Serverless 应用中心

进阶指南

产品文档

腾讯云

文档目录

进阶指南
应用管理

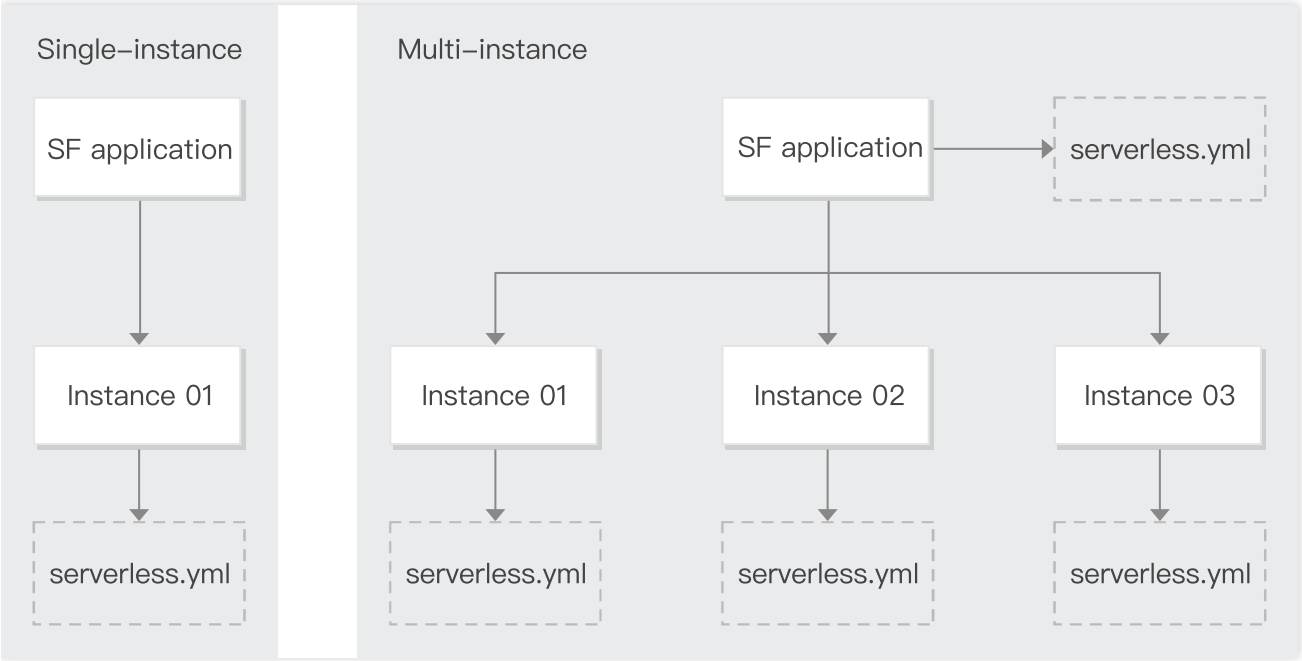
最近更新时间：2021-07-21 17:22:41

概述

每次执行 `sls deploy` ，都是部署一个serverless应用。一个serverless应用是由单个或者多个组件实例构成，每个组件对应一个实例。

每个实例都会涉及一个serverless.yml文件，该文件定义了组件的一些参数，这些参数在部署时用于生成实例的信息。例如 region，定义了资源的所在区。

单实例应用与多实例应用在项目组织上会有一些差别，如下图所示：



单实例应用

项目中只引入一个组件，部署时只生成一个组件实例，这样的应用为单实例应用。

单实例应用一般不需要关心应用名称配置，如果有需要自定义应用名称，可直接在组件的 serverless.yml 中配置。

多实例应用

项目中引入多个组件，部署时生成多个组件实例，这样的应用为多实例应用。

多实例应用需要自定义应用名称，以保证所有组件在统一的应用下进行管理。一般会把应用名称定义在项目目录的 serverless.yml 中，以便所有的组件继承同一个应用名。

Serverless.yml 文件

serverless.yml 文件中定义了应用组织参数及组件 inputs 参数，每次部署时会根据 serverless.yml 文件中的配置信息进行资源的创建、更新和编排。

