Serverless 应用中心

最佳实践

产品文档

腾讯云

文档目录

最佳实践
开发上线 Serverless 应用
开发项目

最近更新时间：2021-06-23 15:55:15

操作场景

本文以 tencent-express 组件部署一个 Express 网站为例， 模拟 Serverless Framework 开发项目、管理项目和部署发布上线全流程。示例链接 >>

开发项目过程可能会涉及以下分支：

分支类型	说明
master	用于生产环境部署
testing	用于测试环境测试
dev	用于日常开发
feature-xxx	用于增加一个新功能，例如不同开发者会从 dev 拉取不同的特性分支进行开发
hotfix-xxx	用于修复一个紧急 bug

操作步骤

初始化项目

1.参考 部署 Express.js 应用 文档，创建一个 express 项目，修改 yml 文件为以下内容：

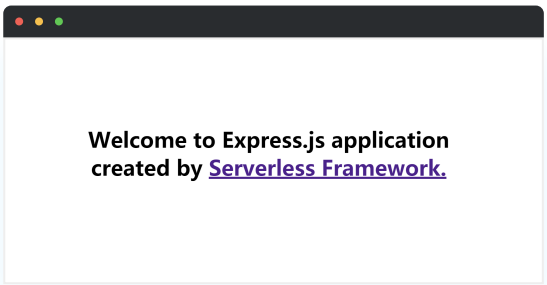
```
#serverless.yml
app: expressDemoApp # 应用名称，默认为与组件实例名称
stage: ${env:STAGE} # 用于开发环境的隔离，默认为dev

component: express # (必填) 引用 component 的名称，当前用到的是 express-tencent 组件
name: expressDemo # (必填) 组件创建的实例名称
inputs:
src:
src: ./
```

```
exclude:
- .env
region: ap-guangzhou
runtime: Nodejs10.15
functionName: ${name}-${stage}-${app} #云函数名称
apigatewayConf:
protocols:
- http
- https
environment: release
```

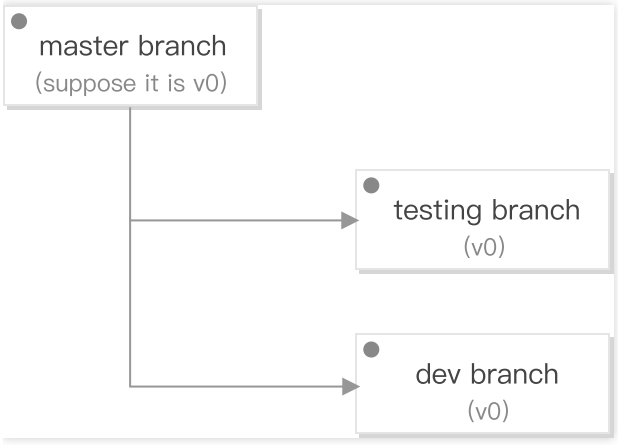
2.在项目根目录下的 .env 文件中配置：

```
TENCENT_SECRET_ID=xxxxxxxxxx #您账号的 SecretId
TENCENT_SECRET_KEY=xxxxxxxx #您账号的 SecretKey
STAGE=prod #STAGE为prod环境，也可以sls deploy --stage prod 参数传递的方式设置
```

3.执行 sls deploy 部署成功后，访问生成的 url 链接，效果如下：



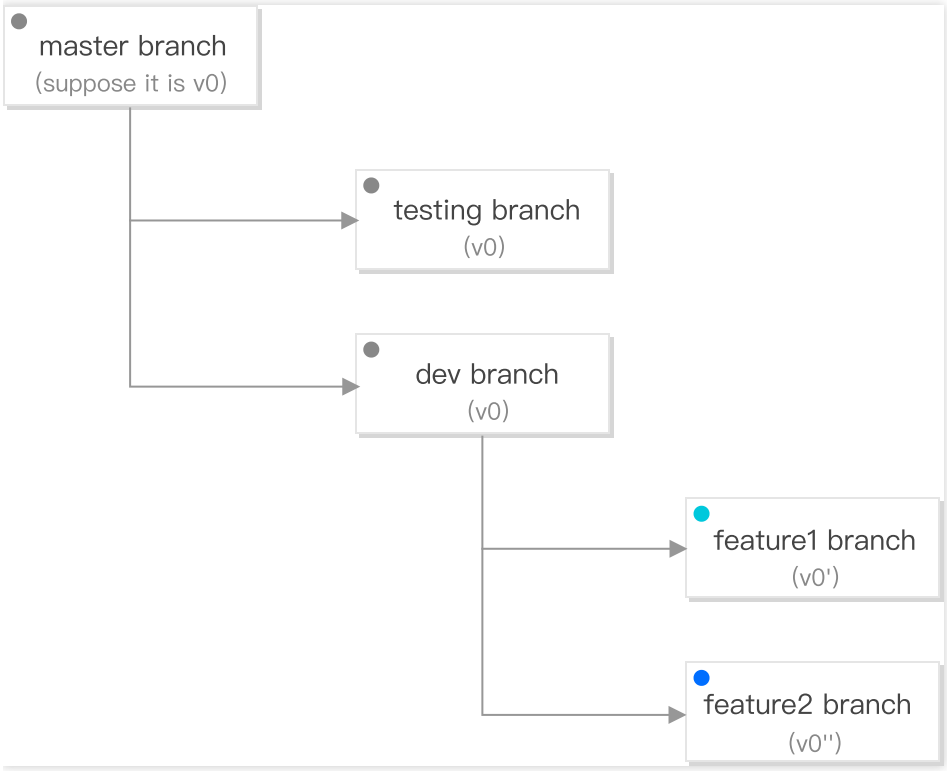
4.创建远程仓库（示例链接），将项目代码提交到远程 master 分支。同时创建 testing、dev。此时三个分支的代码在同一个版本上（假设为版本0）。



开发与测试

背景

现在需要开发某个功能模块。假设需要有两位开发者：Tom、Jorge。两位开发者分别从 dev（版本0）上创建特性分支为 feature1、feature2 进行研发。



Tom 开始开发 feature1。在本示例中，为新增一个 feature.html，里面写文案"This is a new feature 1."。

开发

1.在 sls.js 文件中新增路由器配置：

```
// Routes
app.get(`/feature`, (req, res) => {
res.sendFile(path.join(__dirname, 'feature.html'))
})
```

2.新增 feature.html：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Serverless Component - Express.js</title>
</head>
```

```
<body>
<h1>
This is a new feature 1.
</h1>
</body>
</html>
```

3.在 .env 文件中设置自己的 stage，以便在开发过程中得到独立的运行和调试环境。

例如 Tom 在 serverless.yml 的项目目录下配置 .env 如下：

```
TENCENT_SECRET_ID=xxxxxxxxxx
TENCENT_SECRET_KEY=xxxxxxxx
STAGE=feature1
```

4.执行 sls deploy 部署成功后，返回显示如下：

