```

SecretId 和 SecretKey可以在 API 密钥管理 中获取 。

#### 2. 打包上传

完成授权后，Serverless Framework CLI 会根据您在 `serverless.yml` 文件中配置的项目代码路径，自动为您进行项目的打包与上传。

#### 3. 云端部署

上传后的项目，会根据您在 yml 文件中进行的参数配置，完成云上资源的创建，部署完成后，命令行会输出部署后的资源信息。

### 高级能力

- 查看部署过程中的具体日志信息：

```
sls deploy --debug
```

- 多版本部署时，切换指定流量到 $latest 函数版本，其余流量到最后一次发的函数版本上，实现灰度发布。

```
sls deploy --inputs traffic=0.1 public=true
```

- 应用目录下含有多个 Serverless 实例，只需要更新指定项目：

```
sls deploy --target xxx
```

例如：在该项目根目录下，通过指令 `sls deploy --target ./cos`，仅更新 cos 实例，其它实例不受影响

```
.
├── src
│   ├── serverless.yml
│   └── index1.js
├── cos
│   └── serverless.yml
├── db
│   └── serverless.yml
└── .env
```

### 查看部署信息

完成部署后，通过以下指令，查看项目的部署信息：

```
sls info
```

### 函数调试

**说明：**

当前该指令只支持通过 Serverless Framework 云函数组件 部署的函数项目，其它组件的支持也在计划中。

Serverless Framework 云函数组件支持通过 `invoke` 命令触发云函数进行调试。对于 `sls deploy` 部署成功的云函数，进入对应函数的项目目录下，执行函数调用命令，即可完成云上函数资源的远程调试，调试结果会在命令行进行输出：

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

- `invoke` 命令必须在该函数部署的 serverless.yml 文件同目录下执行。
- `clientContext` 为触发函数时传递的 json 字符串。可以根据 触发事件模板 的 json 字符串格式模拟不同触发事件。

# 常见问题

如您的环境配置了代理，可能会出现以下问题：

- 问题1：输入 `serverless` 时没有默认弹出中文引导。
  解决方案： 请确认您的 IP 在中国大陆区域，并在 .env 文件中增加配置 SERVERLESS_PLATFORM_VENDOR=tencent 即可。

- 问题2：输入 `sls deploy` 后部署报网络错误。
  解决方案：在 .env 文件中增加以下代理配置。

  ```
  HTTP_PROXY=http://127.0.0.1:12345 #请将'12345'替换为您的代理端口
  HTTPS_PROXY=http://127.0.0.1:12345 #请将'12345'替换为您的代理端口
  ```

# 删除应用

最近更新时间：2021-01-15 15:30:53

## 基本功能

通过以下指令，可以快速删除云端资源：

```
sls remove
```

## 高级功能

- 查看删除过程中的具体日志信息：

  ```
  sls remove --debug
  ```

- 应用目录下含有多个 Serverless 实例，只需要删除指定项目：

  ```
  sls remove --target xxx
  ```

  例如：在该项目根目录下，通过指令 `sls remove --target ./cos`，仅移除 COS 实例，其它实例不受影响。