一份简单的 serverless.yml 文件如下：

```
# serverless.yml
#应用信息
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，非必填，默认为dev
#组件信息
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
#组件配置
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
```

yml 文件中配置信息：

应用信息

参数	说明
org	组织信息，默认腾讯云 APPID。保留字段，不建议使用。
app	应用名称，默认与组件信息中的实例名称一致。对于单实例应用和多实例应用 app 参数的定义方式会有不同，详情请参考 部署应用。
stage	环境信息，默认为 dev。通过定义不同的 stage，为serverless应用开发、测试、发布提供独立的运行环境。详情请参考 环境隔离。

组件信息

参数	说明
component	引用 component 的名称，`sls registry` 查询您可以引入的组件。
name	创建的实例名称，每个组件在部署时将创建一个实例。

参数信息

inputs 下的参数为组件配置参数，不同的组件参数不同。为保证环境隔离，资源唯一，组件资源名称默认会采用 `${name}-${stage}-${app}` 格式。

部署应用

单实例应用

serverless.yml 文件中不配置应用名称 app，部署时会默认生成与实例名称 name 相同的 app 应用名称。

例如创建一个 SCF 项目，项目目录如下：

```
scfDemo
|- index.js
└── serverless.yml
```

其中 serverless.yml 文件配置如下：

```
component: scf
name: myscf
inputs:
src: ./
runtime: CustomRuntime
region: ap-guangzhou
functionName: ${name}-${stage}-${app} #云函数名称
events:
- apigw:
parameters:
endpoints:
- path: /
method: GET
```

在 scfDemo 目录下执行 `sls deploy` 进行部署，默认将生成一个 app 为 myscf 的应用，该应用下包含一个叫 myscf 的 SCF 实例。

对于单实例应用项目，一般使用默认应用名称即可。如果要自定义应用名称，可直接在 serverless.yml 中定义，如：

```
app: scfApp #自定义 app 为 scfApp
component: scf
name: myscf
inputs:
src: ./
runtime: CustomRuntime
region: ap-guangzhou
events:
- apigw:
parameters:
endpoints:
- path: /
method: GET
```

在 scfDemo 目录下执行 `sls deploy` 进行部署，将生成一个app为scfApp的应用，该应用下包含一个叫 myscf 的 SCF 实例。

多实例应用

项目包含多个组件，必须给所有组件统一应用名称。一般我们会在项目根目录下定义一个 serverless.yml 文件进行应用名称配置。

例如部署 Vue + Express + PostgreSQL 全栈网站，项目目录如下：

```
fullstack
|- api
| |- sls.js
| |- ...
| └── serverless.yml
|- db
| └── serverless.yml
|- frontend
| |- ...
| └── serverless.yml
|- vpc
| └── serverless.yml
|- scripts
└── serverless.yml
```

项目目录 fullstack 下的 severless.yml 文件配置了 app：

```
#项目应用信息
app: fullstack
```

每个组件目录下的 severless.yml 文件配置了组件信息和参数信息，如 api 目录下的 serverless.yml：

```
#api配置信息
component: express
name: fullstack-api
inputs:
src:
src: ./
exclude:
- .env
functionName: ${name}-${stage}-${app}
region: ${env:REGION}
runtime: Nodejs10.15
functionConf:
timeout: 30
vpcConfig:
vpcId: ${output:${stage}:${app}:fullstack-vpc.vpcId}
subnetId: ${output:${stage}:${app}:fullstack-vpc.subnetId}
environment:
variables:
PG_CONNECT_STRING: ${output:${stage}:${app}:fullstack-db.private.connectionStrin
g}
apigatewayConf:
enableCORS: true
protocols:
- http
- https
```