```
region: ap-guangzhou
apigw:
serviceId: service-xxxxxx
subDomain: service-xxxxxx-123456789.gz.apigw.tencentcs.com
environment: release
url: https://service-xxxxxx-123456789.gz.apigw.tencentcs.com/release/
scf:
functionName: express-demo-feature1
runtime: Nodejs10.15
namespace: default
lastVersion: $LATEST
traffic: 1
Full details: https://serverless.cloud.tencent.com/instances/expressDemoApp%3Afeature1%3AexpressDemo
10s » expressDemo » Success
```

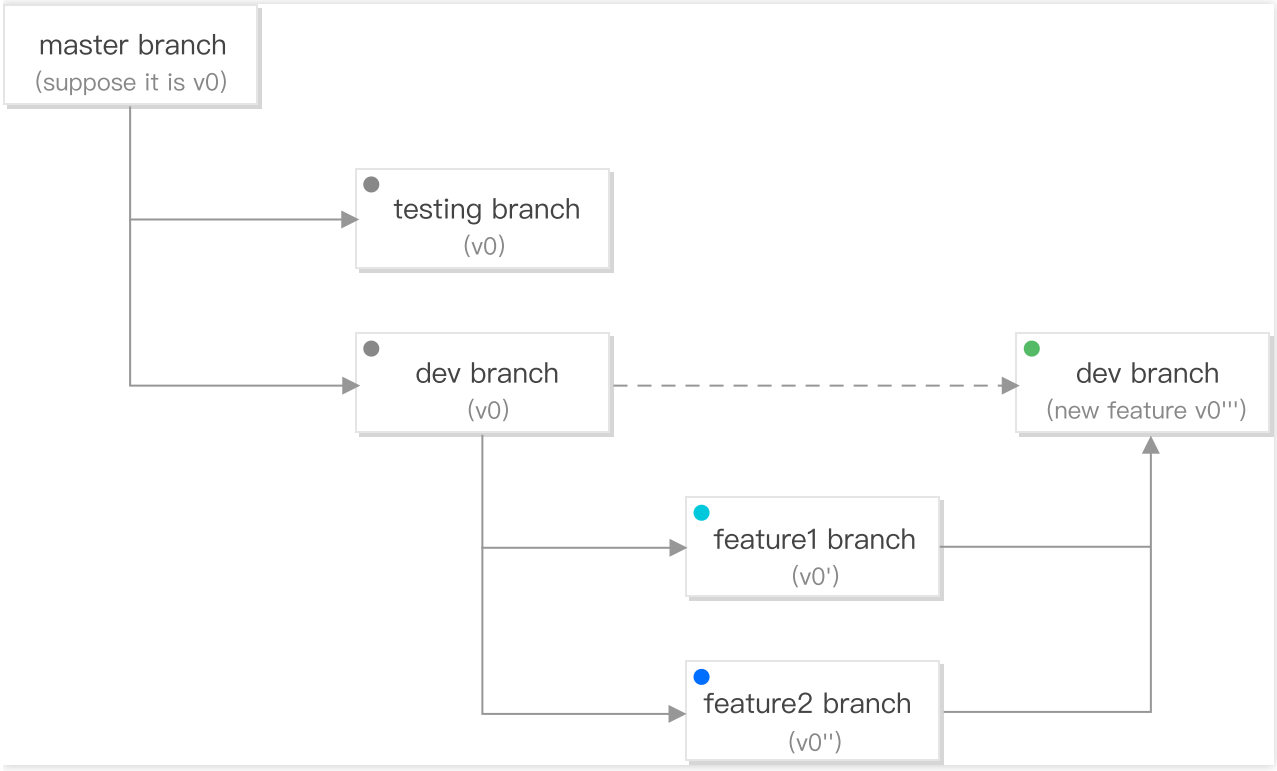
5.访问生成的 url（https://service-xxxxxx-123456789.gz.apigw.tencentcs.com/release/feature），效果如下：

至此，Tom 开发功能完成并自测通过。

假设同时，Jorge 同时也完成自己的特性开发，并自测通过。在本示例中，为新增一个 feature.html，里面写文案"This is a new feature 2."。

联调

1、两人把各自 feature 分支的代码合并到 dev 分支。（可能会存在冲突需要人为解决）



2、在dev进行联调。联调环境中的.env配置如下

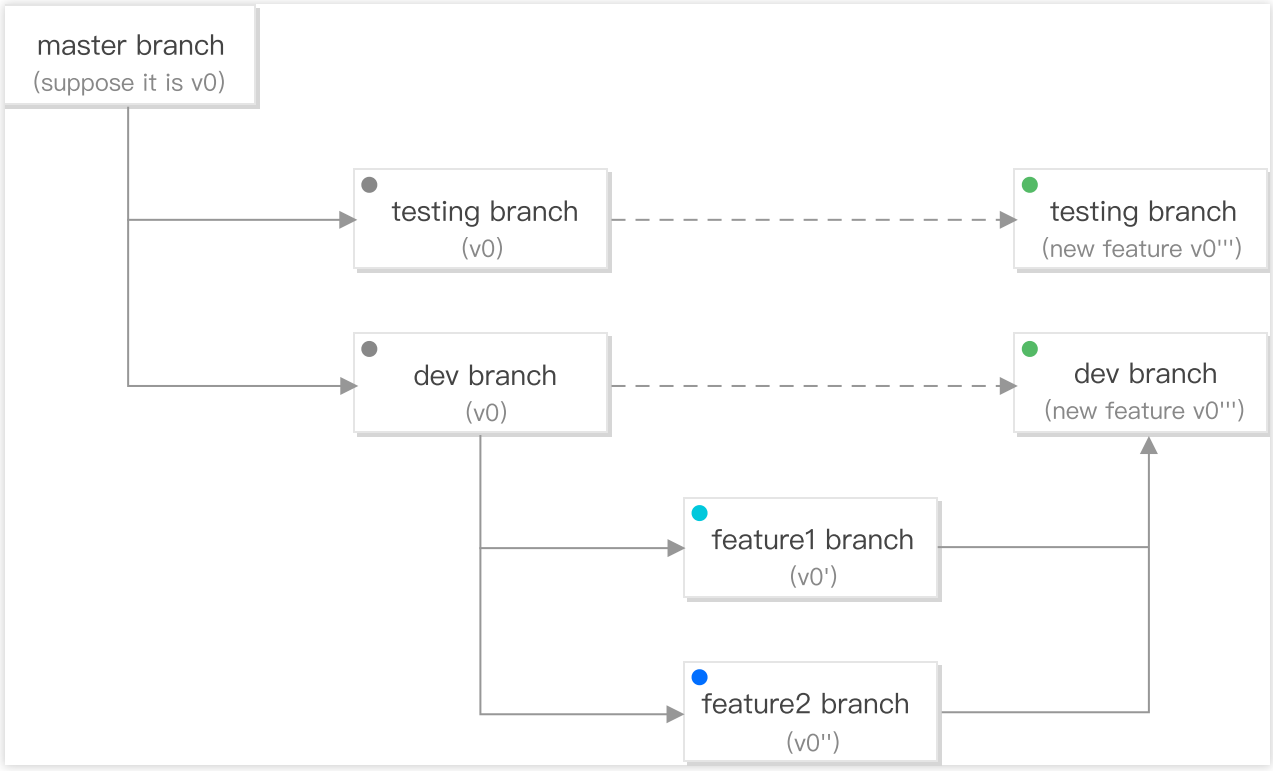
```
TENCENT_SECRET_ID=xxxxxxxxxx
TENCENT_SECRET_KEY=xxxxxxxx
STAGE=dev
```

3、执行 sls deploy 联调部署后，访问 url（https://service-xxxxxx-123456789.gz.apigw.tencentcs.com/release/feature），效果如下：

至此联调完成，整个功能已经开发完毕。

测试

1. 把联调通过的 dev 分支合并到 testing 代码，进入测试。



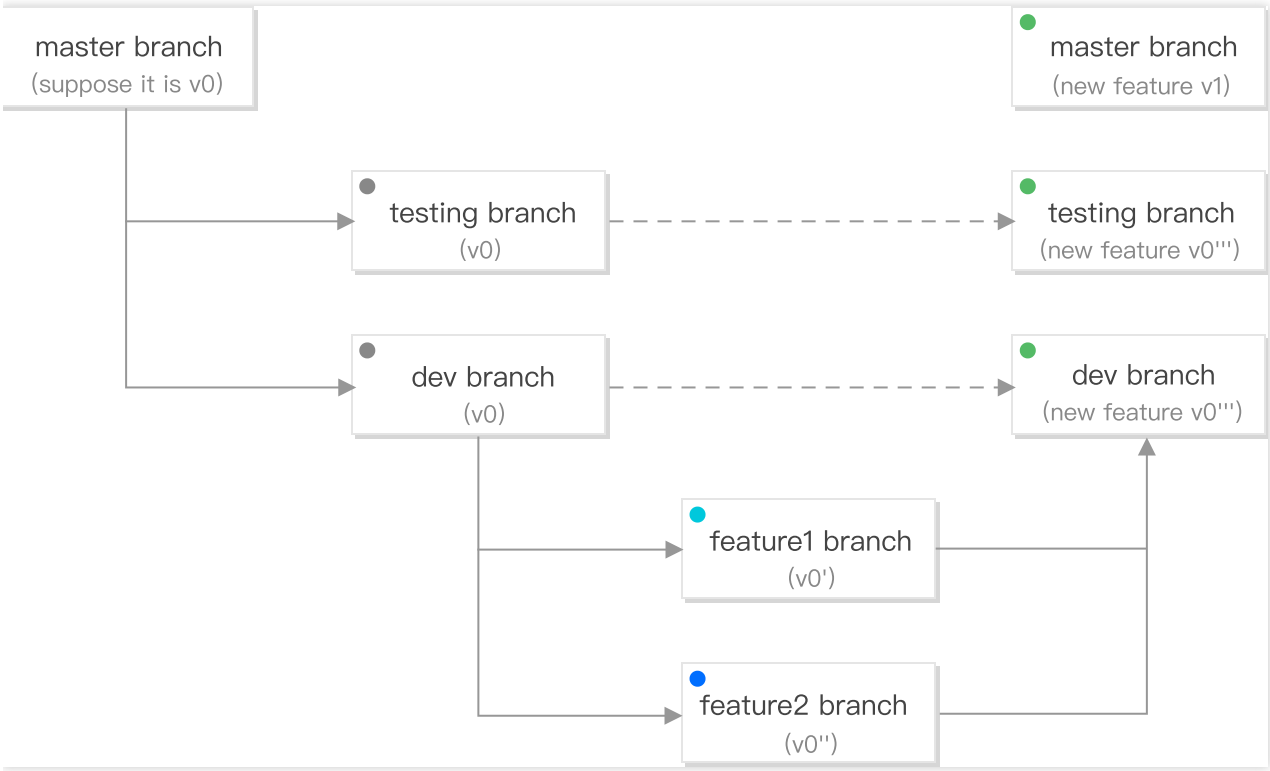
2. 测试环境中的 .env 配置如下：

```
TENCENT_SECRET_ID=xxxxxxxxxx
TENCENT_SECRET_KEY=xxxxxxxx
STAGE=testing
```

3. 执行 sls deploy 部署成功后，测试人员开始进行相关测试，直至功能稳定通过。

发布上线

测试通过后，将测试代码合并到 master 分支，准备发布上线。



设置生产环境中的 .env 为：

```
TENCENT_SECRET_ID=xxxxxxxxxx
TENCENT_SECRET_KEY=xxxxxxxx
STAGE=prod
```

执行部署命令：

```
sls deploy
```

至此，我们完成了一个 severless-express 项目的开发和上线发布。

灰度发布

最近更新时间：2020-12-02 16:17:49

操作场景

在业务进行版本更新及切换时，为了保证线上业务稳定，建议采取灰度发布的方式。本文以已部署的 express 项目为例，为您介绍两种灰度发布的操作步骤。

前提条件

已完成 开发项目。

操作步骤

1. 设置生产环境中的 .env：

```
TENCENT_SECRET_ID=xxxxxxxxxx
TENCENT_SECRET_KEY=xxxxxxxx
STAGE=prod
```

2. 部署到线上环境 $latest，并切换10%的流量在 $latest 版本（90%的流量在最后一次发布的云函数版本N上）：

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

3. 对 $latest 版本进行监控与观察，等版本稳定之后把流量100%切到该版本上：

```
sls deploy --inputs traffic=1.0
```

4. 流量全部切换成功后，对于一个稳定版本，我们需要对它进行标记，以免后续发布新功能时，如果遇到线上问题，方便快速回退版本。部署并发布函数版本 N+1，切换所有流量到版本 N+1：

```
sls deploy --inputs publish=true traffic=0
```

> **说明：**
>
> 云函数组件支持了自定义别名的灰度发布，可以在任意两个函数版本间进行流量规则配置，详细说明请参考Serverless 灰度发布。

自动化部署

最近更新时间：2022-06-13 15:10:22

操作场景

在 Serverless 应用开发中，我们需要手动执行部署命令将开发项目部署到云端。通过引入一些 CI 能力进行 Serverless 应用的自动化部署。

基于 GitHub 的自动化部署

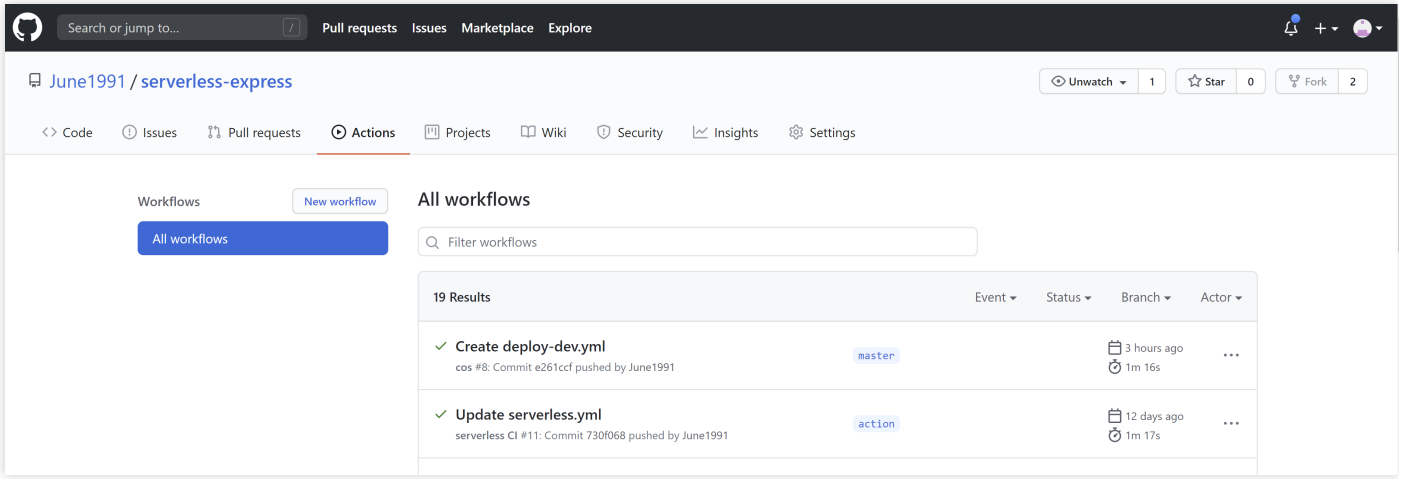
前提条件

- 已创建 Serverless 应用项目。参考 开发项目 创建您的 Serverless 项目并创建各个环境与分支。
- 已托管您的 Serverless 项目到 Github。

操作步骤

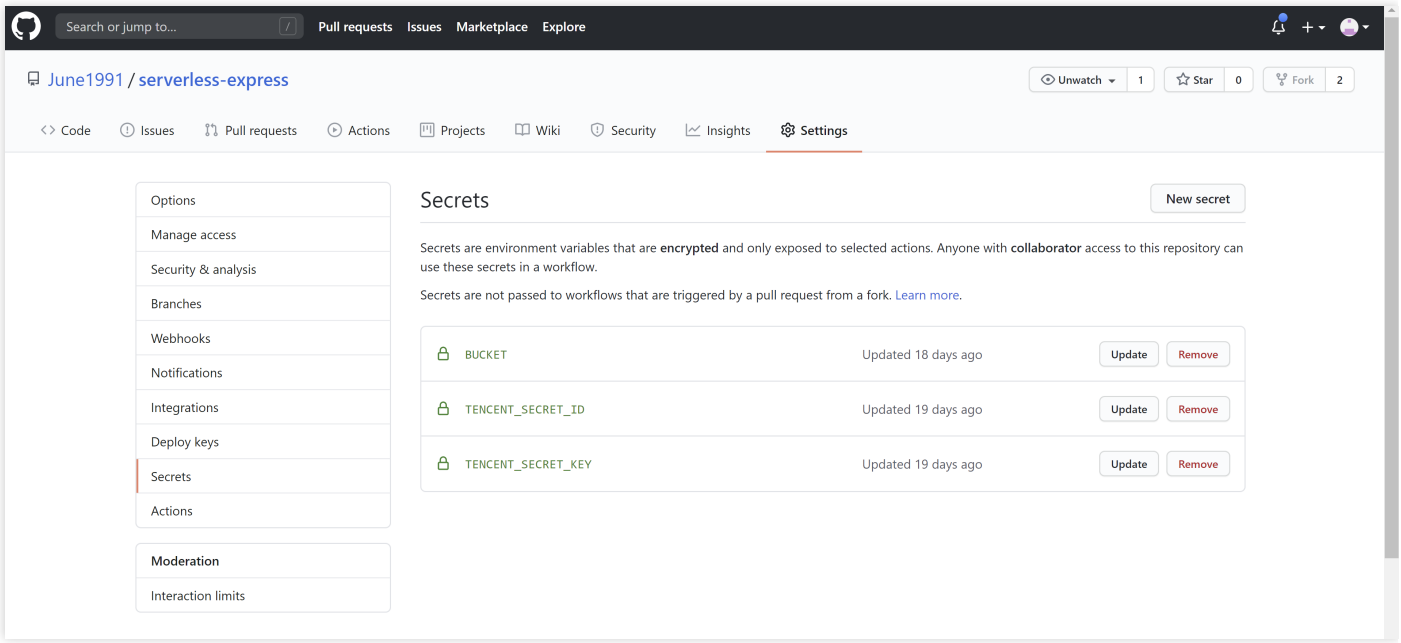
在开发测试阶段，为了方便开发、测试和调试，希望代码每次提交后进行自动化部署。操作如下：

1. 选取一个您需要执行自动化部署的分支（本示例选择 dev 分支）。
2. 在该分支下创建您的 action。



> 注意：
> GitHub 规定如果事件发生在特定仓库分支上，则工作流程文件必须存在于该分支的仓库中 。

3. 配置腾讯云密钥。



4. 配置 action 部署步骤。