  ```
  .
  ├── src
  │   ├── serverless.yml
  │   └── index1.js
  ├── cos
  │   └── serverless.yml
  ├── db
  │   └── serverless.yml
  └── .env
  ```

## 常见问题

**执行 `sls remove` 时，会移除哪些云端资源？**

Serverless Framework 根据 yml 配置文件完成云端资源的移除，只要通过 yml 文件创建的资源，都会进行删除操作；如果引用的已有资源，则不会进行移除。例如：

- 通过 SCF 组件部署函数时，选择新建 API 网关触发器进行部署，则在删除时，创建的函数与 API 网关资源都会进行删除。
- 选择已有 API 网关触发器进行部署，则在删除时，仅删除函数资源，使用的 API 网关不会进行删除操作。

# 支持命令列表

最近更新时间：2022-06-13 15:10:22

Serverless 应用基于 Serverless Framework 部署，支持的 CLI 命令如下：

- ** `serverless registry` **：查看可用的 Components 列表。
- ** `serverless registry publish` **：发布 Component 到 Serverless 组件仓库。

  `--dev` ：支持 dev 参数用于发布 `@dev` 版本的 Component，用于开发或测试。
- ** `serverless init xxx` **：从组件仓库下载指定模板，init 后填入您想下载的模板名称，例 "$ serverless init fullstack"

  `sls init xxx --name my-app` ：支持自定义项目目录名称。

  `--debug` ：列出模板下载过程中的日志信息。
- ** `serverless deploy` **：部署 Component 实例到云端。

  `--debug` ：列出组件部署过程中 `console.log()` 输出的部署操作和状态等日志信息。

  `---inputs publish=true` ：部署函数时发布新版本。

  `---inputs traffic=0.1` ：部署时切换10%流量到 $latest 函数版本，其余流量到最后一次发的函数版本上。

  > 说明：
  > 旧版本命令为 `sls deploy --inputs.key=value` ，Serverless CLI V3.2.3 后命令统一格式为 `sls deploy --inputs key=value` ，旧版本命令在新版本 Serverless CLI 中不可用，升级 Serverles CLI 的用户请使用新版本命令。
- ** `serverless remove` **：从云端移除一个 Component 实例。

  `--debug` ：列出组件移除过程中 `console.log()` 输出的移除操作和状态等日志信息。

- ** `serverless info` **：获取并展示一个 Component 实例的相关信息。

  `--debug` ：列出更多 `state` 。

- ** `serverless dev` **：启动 DEV MODE 开发者模式，通过检测 Component 的状态变化，自动部署变更信息。同时支持在命令行中实时输出运行日志，调用信息和错误等。此外，支持对 Node.js 应用进行云端调试。

# 多函数应用部署

最近更新时间：2021-11-08 16:54:26

基于腾讯云 multi-scf 组件，您可以快速构建您的多函数应用并完成部署，大幅降低了复杂应用的开发成本。

### 使用前提

- 已安装 Serverless Framework，详情见 安装 Serverless Framework。
- 账号开通 Serverless 相关权限，详情见 账号和权限配置。

### 开发部署步骤

示例项目详情见 案例列表。

1. 本地开发您的应用项目，本文以含有两个函数的项目为例，应用目录结构如下：

```
./multi-scf-demo
├── index
│   ├── index.js # 主函数1
│   ├── package.json
│   └── scf_bootstrap # web 类型函数启动文件，事件函数可忽视
├── user
│   ├── index.js # 主函数2
│   ├── package.json
│   └── scf_bootstrap # web 类型函数启动文件，事件函数可忽视
└── serverless.yml # yml 配置文件
```

2. 在**根目录**下，创建 `serverless.yml` 文件，参考如下 yml 示例，为您的项目进行相关的参数配置。更多配置内容，请参见 全量配置。

```
app: multi-scf #应用名称
component: multi-scf #组件类型，为 multi-scf
name: web_demo #实例名称，可以自定义
inputs:
src:
# 这里必须指定代码目录，云端自动根据函数配置来拆分函数代码
src: ./
exclude:
- .env
region: ap-guangzhou #地域
runtime: Nodejs12.16 #函数语言版本
memorySize: 512
```