```
# 当代码推动到 dev 分支时，执行当前工作流程
# 更多配置信息：https://docs.github.com/cn/actions/getting-started-with-github-actions
name: deploy serverless
on: #监听的事件和分支配置
push:
branches:
- dev
jobs:
test: #配置单元测试
name: test
runs-on: ubuntu-latest
steps:
- name: unit test
run: ''
deploy:
name: deploy serverless
runs-on: ubuntu-latest
needs: [test]
steps:
- name: clone local repository
uses: actions/checkout@v2
- name: install serverless
run: npm install -g serverless
- name: install dependency
run: npm install
```

```
- name: build
run: npm build
- name: deploy serverless
run: sls deploy --debug
env: # 环境变量
STAGE: dev #您的部署环境
SERVERLESS_PLATFORM_VENDOR: tencent #serverless 境外默认为 aws，配置为腾讯
TENCENT_SECRET_ID: ${{ secrets.TENCENT_SECRET_ID }} #您的腾讯云账号 sercret ID
TENCENT_SECRET_KEY: ${{ secrets.TENCENT_SECRET_KEY }} #您的腾讯云账号 sercret key
```

完成上述配置后，开发者每次提交代码到 dev 分支时，就会自动部署。

基于 Coding 的自动化部署

前提条件

- 已开通 Coding 账号。腾讯云用户可以通过 CODING DevOps 快速开通。
- 已创建 Serverless 应用项目。如果您未创建 Serverless 应用项目，请参考 开发项目 创建您的 Serverless 项目并创建各个环境与分支。
- 已托管您的 Serverless 项目到 Coding/Github/Gitlab/码云。

操作场景

在开发测试阶段，为了方便开发、测试和调试，希望代码每次提交后进行自动化部署。操作如下：

1. 创建您的 Coding Devops 项目。
2. 创建一个构建计划，选择自定义构建过程。
3. 配置构建计划。

   i. 基础信息配置。本例中配置 Github 仓库：June1991/express-demo。

   ii. 触发规则配置。本例中配置代码推送到 dev 分支时触发构建。

   iii. 环境变量配置。本例中配置 STAGE 变量为部署环境 dev，TENCENT_CLOUD_API_CRED 为腾讯云账号密钥（密钥配置路径：左下角项目设置 > 开发者选项 > 凭据管理 > 录入凭据 > 腾讯云 API 密钥）。

   iv. 流程配置。

```
pipeline {
agent any
```

```
stages {
stage('检出') {
steps {
checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_I
D]]])
}
}
stage('安装依赖') {
steps {
echo '安装依赖中...'
sh 'npm i -g serverless'
sh 'npm install'
echo '安装依赖完成.'
}
}
stage('部署') {
steps {
echo '部署中...'

withCredentials([
cloudApi(
credentialsId: "${env.TENCENT_CLOUD_API_CRED}",
secretIdVariable: 'TENCENT_SECRET_ID',
secretKeyVariable: 'TENCENT_SECRET_KEY'
),
]) {

// 生成凭据文件
sh 'echo "TENCENT_SECRET_ID=${TENCENT_SECRET_ID}\nTENCENT_SECRET_KEY=${TENCEN
T_SECRET_KEY}" > .env'
// 部署
sh 'sls deploy --debug'
// 移除凭据
sh 'rm .env'
}

echo '部署完成'
}
}
}
}
```

完成以上配置后，开发者每次提交代码到 dev 分支时，就会自动部署。

部署 Hexo 博客

最近更新时间：2021-03-29 15:51:16

操作场景

该任务指导您通过 Serverless Website 组件，快速构建一个 Serverless Hexo 站点。

前提条件

- 已安装 Node.js（**2020年9月1日起，Serverless 组件不再支持 Node.js10.0 以下版本，请注意升级**）
- 已安装 Git

如您未安装上述应用程序，可以参考 Hexo 安装说明。

操作步骤

1. 安装

通过 npm 安装 Serverless Framework：

```
$ npm install -g serverless
```

通过 npm 安装 Hexo：

```
$ npm install -g hexo-cli
```

安装 Hexo 完成后，请执行下列命令，Hexo 将会在指定文件夹中新建所需要的文件。

```
$ hexo init hexo # 生成 Hexo 目录
$ cd hexo
$ npm install
```

新建完成后，指定文件夹的目录如下：

```
.
├── _config.yml
├── package.json
├── scaffolds
```

```
├── source
|  ├── _drafts
|  └── _posts
└── themes
```

安装完成后，可以通过 `hexo g` 命令生成静态页面：

```
$ hexo g # generate
```

说明：

如果希望在本地查看效果，也可以运行下列命令，通过浏览器访问 `localhost:4000` 查看页面效果。

```
$ hexo s # server
```

2. 配置

在 `hexo` 目录下，创建 `serverless.yml` 文件：

```
$ touch serverless.yml
```

在 `serverless.yml` 文件中进行如下配置：

```
# serverless.yml

component: website # (必填) 引用 component 的名称，当前用到的是 tencent-website 组件
name: hexodemo # (必填) 该 website 组件创建的实例名称

app: websiteApp # (可选) 该 website 应用名称
stage: dev # (可选) 用于区分环境信息，默认值是 dev

inputs:
src:
src: ./public # Upload static files generated by HEXO
index: index.html
# dist: ./dist
# hook: npm run build
# websitePath: ./
region: ap-guangzhou
bucketName: my-bucket
protocol: https
```

配置完成后，文件目录如下：

```
.
├── .serverless
└── hexo
├── public
├── ...
├── serverless.yml
├── ...
└── source
```

3. 部署

通过 `sls deploy` 命令进行部署，并可以添加 `--debug` 参数查看部署过程中的信息。

```
$ sls deploy

serverless ⚡ framework
Action: "deploy" - Stage: "dev" - App: "websiteApp" - Instance: "hexodemo"

region: ap-guangzhou
website: https://my-bucket-1258834142.cos-website.ap-guangzhou.myqcloud.com


25s › hexodemo › Success
```

访问命令行输出的 Website URL，即可查看您的 Serverless Hexo 站点。

> ⚠ **注意：**
>
> 如果希望更新 Hexo 站点中的文章，需要在本地重新运行 `hexo g` 进行生成静态页面，再运行 `serverless` 更新到页面。

4. 移除

通过以下命令移除 Hexo 网站：

```
$ sls remove --debug

DEBUG - Flushing template state and removing all components.
DEBUG - Starting Website Removal.
DEBUG - Removing Website bucket.
DEBUG - Removing files from the "my-bucket-1250000000" bucket.
DEBUG - Removing "my-bucket-1250000000" bucket from the "ap-guangzhou" region.
```

```
DEBUG — "my-bucket-1250000000" bucket was successfully removed from the "ap-guang
zhou" region.
DEBUG — Finished Website Removal.