```
timeout: 3
type: web #函数类型，此处为 web 函数
functions:
index:
src: ./index #函数1入口函数
handler: scf_bootstrap #启动文件
user:
src: ./user #函数2入口函数
handler: scf_bootstrap #启动文件
triggers: #触发器配置
- type: apigw
parameters:
name: serverless
protocols:
- https
- http
apis:
- path: /
method: ANY
# api 的 function 配置优先级高于外层 function
function: index
- path: /user
method: ANY
# api 的 function 配置优先级高于外层 function
function: user
```

3. 完成配置后，在根目录下执行 `sls deploy`，测试项目是否部署成功。

## 应用控制台上架

通过 工单 方式进行提交，注意您的项目需要包含以下内容：

| 参数 | 说明 |
| --- | --- |
| 基本配置参数列表 | 基本配置参数列表 |
| 高级配置参数列表 | 非必填项 |
| 应用名称、简介、文档链接、tag | 用于控制台卡片展示 |

# 基础组件列表

最近更新时间：2021-01-29 17:30:03

Serverless Framework 基础组件的使用说明和全量配置文档如下：

| 组件名称 | 使用说明 | 全量配置 |
| --- | --- | --- |
| SCF 组件 | 使用说明 | serverless.yml 全量配置 |
| Website 组件 | 使用说明 | serverless.yml 全量配置 |
| API 网关组件 | 使用说明 | serverless.yml 全量配置 |
| VPC 组件 | 使用说明 | serverless.yml 全量配置 |
| COS 组件 | 使用说明 | serverless.yml 全量配置 |
| PostgreSQL 组件 | 使用说明 | serverless.yml 全量配置 |
| CynosDB 组件 | 使用说明 | serverless.yml 全量配置 |
| CDN 组件 | 使用说明 | serverless.yml 全量配置 |
| Layer 组件 | 使用说明 | serverless.yml 全量配置 |
| CynosDB 组件 | 使用说明 | serverless.yml 全量配置 |

# 连接 MySQL 数据库

最近更新时间：2021-07-13 17:08:06

## 操作场景

目前，腾讯云原生数据库 TDSQL-C 已支持 Serverless MySQL 版本，做到按实际使用的计算和存储量计费，按秒计量，按小时结算。Serverless Framework 的 CynosDB 组件也已经支持该类型数据库的创建。

本文以 Node.js 开发语言的函数，指导您快速创建 TDSQL-C Serverless MySQL 实例，并在云函数中进行调用。

## 操作步骤

| 操作步骤 | 操作说明 |
| --- | --- |
| 步骤1：配置环境变量 | - |
| 步骤2：配置私有网络 | 通过 Serverless Framework VPC 组件 创建 VPC 和 子网，支持云函数和数据库的网络打通和使用。 |
| 步骤3：配置 Serverless DB | 通过 Serverless Framework Cynosdb 组件创建 MySQL 实例，为云函数项目提供数据库服务。 |
| 步骤4：编写业务代码 | 通过 Serverless DB SDK 调用数据库，云函数支持直接调用 Serverless DB SDK，连接 PostgreSQL 数据库进行管理操作。 |
| 步骤5：部署应用 | 通过 Serverless Framework 部署项目至云端，并通过云函数控制台进行测试。 |
| 步骤6：移除项目（可选） | 可通过 Serverless Framework 移除项目。 |

### 步骤1：配置环境变量

1. 在本地建立目录，用于存放代码及依赖模块。本文以 `test-MySQL` 文件夹为例。

```
mkdir test-MySQL && cd test-MySQL
```

2. 由于目前 TDSQL-C Serverless 只支持 `ap-beijing-3`，`ap-guangzhou-4`，`ap-shanghai-2` 和 `ap-nanjing-1` 四个区域，所以这里还需要配置下，只需要在项目根目录下创建 `.env` 文件，然后配置 `REGION` 和 `ZONE` 两个环境变量：

```
# .env
REGION=xxx
ZONE=xxx
```

### 步骤2：配置私有网络

1. 在 `test-MySQL` 目录下创建文件夹 `VPC` 。

```
mkdir VPC && cd VPC
```

2. 在 `VPC` 中新建 serverless.yml 文件，使用 VPC 组件完成私有网络和子网的创建。
`serverless.yml` 示例内容如下（全量配置参考 产品文档）：

```
#serverless.yml
app: mysql-app
stage: dev
component: vpc # (required) name of the component. In that case, it's vpc.
name: mysql-app-vpc # (required) name of your vpc component instance.
inputs:
region: ${env:REGION}
zone: ${env:ZONE}
vpcName: serverless-mysql
subnetName: serverless-mysql
```

### 步骤3：配置 Serverless DB

1. 在 `test-MySQL` 下创建文件夹 `DB` 。

2. 在 `DB` 文件夹下新建 `serverless.yml` 文件，并输入以下内容，通过 Serverless Framework 组件完成云开发环境配置。
`serverless.yml` 示例内容如下（全量配置参考 产品文档）：

```
# serverless.yml
app: mysql-app
stage: dev
component: cynosdb
name: mysql-app-db
inputs:
region: ${env:REGION}
```

```
zone: ${env:ZONE}
vpcConfig:
vpcId: ${output:${stage}:${app}:mysql-app-vpc.vpcId}
subnetId: ${output:${stage}:${app}:mysql-app-vpc.subnetId}
```

### 步骤4：编写业务代码与配置文件

1. 在 `test-MySQL` 下创建文件夹 `src`，用于存放业务逻辑代码和相关依赖项。

2. 在 `src` 文件夹下创建文件 `index.js`，并输入如下示例代码。在函数中通过 SDK 连接数据库，并在其中完成 MySQL 数据库的调用。

```
exports.main_handler = async (event, context, callback) =&gt; {
var mysql = require('mysql2');
var connection = mysql.createConnection({
host : process.env.HOST,
user : 'root',
password : process.env.PASSWORD
});
connection.connect();
connection.query('SELECT 1 + 1 AS solution', function (error, results, fields)
{
if (error) throw error;
console.log('The solution is: ', results[0].solution);
});
connection.end();
}
```

3. 安装所需依赖模块。

```
npm install mysql2
```

4. 完成业务代码编写和依赖安装后，创建 `serverless.yml` 文件，示例文件如下：

```
app: mysql-app
stage: dev
component: scf
name: mysql-app-scf

inputs:
src: ./
```

```
functionName: ${name}
region: ${env:REGION}
runtime: Nodejs10.15
timeout: 30
vpcConfig:
vpcId: ${output:${stage}:${app}:mysql-app-vpc.vpcId}
subnetId: ${output:${stage}:${app}:mysql-app-vpc.subnetId}
environment:
variables:
HOST: ${output:${stage}:${app}:mysql-app-db.connection.ip}
PASSWORD: ${output:${stage}:${app}:mysql-app-db.adminPassword}
```

### 步骤5：快速部署

完成创建后，项目目录结构如下：

```
./test-MySQL
├── vpc
│ └── serverless.yml # vpc 配置文件
├── db
│ └── serverless.yml # db 配置文件
├── src
│ ├── serverless.yml # scf 组件配置文件
│ ├── node_modules # 项目依赖文件
│ └── index.js # 入口函数
└── .env # 环境变量文件
```

1. 使用命令行在 `test-MySQL` 下，执行以下命令进行部署。

```
sls deploy
```

> 说明：
>
> - 部署时需要扫码授权，如果没有腾讯云账号，请先 注册新账号。
> - 如果是子账号，请参考 子账号权限配置 完成授权。

返回结果如下所示，即为部署成功。

```
mysql-app-vpc:
region: xxx
```

```
  zone: xxx
  vpcId: xxxx-xxx
  ...

  mysql-app-db:
  dbMode: xxxx
  region: xxxx
  zone: xxxx
  ...

  mysql-app-scf:
  functionName: xxxx
  description: xxx
  ...