6s » myWebsite » done
```

账号配置（可选）

当前默认支持 CLI 扫描二维码登录，如您希望配置持久的环境变量/密钥信息，也可以本地创建 `.env` 文件：

```
$ touch .env # 腾讯云的配置信息
```

在 `.env` 文件中配置腾讯云的 SecretId 和 SecretKey 信息并保存：

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

说明：

- 如果没有腾讯云账号，请先 注册新账号。
- 如果已有腾讯云账号，可以在 API 密钥管理 中获取 SecretId 和 SecretKey。

部署 Vue + Express + PostgreSQL 全栈网站

最近更新时间：2022-06-13 15:10:22

操作场景

该模板可以快速部署一个基于 Vue + Express + PostgreSQL 的全栈 Serverless 应用。主要包含以下组件：

- Serverless RESTful API：通过**云函数**和 **API 网关**构建的 Express 框架实现 RESTful API。
- Serverless 静态网站：前端通过托管 Vue.js 静态页面到 **COS 对象存储**中。
- PostgreSQL Serverless：通过创建 **PostgreSQL DB** 为全栈网站提供数据库服务。
- VPC：通过创建 **VPC** 和 **子网**，提供 SCF 云函数和数据库的网络打通和使用。

前提条件

- 已安装 Node.js（**2020年9月1日起，Serverless 组件不再支持 Node.js10.0 以下版本，请注意升级**）
- 账号已经配置 **QcloudPostgreSQLFullAccess** 策略，配置方法详见 账号和权限配置

操作步骤

安装

通过 npm 全局安装 Serverless Framework：

```
npm install -g serverless
```

如果之前您已经安装过 Serverless Framework，可以通过下列命令升级到最新版：

```
npm update -g serverless
```

安装完毕后，通过运行 serverless -v 命令，查看 Serverless Framework 的版本信息，确保版本信息不低于以下版本。返回结果如下所示：

```
$ serverless -v
Framework Core: 1.74.1 (standalone)
Plugin: 3.6.14
SDK: 2.3.1
Components: 2.31.6
```

配置

1. 新建一个本地文件夹，使用 `serverless init` 命令，下载相关 template。

```
serverless init fullstack
```

2. 在项目根目录下新建 .env 文件，并在其中配置对应的腾讯云 SecretId、SecretKey、地域和可用区信息。

```
# .env
TENCENT_SECRET_ID=xxx // 您账号的 SecretId
TENCENT_SECRET_KEY=xxx // 您账号的 SecretKey
# 地域可用区配置
REGION=ap-guangzhou //资源部署区，该项目中指云函数与静态页面部署区
ZONE=ap-guangzhou-2 //资源部署可用区 ，该项目中指 DB 部署所在的可用区
```

> 说明：
>
> - 如果没有腾讯云账号，请先 注册新账号。
> - 如果已有腾讯云账号，请保证您的账号已经授权了 AdministratorAccess 权限。 您可以在 API 密钥管理 中获取 SecretId 和 SecretKey。
> - ZONE 目前只支持 ap-beijing-3 、ap-guangzhou-2、ap-shanghai-2。

3. 通过执行以下命令，安装所需依赖：

```
npm run bootstrap
```

部署

1. 执行 `sls deploy --all` 命令进行部署。返回信息如下所示：

```
$ sls deploy --all

serverless ⚡ framework

serverlessVpc:
region: ap-guangzhou
zone: ap-guangzhou-2
vpcId: vpc-xxx
```

```
vpcName: serverless
subnetId: subnet-xxx
subnetName: serverless

fullstackDB:
region: ap-guangzhou
zone: ap-guangzhou-2
vpcConfig:
subnetId: subnet-100000
vpcId: vpc-1000000
dBInstanceName: fullstackDB
dBInstanceId: postgres-100000
private:
connectionString: postgresql://tencentdb_100000xxxxxxxxxxxxx@172.16.250.15:543
2/tencentdb_1000000
host: 172.16.250.15
port: 5432
user: tencentdb_100000
password: xxxxxxxx
dbname: tencentdb_100000

fullstack-api:
region: ap-guangzhou
apigw:
serviceId: service-100000
subDomain: service-100000-123456789.gz.apigw.tencentcs.com
environment: release
url: https://service-100000-123456789.gz.apigw.tencentcs.com/release/
scf:
functionName: fullstack-api
runtime: Nodejs10.15
namespace: default

fullstack-frontend:
website: https://fullstack-serverless-db-123456789.cos-website.ap-guangzhou.my
qcloud.com

50s › tencent-fullstack › Success
```

部署成功后，您可以使用浏览器访问项目产生的 website 链接，即可看到生成的网站。

> 说明：
> 本项目云函数因 VPC，导致无法直接访问外网，如需访问外网请参考 云函数网络配置。

2. 执行 `sls remove --all`，可移除项目。返回信息如下所示：

```
$ sls remove --all
serverless ⚡ framework
38s › tencent-fullstack › Success
```

部署互动直播间语音识别服务

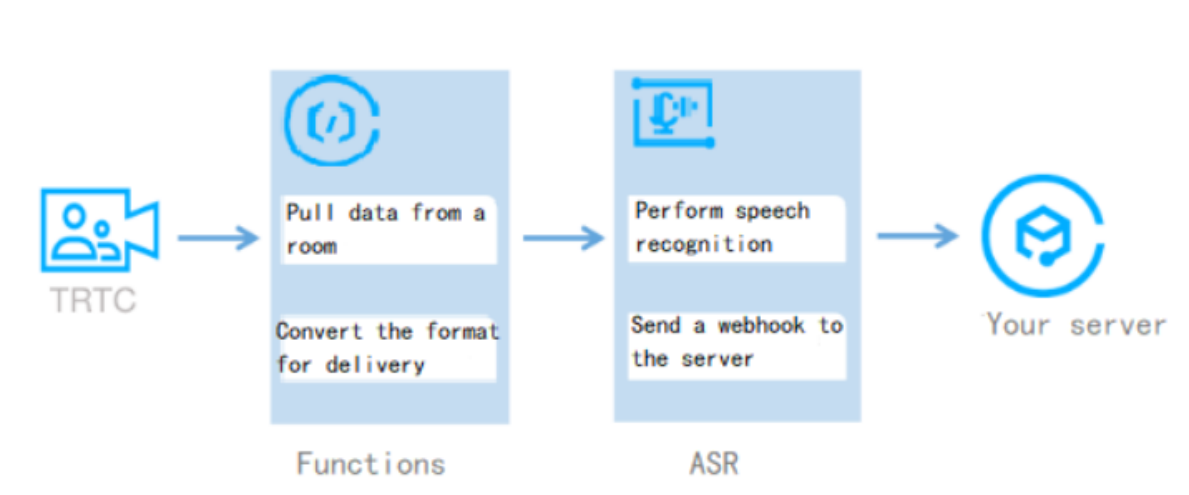
最近更新时间：2023-05-05 14:56:19

使用场景

你画我猜：可以实时拉取房间内某个用户的音频进行实时识别，转换成文本之后回调给客户的业务服务器，进行业务逻辑判断
语音审核：和业务关联比较多的语音审核，可以采取该接口将数据流投递到语音识别接口进行语音识别，然后进行关键词过滤。
实时字幕：可以通过该接口实时识别房间音频数据，形成文本，在前端做呈现。

架构原理

具体流程如下图所示：



应用优势

实时返回：可以将 trtc 房间的语音数据实时识别返回，快速高效。
流程简单：深度融合 trtc 和 asr，数据流完全打通，不需要复杂接入流程。
使用灵活：数据返回给业务服务器之后，可以和业务逻辑实时关联。

注意事项

一般情况下，语音识别的处理时间较长是由于部署函数时开启了 异步执行。
目前识别的结果将下发至业务服务器。暂不支持 websocket 的形式，无法下发至客户端。
默认的鉴权形式是 应用鉴权，测试时可更改为 无鉴权。

使用流程

1. 开通服务

您需要开启腾讯云语音识别服务，操作详情见 开通语音识别服务。

2. 部署函数服务

1. 登录 Serverless 应用控制台。
2. 单击**新建应用**，进入“新建应用”页面。
3. 选择**直播房间实时语音识别服务**，并进行**基础配置**。

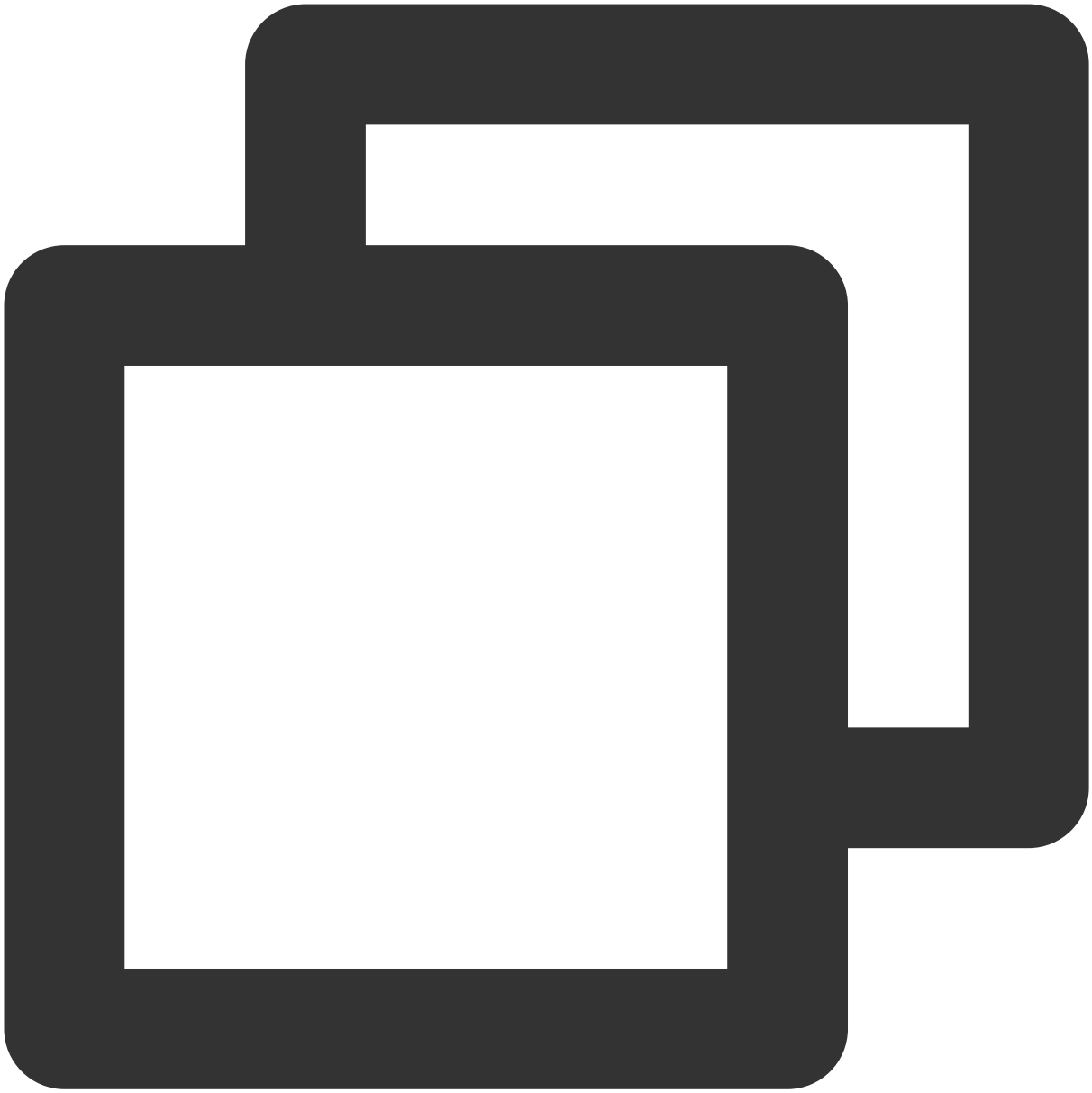
应用名：自定义名称。

地域：根据实际情况进行选择。

密钥信息：您可在 API密钥管理 中查看腾讯云账号密钥信息。

4. 单击**完成**即可。
5. 单击**云函数** > **函数名称**进入函数详情页，在“触发管理”中获取访问路径。

3. 语音识别启动接口



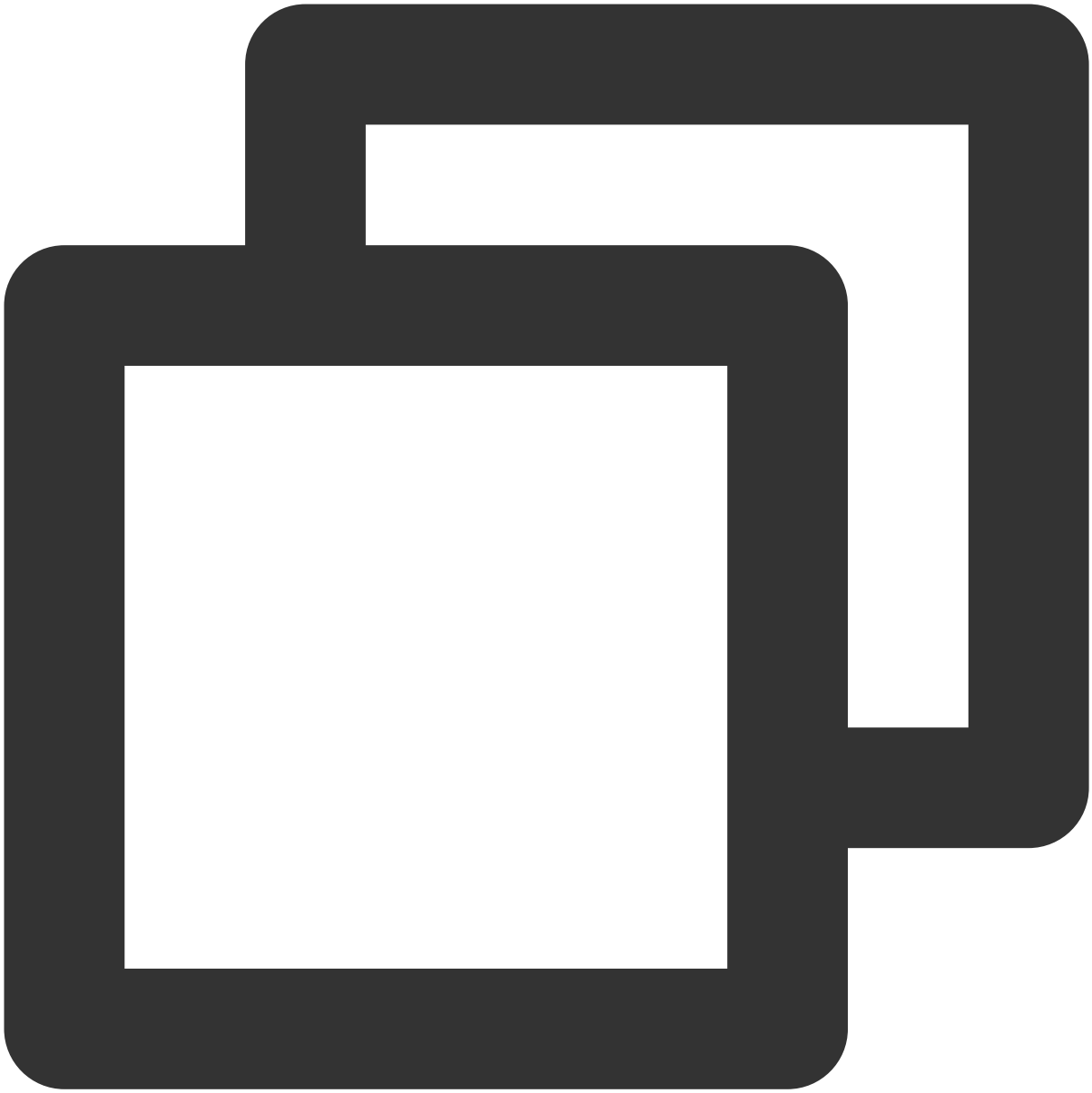
```
proto: HTTPS
Method: POST
URL: https://service-xxx-xxxx.sh.apigw.tencentcs.com/release/asr_speech
```

请求参数：

参数	类型	必填	说明
SdkAppId	Int	是	应用 ID，用于区分不同 TRTC 应用。

RoomId	Int	否	整型房间号 ID，用于在一个 TRTC 应用中唯一标识一个房间。
StrRoomId	String	否	字符串房间号 ID，RoomId 与 StrRoomId 必须配置一项，如果 RoomId 与 StrRoomId 同时配置，使用 RoomId。
UserId	String	是	录制用户 ID，用于在一个 TRTC 应用中唯一标识一个用户。
UserSig	String	是	录制用户签名，用于对一个用户进行登录鉴权认证。
Callback	String	否	录制结束后的回调地址，并使用 POST 方式进行回调。

请求示例：



```
{
    "SdkAppId": 1400000000,
    "RoomId": 43474,
    "UserId": "user_55952145",
    "UserSig": "eJwtzNEKgkAUBNBxxxxxxx",
    "Callback": "https:xxxxxxxx.com/post/xxx"
}
```

识别结果回调接口

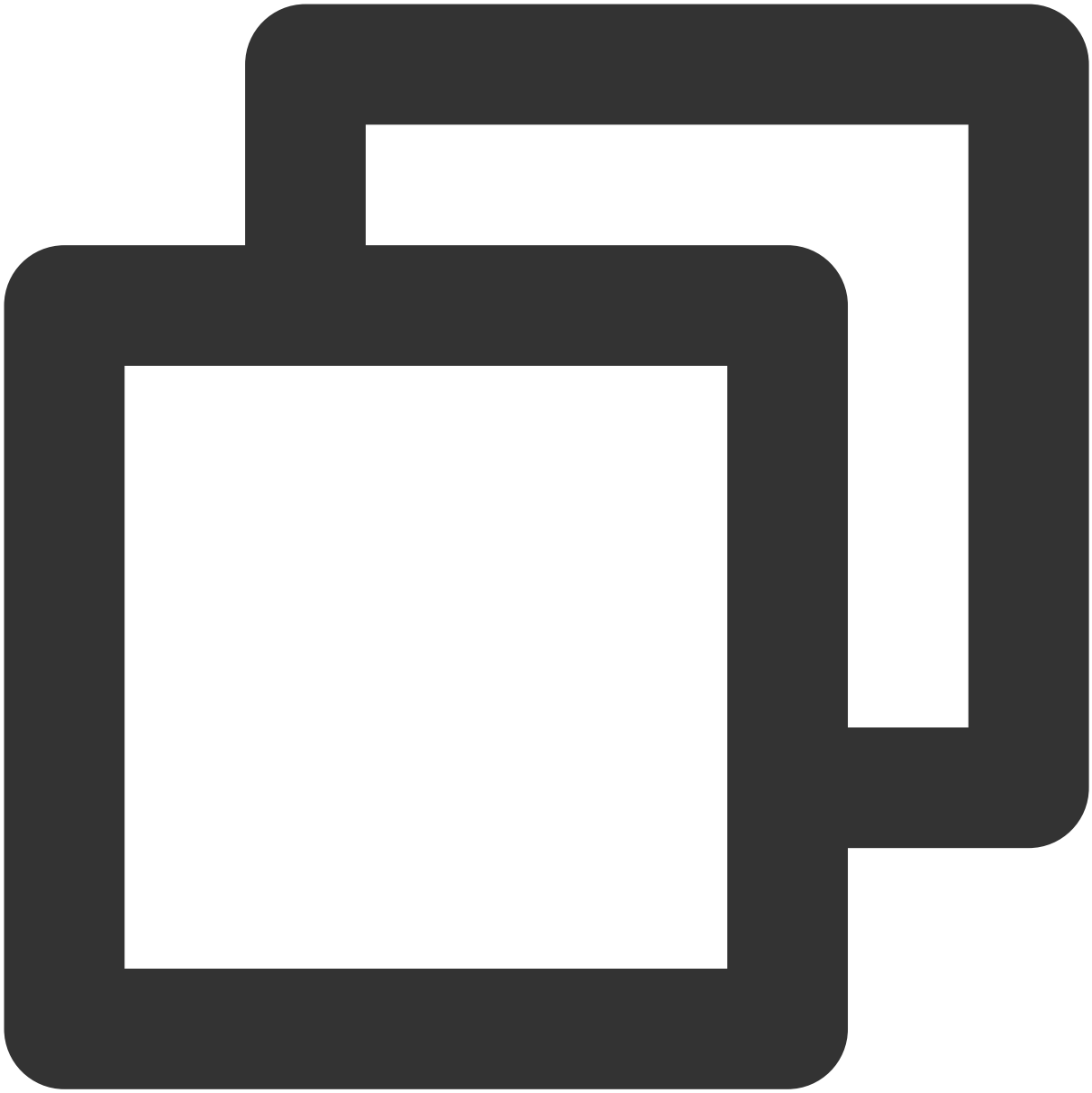
回调参数说明：

参数	类型	必填	说明
SdkAppId	Int	是	应用 ID。
RoomId	int	是	整型房间 ID。
UserId	String	是	识别的用户 ID。
StrRoomId	String	是	字符串房间 ID。
Result	Array	是	语音识别结果 [{},{},{},{}]
Status	String	是	当前用户语言识别状态，normal/finished

Result 为数组类型，元素封装为 JSON 对象，封装格式如下：

参数名称	类型	必选	描述
Voice	String	是	当前一句话文本结果，编码为 UTF8。
Index	Integer	是	当前一句话结果在整个音频流中的序号，从 0 开始逐句递增。
StartTime	Integer	是	当前一句话结果在整个音频流中的起始时间。
EndTime	Integer	是	当前一句话结果在整个音频流中的结束时间。
Message	String	是	识别任务的执行结果。例如，识别结束，识别中，识别失败等。

结果示例：




```
{
    "RequestID": "95941e2c85898384a95b81c2a5******",
    "SdkAppId": 1400000000,
    "RoomId": 43474,
    "UserId": "user_55952145",
    "Status": "recognizing/finished",
    "Result": [{
        "Voice": "实时语音识别",
        "Index": 0,
        "StartTime": 0,
        "EndTime": 1024,
```

```
            "Message": "success"
        }]
    }
```

部署静态网站

最近更新时间：2021-07-13 17:08:06

操作场景

腾讯云 Website 静态网站组件通过使用 Tencent Serverless Framework，基于云上 Serverless 服务（如对象存储等），实现“0”配置，便捷开发，极速部署您的静态网站，Website 静态网站组件支持丰富的配置扩展，如自定义域名和 CDN 加速等。提供了目前最易用、低成本并且弹性伸缩的静态站点开发和托管能力。

特性介绍：

- **按需付费**：按照请求的使用量进行收费，没有请求时无需付费。
- **"0"配置**：只需要关心项目代码，之后部署即可，Serverless Framework 会搞定所有配置。
- **极速部署**：仅需几秒，部署您的静态网站。
- **实时日志**：通过实时日志的输出查看业务状态，便于直接在云端开发应用。
- **便捷协作**：通过云端的状态信息和部署日志，方便的进行多人协作开发。
- **CDN 加速，SSL证书配置和自定义域名** ：支持配置CDN加速，支持自定义域名及 HTTPS 访问。

操作步骤

1. 安装

通过 npm 安装最新版本的 Serverless Framework：

```
$ npm install -g serverless
```

2. 创建

创建并进入一个全新目录：

```
$ mkdir tencent-website && cd tencent-website
```

通过如下命令和模板链接，快速创建一个静态网站托管应用：

```
$ serverless init website-demo
$ cd website-demo
```

下载完毕后，目录结构如下所示：

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

在 `src` 目录中既可以托管简单的 html 文件，也可以托管完整的 React/Vue 的应用。

3. 部署

在 `serverless.yml` 文件下的目录中运行如下命令进行静态网站的部署。部署完毕后，您可以在命令行的输出中查看到您静态网站的 URL 地址，点击地址即可访问网站托管的链接。