  59s › test-MySQL › "deploy" ran for 3 apps successfully.
```

2. 部署成功后，您可通过 云函数控制台，查看并进行函数调试。

### 步骤6：移除项目（可选）

在 `test-MySQL` 目录下，执行以下命令可移除项目。

```
sls remove
```

返回如下结果，即为成功移除。

```
serverless ⚡ framework
4s › test-MySQL › Success
```

# 示例代码

## Python

Python 可使用云函数环境已经内置的 **pymysql** 依赖包进行数据库连接。示例代码如下：

```
# -*- coding: utf8 -*-
from os import getenv

import pymysql
from pymysql.err import OperationalError
```

```
mysql_conn = None

def __get_cursor():
try:
return mysql_conn.cursor()
except OperationalError:
mysql_conn.ping(reconnect=True)
return mysql_conn.cursor()

def main_handler(event, context):
global mysql_conn
if not mysql_conn:
mysql_conn = pymysql.connect(
host = getenv('DB_HOST', '<your db="" host="">'),
user = getenv('DB_USER','<your db="" user="">'),
password = getenv('DB_PASSWORD','<your db="" password="">'),
db = getenv('DB_DATABASE','<your db="" database="">'),
port = int(getenv('DB_PORT','<your db="" port="">')),
charset = 'utf8mb4',
autocommit = True
)

with __get_cursor() as cursor:
cursor.execute('select * from employee')
myresult = cursor.fetchall()
print(myresult)
for x in myresult:
print(x)
```

## Node.js

Node.js 支持使用连接池进行连接，连接池具备自动重连功能，可有效避免因云函数底层或者数据库释放连接造成的连接不可用情况。示例代码如下：

> 说明：
> 使用连接池前需先安装 **mysql2** 依赖包，详情请参见 依赖安装。

```
'use strict';

const DB_HOST = process.env[`DB_HOST`]
const DB_PORT = process.env[`DB_PORT`]
const DB_DATABASE = process.env[`DB_DATABASE`]
const DB_USER = process.env[`DB_USER`]
```

```
const DB_PASSWORD = process.env[`DB_PASSWORD`]

const promisePool = require('mysql2').createPool({
host : DB_HOST,
user : DB_USER,
port : DB_PORT,
password : DB_PASSWORD,
database : DB_DATABASE,
connectionLimit : 1
}).promise();

exports.main_handler = async (event, context, callback) =&gt; {
let result = await promisePool.query('select * from employee');
console.log(result);
}
```

### PHP

PHP 可使用 **pdo_mysql** 或 **mysqli** 依赖包进行数据连接。示例代码如下：

- **pdo_mysql**

```
<?php
function handler($event, $context) {
try{
$pdo = new PDO('mysql:host= getenv("DB_HOST");dbname= getenv("DB_DATABASE"),get
env("DB_USER"),getenv("DB_PASSWORD")');
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}catch(PDOException $e){
echo '数据库连接失败: '.$e->getMessage();
exit;
}
}
```

- **mysqli**

```
<?php
function main_handler($event, $context) {
$host = "";
$username = "";
$password = "";
// 创建连接
$conn = mysqli_connect($servername, $username, $password);
// 检测连接
if (!$conn) {
die("Connection failed: " . mysqli_connect_error());
```

```
}
echo "连接成功";
mysqli_close($conn);
echo "断开连接";
}
?>
```

## Java

1. 请参考 依赖安装，安装以下依赖。

```
<dependencies>
<dependency>
<groupid>com.tencentcloudapi</groupid>
<artifactid>scf-java-events</artifactid>
<version>0.0.2</version>
</dependency>
<dependency>
<groupid>com.zaxxer</groupid>
<artifactid>HikariCP</artifactid>
<version>3.2.0</version>
</dependency>
<dependency>
<groupid>mysql</groupid>
<artifactid>mysql-connector-java</artifactid>
<version>8.0.11</version>
</dependency>
</dependencies>
```

2. 使用 Hikari 连接池进行连接，示例代码如下：