```
$ serverless deploy
```

如您的账号未 登录 或 注册 腾讯云，您可以直接通过**微信**扫描命令行中的二维码进行授权登录和注册。

如果希望查看更多部署过程的信息，可以通过 `sls deploy --debug` 命令查看部署过程中的实时日志信息， `sls` 是 `serverless` 命令的缩写。

4. 配置

静态网站组件支持“0”配置部署，也就是可以直接通过配置文件中的默认值进行部署。但您依然可以修改更多可选配置来进一步开发该静态网站项目。

以下是静态网站 Website 组件的 `serverless.yml` 部分配置说明：

```
# serverless.yml
component: website # (必填) 引用 component 的名称，当前用到的是 tencent-website 组件
name: websitedemo # (必填) 该 website 组件创建的实例名称
app: websiteApp # (可选) 该 website 应用名称
stage: dev # (可选) 用于区分环境信息，默认值是 dev
inputs:
src:
src: ./src # 部署项目的目录路径
# dist: ./dist # build 完成后输出目录，如果配置 hook， 此参数必填
# hook: npm run build # hook 脚本
index: index.html
websitePath: ./
region: ap-guangzhou
bucketName: my-bucket
protocol: http
hosts:
- host: abc.com
https:
switch: on
http2: on
```

```
  certInfo:
  certId: 'abc'
```

查看 全量配置及配置说明 >>

当您根据该配置文件更新配置字段后，再次运行 `serverless deploy` 或者 `serverless` 就可以更新配置到云端。

5. 开发调试

部署了静态网站应用后，可以通过开发调试能力对该项目进行二次开发，从而开发一个生产应用。在本地修改和更新代码后，不需要每次都运行 `serverless deploy` 命令来反复部署。您可以直接通过 `serverless dev` 命令对本地代码的改动进行检测和自动上传。

可以通过在 `serverless.yml` 文件所在的目录下运行 `serverless dev` 命令开启开发调试能力。

`serverless dev` 同时支持实时输出云端日志，每次部署完毕后，对项目进行访问，即可在命令行中实时输出调用日志，便于查看业务情况和排障。

6. 查看状态

在 `serverless.yml` 文件所在的目录下，通过如下命令查看部署状态：

```
$ serverless info
```

7. 移除

在 `serverless.yml` 文件所在的目录下，通过以下命令移除部署的静态网站 Website 服务。移除后该组件会对应删除云上部署时所创建的所有相关资源。

```
$ serverless remove
```

和部署类似，支持通过 `sls remove --debug` 命令查看移除过程中的实时日志信息（ `sls` 是 `serverless` 命令的缩写）。

账号配置

当前默认支持 CLI 扫描二维码登录，如您希望配置持久的环境变量/密钥信息，也可以本地创建 `.env` 文件：

```
$ touch .env # 腾讯云的配置信息
```

在 `.env` 文件中配置腾讯云的 SecretId 和 SecretKey 信息并保存：

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

说明：

- 如果没有腾讯云账号，请先 注册新账号。
- 如果已有腾讯云账号，可以在 API 密钥管理中获取 SecretId 和 SecretKey。

接入 Serverless DB
连接 PostgreSQL

最近更新时间：2021-03-30 10:02:51

操作场景

通过 Serverless Framework 组件，您可轻松完成 Serverless DB 的创建部署管理，并通过 SDK 在云函数中轻松完成数据库的连接访问，基于云上 Serverless 服务，实现“0”配置，极速部署，便捷开发，助力业务实现。

Serverless Framework 目前支持 **PostgreSQL** 与 **NoSQL** 两个类型数据库的部署连接，本文介绍如何使用云函数连接 PostgreSQL。

前提条件

- 已安装 Serverless Framework，且不低于以下版本。如未安装，请参考 安装 Serverless Framework 完成安装。

```
Framework Core: 1.67.3
Plugin: 3.6.6
SDK: 2.3.0
Components: 2.30.1
```

- 请确保当前使用账号已配置 **QcloudPostgreSQLFullAccess** 策略。配置方法请参见 授权管理。

操作步骤

本文以 Node.js 开发语言的函数为例，介绍如何通过 Serverless Framework 组件编写创建函数，并访问 PostgreSQL 数据库。配置流程如下：

1. **配置身份信息**：在本地配置腾讯云账户信息。
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。
3. **配置 Serverless DB**：通过Serverless Framework PostgreSQL 组件创建 PostgreSQL 实例，为云函数项目提供数据库服务。
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。
5. **部署与调试**：通过 Serverless Framework 部署项目至云端，并通过云函数控制台进行测试。

6. **移除项目**：可通过 Serverless Framework 移除项目。

配置身份信息

1. 在本地建立目录，用于存放代码及依赖模块。本文以 `test-postgreSQL` 为例。
2. 在 `test-postgreSQL` 下创建 `.env` 文件，并按照以下格式在文件中配置您的腾讯云 SecretId、SecretKey、地域和可用区信息。

```
# .env
TENCENT_SECRET_ID=xxx # 您账号的 SecretId
TENCENT_SECRET_KEY=xxx # 您账号的 SecretKey
# 地域可用区配置
REGION=ap-guangzhou #资源部署区，该项目中指云函数与静态页面部署区
ZONE=ap-guangzhou-2 #资源部署可用区 ， 该项目中指 DB 部署所在的可用区
```

说明：

- 如果没有腾讯云账号，请 注册新账号。
- 如果已有腾讯云账号，请确保您的账号已经授权了 `AdministratorAccess` 权限。 同时，您可在 API 密钥管理 中获取 SecretId 和 SecretKey。

配置私有网络

1. 在 `test-postgreSQL` 下创建文件夹 `VPC` 。
2. 在 `VPC` 中新建 `serverless.yml` 文件，输入以下内容完成私有网络和子网的配置。

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

配置 Serverless DB

1. 在 `test-postgreSQL` 下创建文件夹 `DB` 。
2. 在 `DB` 中新建 `serverless.yml` 文件，输入以下内容完成 PostgreSQL 数据库创建配置。

```
org: fullstack
app: fullstack-serverless-db
stage: dev
component: postgresql
name: fullstackDB
inputs:
region: ${env:REGION}
zone: ${env:ZONE}
dBInstanceName: ${name}
vpcConfig:
vpcId: ${output:${stage}:${app}:serverlessVpc.vpcId}
subnetId: ${output:${stage}:${app}:serverlessVpc.subnetId}
extranetAccess: false
```

编写业务代码

1. 在 `test-postgreSQL` 下创建文件夹 `api`，用于存放业务逻辑代码和相关依赖项。
2. 在文件夹 `api` 下创建文件夹 `src`，并在命令行中进入 `src` 目录，执行以下命令，安装 PostgreSQL 依赖包。

```
npm install pg
```

3. 在 `src` 文件夹下，创建 `index.js` 文件，并输入如下示例代码。在函数中通过 PostgreSQLSDK 创建连接池，并调用数据库。

```
'use strict';
const { Pool } = require('pg');
exports.main_handler = async (event, context, callback) => {
let pgPool = new Pool({
connectionString: process.env.PG_CONNECT_STRING,
});
await pgPool.query(`CREATE TABLE IF NOT EXISTS users (
ID serial NOT NULL,
NAME TEXT NOT NULL,
EMAIL CHAR(50) NOT NULL,
SITE CHAR(50) NOT NULL
);`);
const client = await pgPool.connect();
const { rows } = await client.query({
text: 'select * from users',
});
await client.release();
// 此处要求 postgresql 版本为 8.0 及以上，请检查您的 pg 依赖项版本，如果您使用的版本为 8.
0 以下，请通过 client.end() 来释放链接
```

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

4. 在 `api` 下创建 `serverless.yml` 文件，并输入以下内容，在环境变量中配置 Serverless DB 的 Connection String。

```
org: fullstack
app: fullstack-serverless-db
stage: dev
component: scf
name: fullstack-serverless-db
inputs:
name: ${app}
src:
src: ./src
exclude:
- .env
region: ${env:REGION}
runtime: Nodejs10.15
handler: index.main_handler
timeout: 30
vpcConfig:
vpcId: ${output:${stage}:${app}:serverlessVpc.vpcId}
subnetId: ${output:${stage}:${app}:serverlessVpc.subnetId}
environment:
variables:
PG_CONNECT_STRING: ${output:${stage}:${app}:fullstackDB.private.connectionStrin
g}
```

部署与调试

1. 通过命令行，在 `test-postgreSQL` 目录下，执行以下命令进行部署。

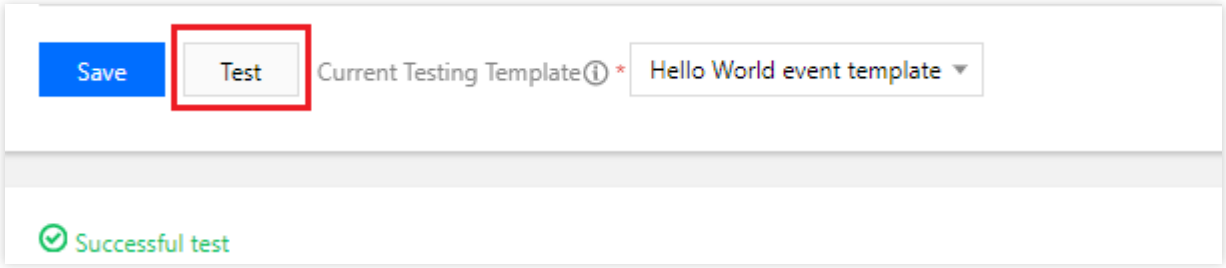
```
sls deploy --all
```

返回结果如下，即为部署成功。

```
serverless ⚡ framework
serverlessVpc:
region: ap-guangzhou
zone: ap-guangzhou-2
vpcId: vpc-0ncak84t
vpcName: serverless
subnetId: subnet-gi085his
subnetName: serverless
fullstackDB:
```

```
region: ap-guangzhou
zone: ap-guangzhou-2
vpcConfig:
subnetId: subnet-gi085his
vpcId: vpc-0ncak84t
dBInstanceName: fullstackDB
dBInstanceId: postgres-0y2x3fd3
private:
connectionString: postgresql://tencentdb_0y2x3fd3:GD0U1(q~g7%3D6ySI@10.0.0.10:5
432/tencentdb_0y2x3fd3
host: 10.0.0.10
port: 5432
user: tencentdb_0y2x3fd3
password: GD0U1(q~g7=6ySI
dbname: tencentdb_0y2x3fd3
fullstack-serverless-db:
FunctionName: fullstack-serverless-db
Description:
Namespace: default
Runtime: Nodejs10.15
Handler: index.main_handler
MemorySize: 128
25s › fullstack-serverless-db › Success
```

2. 部署成功后，您可登录 云函数控制台 查看函数并进行调试。测试步骤请参见 云端测试，测试成功如下图所示：



> 您还可通过 Serverless Dashboard，轻松实现已部署项目的实时监控。

移除项目

在 `test-postgreSQL` 目录下，执行以下命令可移除项目。

```
sls remove --all
```

返回结果如下，即为成功移除。

```
serverless ⚡ framework
38s › tencent-fullstack › Success
```

连接 NoSQL DB

最近更新时间：2020-06-12 16:36:48

操作场景

通过 Serverless Framework 组件，您可轻松完成 Serverless DB 的创建部署管理，并通过 SDK 在云函数中轻松完成数据库的连接访问，基于云上 Serverless 服务，实现“0”配置，极速部署，便捷开发，助力业务实现。

Serverless Framework 目前支持 **PostgreSQL** 与 **NoSQL** 两个类型数据库的部署连接，本文介绍如何使用云函数连接 NoSQL DB。

前提条件

已安装 Serverless Framework，且不低于以下版本。如未安装，请参考 安装 Serverless Framework 完成安装。