```
package example;

import com.qcloud.scf.runtime.Context;
import com.qcloud.services.scf.runtime.events.APIGatewayProxyRequestEvent;
import com.qcloud.services.scf.runtime.events.APIGatewayProxyResponseEvent;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;

public class Http {
private DataSource dataSource;

public Http() {
HikariConfig config = new HikariConfig();
config.setJdbcUrl("jdbc:mysql://" + System.getenv("DB_HOST") + ":"+ System.get
env("DB_PORT") + "/" + System.getenv("DB_DATABASE"));
config.setUsername(System.getenv("DB_USER"));
config.setPassword(System.getenv("DB_PASSWORD"));
config.setDriverClassName("com.mysql.jdbc.Driver");
config.setMaximumPoolSize(1);
dataSource = new HikariDataSource(config);
}

public String mainHandler(APIGatewayProxyRequestEvent requestEvent, Context co
ntext) {
System.out.println("start main handler");
System.out.println("requestEvent: " + requestEvent);
System.out.println("context: " + context);

try (Connection conn = dataSource.getConnection(); PreparedStatement ps = con
n.prepareStatement("SELECT * FROM employee")) {
ResultSet rs = ps.executeQuery();
while (rs.next()) {
System.out.println(rs.getInt("id"));
System.out.println(rs.getString("first_name"));
System.out.println(rs.getString("last_name"));
System.out.println(rs.getString("address"));
System.out.println(rs.getString("city"));
}
} catch (SQLException e) {
e.printStackTrace();
}

APIGatewayProxyResponseEvent apiGatewayProxyResponseEvent = new APIGatewayProx
yResponseEvent();
apiGatewayProxyResponseEvent.setBody("API GW Test Success");
apiGatewayProxyResponseEvent.setIsBase64Encoded(false);
apiGatewayProxyResponseEvent.setStatusCode(200);

Map<string, string=""> headers = new HashMap&lt;&gt;();
headers.put("Content-Type", "text");
headers.put("Access-Control-Allow-Origin", "*");
```

```
apiGatewayProxyResponseEvent.setHeaders(headers);

return apiGatewayProxyResponseEvent.toString();
}
}
```

## SCF DB SDK for MySQL

为了方便使用，云函数团队将 Node.js 和 Python 连接池相关代码封装为 SCF DB SDK for MySQL，请参考 依赖安装 进行安装使用。通过该 SDK，您可以在云函数代码中连接 MySQL、TDSQL-C 或 TDSQL MySQL版 数据库，并实现对数据库的插入、查询等操作。

SCF DB SDK for MySQL 具备以下特点：

- 自动从环境变量初始化数据库客户端。
- SDK 会在全局维护一个数据库长连接，并处理连接中断后的重连。
- 云函数团队会持续关注 issue，确保获得连接即可用，不需要关注数据库连接。

### 1. Node.js SDK

```
'use strict';
const database = require('scf-nodejs-serverlessdb-sdk').database;

exports.main_handler = async (event, context, callback) =&gt; {
let pool = await database('TESTDB2').pool()
pool.query('select * from coffee',(err,results)=&gt;{
console.log('db2 callback query result:',results)
})
// no need to release pool

console.log('db2 query result:',result)
}
```

> 说明：
> Node.js SDK 具体使用方法请参考 SCF DB SDK for MySQL。

### 2. Python SDK

```
from serverless_db_sdk import database

def main_handler(event, context):
```

```
print('Start Serverlsess DB SDK function')

connection = database().connection(autocommit=False)
cursor = connection.cursor()

cursor.execute('SELECT * FROM name')
myresult = cursor.fetchall()

for x in myresult:
print(x)
```

# 快速部署 Web 函数

最近更新时间：2022-04-28 11:55:20

## 操作场景

Web 函数是腾讯云云函数 SCF 新支持的函数能力，区别于事件函数（Event Function）对于事件格式的限制，该类型函数专注于优化 Web 服务场景，用户可以直接发送 HTTP 请求到 URL 触发函数执行，详情请参见 函数概述。

Serverless Framework SCF 组件现已支持 Web 类型函数部署，您可以通过 SCF 组件，快速创建与部署 Web 函数。

## 操作步骤

1. 执行以下命令，初始化 Serverless Web 函数模板。