```
Framework Core: 1.67.3
Plugin: 3.6.6
SDK: 2.3.0
Components: 2.30.1
```

注意事项

- 请确保您使用的账户下的 `SLS_QcsRole` 的运行角色已具备 `QcloudTCBFullAccess` 策略。如未具备，请前往 访问管理控制台 进行配置。
- 目前云开发 TCB 端仅支持每月最多创建销毁**4**次环境，请谨慎创建，若超过4次部署将会报错。

操作步骤

本文以 Node.js 开发语言的函数为例，介绍如何通过 Serverless Framework 组件编写创建函数，创建并访问 NoSQL DB。配置流程如下：

1. **身份信息配置**：在本地配置腾讯云账户信息。
2. **创建云开发环境配置文件**：通过 Serverless Framework 组件 创建云开发环境，在其中创建并使用 NoSQL DB。
3. **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，创建 NoSQL DB 并进行管理操作。

4. **部署与调试**：通过 Serverless Framework 部署项目至云端，并通过云函数控制台进行测试。
5. **移除项目**：可通过 Serverless Framework 移除项目。

身份信息配置

1. 在本地建立目录，用于存放代码及依赖模块。本文以 `test-NoSQL` 文件夹为例。
2. Serverless Framework 支持以下2种方式配置身份信息，请按需选择：
   - 执行以下命令，并进行身份验证。

     ```
     serverless login
     ```

   - 在 `test-NoSQL` 下创建 `.env` 文件，按照以下内容配置对应的腾讯云 SecretId、SecretKey。

     ```
     # .env
     TENCENT_SECRET_ID=xxx // 您账号的 SecretId
     TENCENT_SECRET_KEY=xxx // 您账号的 SecretKey
     ```

     - 如果没有腾讯云账号，请 注册新账号。
     - 如果已有腾讯云账号，请确保您的账号已经授权了 `AdministratorAccess` 权限。同时，您可在 API 密钥管理 中获取 SecretId 和 SecretKey。

创建云开发环境配置文件

1. 在 `test-NoSQL` 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
component: mongodb
name: mongoDBDemoMongo
org: anycodes
app: mongoDBAPP
stage: dev
inputs:
name: Mydemo
```

编写业务代码

1. 在 `test-NoSQL` 下创建文件夹 `function`，用于存放业务逻辑代码和相关依赖项。
2. 在文件夹 `function` 下创建文件夹 `src`，并在命令行中进入 `src` 目录，执行以下命令安装 tcb 依赖包。

```
npm install --save tcb-admin-node@latest
```

3. 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 Serverless TCB SDK 调用云开发环境，并在其中完成 NoSQL 数据库的调用。

```
const tcb = require('tcb-admin-node')
const app = tcb.init({
secretId: process.env.SecretId,
secretKey: process.env.SecretKey,
env: process.env.MongoId,
serviceUrl: 'https://tcb-admin.tencentcloudapi.com/admin'
})
const db = app.database()
const _ = db.command
exports.main = async (event, context) => {
await db.createCollection('serverless')
const username = 'serverless'
const collection = db.collection('serverless')
if (username) {
await collection.add({username: username})
}
const userList = await collection.get()
return userList
}
```

4. 完成业务代码编写后，创建 `serverless.yml` 文件，并在环境变量中填写您的 **SecretId** 和 **SecretKey**。

> 若使用如下配置，则会创建免费云开发环境。如您已具备免费云开发环境，请将云开发环境 ID 填入 `MongoId` 中，否则会出现报错。

```
component: scf
name: mongoDBDemoSCF
org: anycodes
app: mongoDBAPP
stage: dev
inputs:
name: MongoDBDemo
src: ./src
runtime: Nodejs8.9
region: ap-guangzhou
handler: index.main
environment:
variables:
```

```
SecretId: 请填入您的SecretId
SecretKey: 请填入您的SecretKey
MongoId: ${output:${stage}:${app}:mongoDBDemoMongo.EnvId}
```

部署与调试

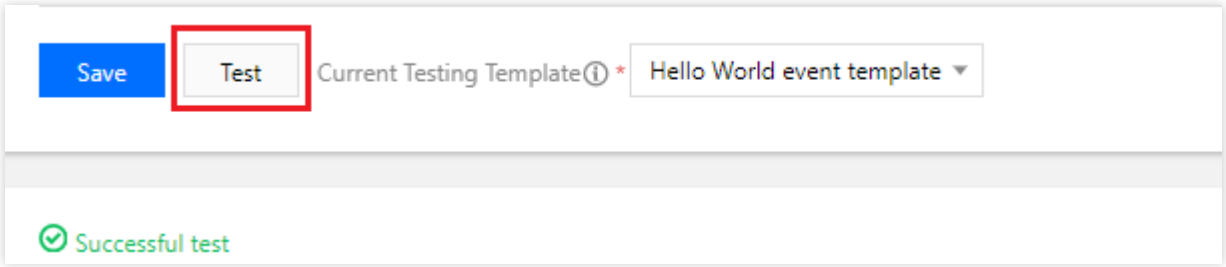
1. 使用命令行在 `test-NoSQL` 下，执行以下命令进行部署。

```
sls deploy --all
```

返回结果如下所示，即为部署成功。

```
serverless ⚡ framework
mongoDBDemoMongo:
Region: ap-guangzhou
Name: Mydemo
EnvID: Mydemo-dyxfxv
FreeQuota: basic
mongoDBDemoSCF:
FunctionName: MongoDBDemo
Description:
Namespace: default
Runtime: Nodejs8.9
Handler: index.main
MemorySize: 128
25s › tcbdemo › Success
```

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。测试步骤请参见 云端测试，测试成功如下图所示：



> 您还可通过 Serverless Dashboard，松实现部署项目的实时监控。

移除项目

在 `test-NoSQL` 目录下，执行以下命令可移除项目。

```
sls remove --all
```

返回如下结果，即为成功移除。

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

部署基于 OCR 的文字识别应用

最近更新时间：2020-08-28 10:05:52

腾讯云文字识别（OCR）基于行业前沿的深度学习技术，将图片上的文字内容智能识别成为可编辑的文本，支持多场景下的印刷体、手写体文字识别，覆盖不同场景下的文字识别需求。
通过 Serverless Framework Component 和 OCR SDK， 您可快速部署一个基于 COS + API + SCF 的通用文字识别应用，主要包含以下组件：

- **Serverless RESTful API：** 通过云函数和 API 网关构建的 Express 框架实现 RESTful API。
- **Serverless 静态网站：** 前端通过托管 React 静态页面到对象存储 COS 中。
- **COS 云端存储:** 用户通过自己创建存储桶来存放目标图像。

前提条件

- 已安装 Node.js（Node.js 版本需不低于8.6版本，建议使用 Node.js10.0 及以上版本）
- 已开通OCR 通用文字识别服务

操作步骤

安装

通过 npm 全局安装 Serverless Framework：

```
$ npm install -g serverless
```

如果之前您已经安装过 Serverless Framework，可以通过下列命令升级到最新版：

```
$ npm update -g serverless
```

安装完毕后，通过运行 `serverless -v` 命令，查看 Serverless Framework 的版本信息，确保版本信息不低于以下版本：

```
$ serverless -v
Framework Core: 1.74.1 (standalone)
Plugin: 3.6.14
SDK: 2.3.1
Components: 2.31.6
```

配置

1.新建一个本地文件夹，使用 `serverless init` 命令，下载相关 template。

```
$ serverless init ocr-app
```

2.在模版中找到 `.env.example` 文件，并改名为 `.env` ，在里面输入您的账户、密钥信息和指定存储桶（此存储桶用于存放上传的图像）。

```
# .env
TENCENT_APP_ID=xxx
TENCENT_SECRET_ID=xxx
TENCENT_SECRET_KEY=xxx

# region of bucket
REGION=ap-guangzhou
```

3.下载所有 npm 依赖。

```
$ npm run bootstrap
```

本地调试

1. 输入以下指令启动服务端：

```
$ cd server && npm run start
```

2. 输入以下指令启动前端：

```
$ cd frontend && npm run start
```

3. 通过 http://localhost:3000 登录前端页面进行本地调试。

部署

1.执行以下命令进行部署：

```
$ sls deploy --all

serverless ⚡ framework

backend:
region: ap-guangzhou
apigw:
serviceId: service-4i62q1pg
```

```
subDomain: service-4i62q1pg-1258834142.gz.apigw.tencentcs.com
environment: release
url: https://service-4i62q1pg-1258834142.gz.apigw.tencentcs.com/release/
scf:
functionName: serverless-ocr
runtime: Nodejs10.15
namespace: default

frontend:
region: ap-guangzhou
website: https://serverless-ocr-1258834142.cos-website.ap-guangzhou.myqcloud.com

38s › serverless-ocr › Success
```

部署成功后，您可以使用浏览器访问项目产生的 website 链接，即可看到生成的网站，单击【上传图片】，项目即可通过 OCR SDK 完成文字识别。

2.执行 `sls remove --all`，可移除项目。

```
$ sls remove --all

serverless ⚡ framework

38s › tencent-fullstack › Success
```

部署流式转码应用

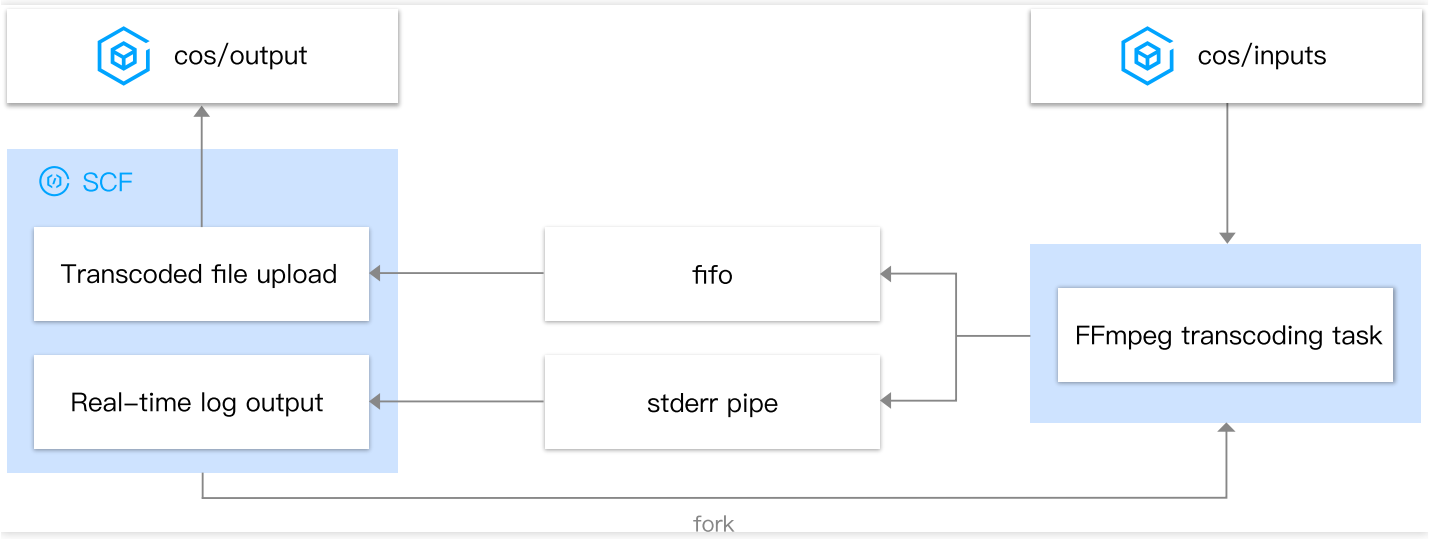
最近更新时间：2021-03-29 15:51:16

应用简介

通过使用 COS + 云函数 + CLS + FFmpeg，您可以快速构建高可用、并行处理、实时日志、高度自定义的视频转码服务。

架构原理

通过云函数创建 FFmpeg 任务进程，云函数进程与 FFmpeg 任务进程通过 pipe 和 FIFO 的方式进行数据传输。云函数进程中的两个任务线程分别接收 FFmpeg 任务进程向函数进程输出的 FFmpeg 日志流与转码后的文件流。实时日志线程将日志流输出，上传任务负责缓存文件流并上传至用户定义的输出 COS。



应用优势

- **流式转码**
  采用流式拉取源视频文件，流式上传转码文件的工作方式，突破了本地存储的限制，且不需要额外部署 CFS 等产品。
- **实时日志**
  视频转码过程中，可通过 CLS 日志实时查看转码进度。同时支持输出 FFmpeg 应用的完整日志。
- **长时运行**
  利用云函数的 长时运行机制，支持 12h-24h 的运行时长，可覆盖大文件耗时较长的转码场景。
- **自定义参数**
  支持用户自定义配置 FFmpeg 命令参数。

应用资源

转码应用部署后，将为您创建以下资源：

- **云函数** ： 流式读取 COS 文件，使用 FFmpeg 转码后流式输出回 COS 中，并将转码过程的实时日志输出到 CLS。
- **CLS 日志** ：存储转码过程的实时日志。CLS 日志可能会产生一定计费，详情参考 CLS 计费规则。

注意事项

- 转码应用需要依赖云函数长时运行能力，详情请参考 异步执行。
- 转码输出桶与函数建议配置在同一区域，因为跨区域配置转码应用稳定性及效率都会降低，并且会产生跨区流量费用。
- 需要为转码应用的云函数创建运行角色，并授权cos读写权限。详细配置参考 运行角色。
- FFmpeg 不同转码场景下指令配置参数不同，因此需要您具有一定的 FFmpeg 使用经验 。 本文中仅提供几个样例作为参考，更多 FFmpeg 指令参考 FFmpeg 官网。

前提条件

1. 安装 Serverlesss Framework。
2. 配置部署账号权限。参考 账号和权限配置
3. 配置 运行角色 权限。

操作步骤

1. 下载转码应用

```
sls init transcode-app
```

进入项目目录 `transcode-app` ，将看到目录结构如下：

```
transcode-app
|- .env #环境配置
|- serverless.yml # 应用配置
|- log/ #log 日志配置
| └── serverless.yml
└──transcode/ #转码函数配置
|- src/
| |- ffmpeg #转码 FFmpeg 工具
| └── index.py
└── serverless.yml
```

- `log/serverless.yml` 定义一个 CLS 日志集和主题，用于转码过程输出的日志保存，目前采用腾讯云 CLS 日志存储。每个转码应用将会根据配置的 CLS 日志集和主题去创建相关资源，CLS 的使用会产生计费，具体参考 CLS 计费规则。
- `transcode/serverless.yml` 定义函数的基础配置及转码参数配置。
- `transcode/src/index.py` 转码功能实现。
- `transcode/src/ffmpeg` 转码工具 FFmpeg。

2. 配置环境变量和应用参数

- 应用参数，文件 `transcode-app/serverless.yml`

```
#应用信息
app: transcodeApp # 您需要配置成您的应用名称
stage: dev # 环境名称，默认为dev
```

- 环境变量，文件 `transcode-app/.env`

```
REGION=ap-shanghai # 应用创建所在区
TENCENT_SECRET_ID=xxxxxxxxxxxx # 您的腾讯云 sercretId
TENCENT_SECRET_KEY=xxxxxxxxxxxx # 您的腾讯云 sercretKey
```

说明：

- 您可以登录腾讯云控制台，可以在 API 密钥管理 中获取 SecretId 和 SecretKey。
- 如果您的账号为主账号，或者子账号具有扫码权限，也可以不配置 SercretId 与 SercretKey，直接扫码部署应用。更多详情参考 账号和权限配置。

3. 配置转码需要的参数信息

- CLS 日志定义，文件 `transcode-app/log/serverless.yml` ：

```
#组件信息全量配置参考 https://github.com/serverless-components/tencent-cls/blob/master/docs/configure.md
component: cls # 引用 component 的名称
name: cls-video # 创建的实例名称，请修改成您的实例名称

#组件参数
inputs:
name: cls-log # 您需要配置一个 name，作为您的 cls 日志集名称
```

```
  topic: video-log # 您需要配置一个 topic，作为您的 cls 日志主题名称
  region: ${env:REGION} # 区域，统一在环境变量中定义
  period: 7 # 日志保存时间，单位天
```

- 云函数及转码配置，文件 `transcode-app/transcode/serverless.yml` ：

```
#组件信息 全量配置参考https://github.com/serverless-components/tencent-scf/blob/master/docs/configure.md
component: scf # 引用 component 的名称
name: transcode-video # 创建的实例名称，请修改成您的实例名称


#组件参数
inputs:
name: transcode-video-${app}-${stage}
src: ./src
handler: index.main_handler
role: transcodeRole # 函数运行角色，已授予cos对应桶全读写权限
runtime: Python3.6
memorySize: 3072 # 内存大小，单位MB
timeout: 43200 # 函数执行超时时间，单位秒，即本demo目前最大支持12h运行时长
region: ${env:REGION} # 函数区域，统一在环境变量中定义
asyncRunEnable: true # 开启长时运行
cls: # 函数日志
logsetId: ${output:${stage}:${app}:cls-video.logsetId} # cls日志集 cls-video为cls组件的实例名称
topicId: ${output:${stage}:${app}:cls-video.topicId} # cls日志主题
environment:
variables: # 转码参数
REGION: ${env:REGION} # 输出桶区域
DST_BUCKET: test-123456789 # 输出桶名称
DST_PATH: video/outputs/ # 输出桶路径
DST_FORMAT: avi # 转码生成格式
FFMPEG_CMD: ffmpeg -i {input} -y -f {dst_format} {output} # 转码基础命令，您可自定义配置，但必须包含ffmpeg配置参数和格式化部分，否则会造成转码任务失败。
FFMPEG_DEBUG: 1 # 是否输出ffmpeg日志 0为不输出 1为输出
TZ: Asia/Shanghai # cls日志输出时间的时区
events:
- cos: # cos触发器
parameters:
bucket: test-123456789.cos.ap-shanghai.myqcloud.com # 输入文件桶
filter:
prefix: video/inputs/ # 桶内路径
events: 'cos:ObjectCreated:*' # 触发事件
enable: true
```

说明：

- 输出桶与函数建议配置在同一区域，跨区域配置应用稳定性及效率都会降低，并且会产生跨区流量费用。
- 内存大小上限为3072MB，运行时长上限为43200s。如需调整，请 提交工单 申请配额调整。
- 转码应用必须开启函数长时运行 asyncRunEnable: true。
- 运行角色请根据 运行角色 创建并授权。
- 示例配置的 FFmpeg 指令仅适用于 AVI 转码场景，详细介绍参考 FFmpeg 指令。

4. 部署项目

在 `transcode-app` 项目目录下，执行 `sls deploy` 部署项目。

```
cd transcode-app && sls deploy
```

5. 上传视频文件

上传视频文件到已经配置好的cos桶指定路径，则会自动转码。本示例中是 COS 桶 `test-123456789.cos.ap-shanghai.myqcloud.com` 下的 `/video/inputs/`

转码成功后，文件将保存在您配置的输出桶路径中。本示例中是 COS 桶 `test-123456789.cos.ap-shanghai.myqcloud.com` 下的 `/video/outputs/`

6. 重新部署

如果需要调整转码配置，修改文件 `transcode/serverless.yml` 后，重新部署云函数即可：

```
cd transcode && sls deploy
```

监控与日志

批量文件上传到 COS 会并行触发转码执行。

1. 登录 云函数控制台 的【函数服务】页面中，单击函数名进入函数管理页面。
2. 单击【日志查询】，即可查看日志监控。
3. 单击【函数管理】>【函数配置】，单击日志主题的链接，跳转至日志服务控制台。
4. 在日志服务控制台的【检索分析】页面中，选择日志集合日志主题，即可查看日志检索分析 。

FFmpeg 工具

FFmpeg 指令

yml 文件 `transcode-app/transcode/serverless.yml` 中 `DST_FORMAT` 与 `FFMPEG_CMD` 指定了转码应用的转码指令，您可根据应用场景自定义配置。

例如：转码 MP4 格式视频，可以将 FFMPEG_CMD 配置如下:

```
DST_FORMAT: mp4
FFMPEG_CMD: ffmpeg -i {input} -vcodec copy -y -f {dst_format} -movflags frag_keyf
rame+empty_moov {output}
```

说明：

- FFMPEG_CMD 必须包含 FFmpeg 配置参数和格式化部分，否则会造成转码任务失败。
- FFmpeg 不同转码场景下指令配置参数不同，因此需要您具有一定的 FFmpeg 使用经验。以上提供的指令仅是针对这几个应用场景的指令。更多 FFmpeg 指令参考 FFmpeg 官网。

自定义 FFmpeg

转码应用场景中提供了默认的 FFmpeg 工具，如果您想自定义 FFmpeg，执行以下操作：

1. 将样例中的 FFmpeg 替换成您自定义的 FFmpeg。
2. 在 `transcode-app/transcode` 目录下再次执行 `sls deploy` 部署更新。

```
cd transcode && sls deploy
```

说明：

自行编译的 FFmpeg 环境与云函数运行环境如果不同，可能会导致 FFmpeg 权限问题。我们提供了云函数执行环境的官方镜像，请使用 官方镜像环境 编译您的 FFmpeg。

运行角色

转码函数运行时需要读取 COS 资源进行转码，并将转码后的资源写回 COS，因此需要给函数配置一个授权 COS 全读写的运行角色。更多参考 函数运行角色。

1. 登录 访问管理控制台，选择新建角色，角色载体为腾讯云产品服务。

2. 在“输入角色载体信息”步骤中勾选【云函数（scf）】，并单击【下一步】：

3. 在“配置角色策略”步骤中，选择函数所需策略并单击【下一步】。

说明：

您可以直接选择 `QcloudCOSFullAccess` 对象存储（COS）全读写访问权限，如果需要更细粒度的权限配置，请根据实际情况配置选择。

4. 输入角色名称，完成创建角色及授权。该角色将作为函数的运行角色，配置在文件 `transcode-app/transcode/serverless.yml` 中。

说明：

由于运行角色密钥最长有效期为12小时，因此函数配置的超时时间不能大于12小时。如果您需要更长的函数执行时长，可以通过改造 `transcode-app/transcode/src/index.py` 中的访问 COS 方式，配置永久密钥去读写访问 COS。但这样会暴露您的密钥在代码中，请谨慎使用。