```
sls init http-demo
```

2. 进入示例项目，查看目录结构。示例如下：

```
. http-demo
├── serverless.yml # 配置文件
├── package.json # 依赖项文件
├── scf_bootstrap # 项目启动文件
└── index.js # 服务函数
```

其中 `scf_bootstrap` 为项目启动文件，具体编写规则请参见 启动文件说明。

3. 打开 `serverless.yml` ，查看配置信息。

您只需要在 `yml` 里新增 `type` 参数，指定函数类型，即可完成 Web 类型函数部署。

注意

- 对于 Web 类型函数，无需再指定入口函数。
- 不填 `type` 参数时，默认为事件型函数。
- 如果本地代码里无 `scf_bootstrap` 启动文件，您可以在 `yml` 里指定 `entryFile` 参数指定入口函数，组件会根据运行语言，为您生成默认 `scf_bootstrap` 启动文件完成部署。部署完成后，需根据您的实际项目情况，在 云函数控制台 修改 `scf_bootstrap` 文件内容。

示例 `yml` 如下:

```
component: scf
name: http
inputs:
src:
src: ./
exclude:
- .env
# 指定 SCF 类型为 Web 类型
type: web
name: web-function
region: ap-guangzhou
runtime: Nodejs12.16
# 对于 Node.js，可以支持打开自动安装依赖
installDependency: true
events:
- apigw:
parameters:
protocols:
- http
- https
environment: release
endpoints:
- path: /
method: ANY
```

4. 在根目录下执行 `sls deploy` 命令，即可完成服务部署。示例如下：

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

```
serviceId: service-xxxxxx
serviceName: serverless
subDomain: service-xxxxxx.cd.apigw.tencentcs.com
protocols: http&https
environment: release
apiList:
-
path: /
method: ANY
apiName: index
created: true
authType: NONE
businessType: NORMAL
isBase64Encoded: false
apiId: api-xxxxxx
internalDomain:
url: https://service-xxxx.cd.apigw.tencentcs.com/release/
18s › http › 执行成功
```

# 相关命令

### 查看访问日志

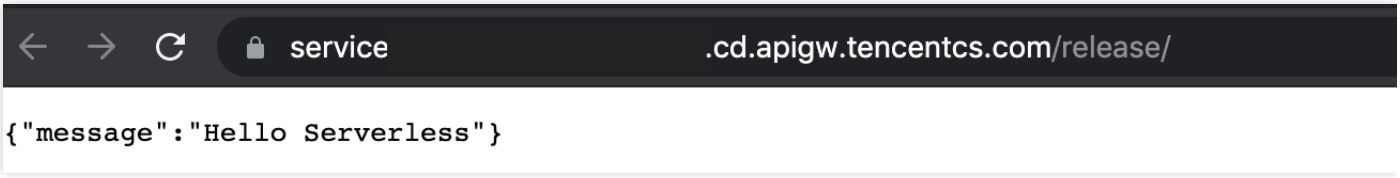
与事件型函数相同，可直接通过 `sls log` 命令查看部署完成的函数最近10条日志信息。示例如下：

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

```
"""
"""
```

### 测试服务

- 方案1：在浏览器直接打开输出的路径 URL，如果可以正常访问，则说明函数创建成功。如下图所示：




- 方案2：您可以使用其他 HTTP 测试工具，例如 CURL、POSTMAN 等工具测试您已创建成功的 Web 函数。如下示例为通过 CURL 工具测试：

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

### 删除服务

执行以下命令，即可移除您已部署的云上资源。

```
sls remove
```

### Web 框架迁移

Serverless Framework Cli 还提供了专门针对 Web 框架部署的 HTTP 组件，快速实现 Web 框架部署、创建层、静态资源分离、CDN 加速等功能，使用方式请参考 通过命令行完成框架部署。