说明：
旧版本的模板示例中，会把应用名称 app 写到每个组件里，前提必须保证项目下所有组件的应用名称一致，后续不建议此使用方式。

开发项目

最近更新时间：2020-12-18 11:47:52

前提条件

- 了解 快速部署
- 了解 Serverless 应用
- 了解 账号和权限配置

开发流程

一个项目的开发上线流程大致如下：

1. 初始化项目：将项目进行初始化。例如选择一些开发框架和模板完成基本的搭建工作。
2. 开发阶段：对产品功能进行研发。可能涉及到多个开发者协作，开发者拉取不同的 feature 分支，开发并测试自己负责的功能模块；最后合并到 dev 分支，联调各个功能模块。
3. 测试阶段：测试人员对产品功能进行测试。
4. 发布上线：对于已完成测试的产品功能发布上线。由于新上线的版本可能有不稳定的风险，所以一般会进行灰度发布，通过配置一些规则监控新版本的稳定性，等到版本稳定后，流量全部切换到新版本。

环境隔离

在开发项目的每个阶段，我们都需要一个独立运行的环境来对开发的操作进行隔离。

在 serverless.yml 文件中定义 stage，并把 stage 作为参数写入到组件的资源名称中，部署时以 `实例名-{stage}-应用名` 的方式生成资源。这样我们在不同阶段只要定义不同的 stage 就可以生成不同的资源，达到环境隔离的目的。

以 SCF 组件的 serverless.yml 为例：

```
#应用信息
app: myApp
stage: dev #定义了环境为dev
```

```
#组件信息
component: scf
name: scfdemo

#组件参数
inputs:
name: ${name}-${stage}-${app} #函数名称，以变量${stage}作为资源名称的一部分
src: ./
handler: index.main_handler
runtime: Nodejs10.15
region: ap-guangzhou
events:
- apigw:
parameters:
endpoints:
- path: /
method: GET
```

- 云函数 name 定义为 `${name}-${stage}-${app}` 。
- 开发测试阶段定义 stage 为 `dev` ，部署后云函数为 scfdemo-dev-myApp。
- 上线发布阶段定义 stage 为 `pro` ，部署后云函数为 scfdemo-pro-myApp。
- 不同阶段操作不同的云函数资源，从而达到开发与发布隔离的目的。

说明：

stage 可以直接在 serverless.yml 文件中定义，也可以通过 `sls deploy --stage dev` 直接传参。

权限管理

在开发项目中，需要对不同的人员进行权限分配和管理。例如对于开发人员，只允许其访问某个项目某个环境下的权限，可以参考 账号和权限配置，授予子账号 Serverless Framework 特定资源的操作权限。

以 myApp 项目 dev 环境为例，配置如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"sls:*"
],
"resource": "qcs::sls:ap-guangzhou::appname/myApp/stagename/dev", #app为myApp,sta
```

```
ge为dev
"effect": "allow"
}
]
}
```

灰度发布

灰度发布（又名金丝雀发布）是指在黑与白之间，能够平滑过渡的一种发布方式 。为保证线上业务的稳定性，开发上线项目推荐使用灰度发布。

Serverless应用的灰度发布支持两种方式：**默认别名**与**自定义别名**。更多详情参考 Serverless 灰度发布。

对比项	配置	流量规则设置	适用组件
默认别名	配置简单	只能在最后一次发布的函数版本和 $latest 版本间进行	• 云函数组件 • 涉及云函的相关组件
自定义别名	配置灵活	可以在两个任意函数版本间进行	云函数组件

Serverless 命令

开发项目到上线过程中，需要用到一些Serverless 的相关命令。更多命令请查看 Serverless 支持命令列表。

```
#初始化项目
sls

#下载模板项目scf-demo，模板支持可通过sls registry查询
sls init scf-demo

#下载模板项目scf-demo，并初始化该项目为myapp
sls init scf-demo --name my-app

#部署应用
sls deploy

#部署应用，指定stage为dev
sls deploy --stage dev

#部署应用，并打印部署信息
sls deploy --debug
```

```
#部署并发布函数版本
sls deploy --inputs publish=trues

#部署并切换20%流量到 $latest 版本
sls deploy --inputs traffic=0.2
```

项目实践

参考 开发上线 Serverless 应用。

灰度发布

最近更新时间：2021-01-19 15:06:45

概述

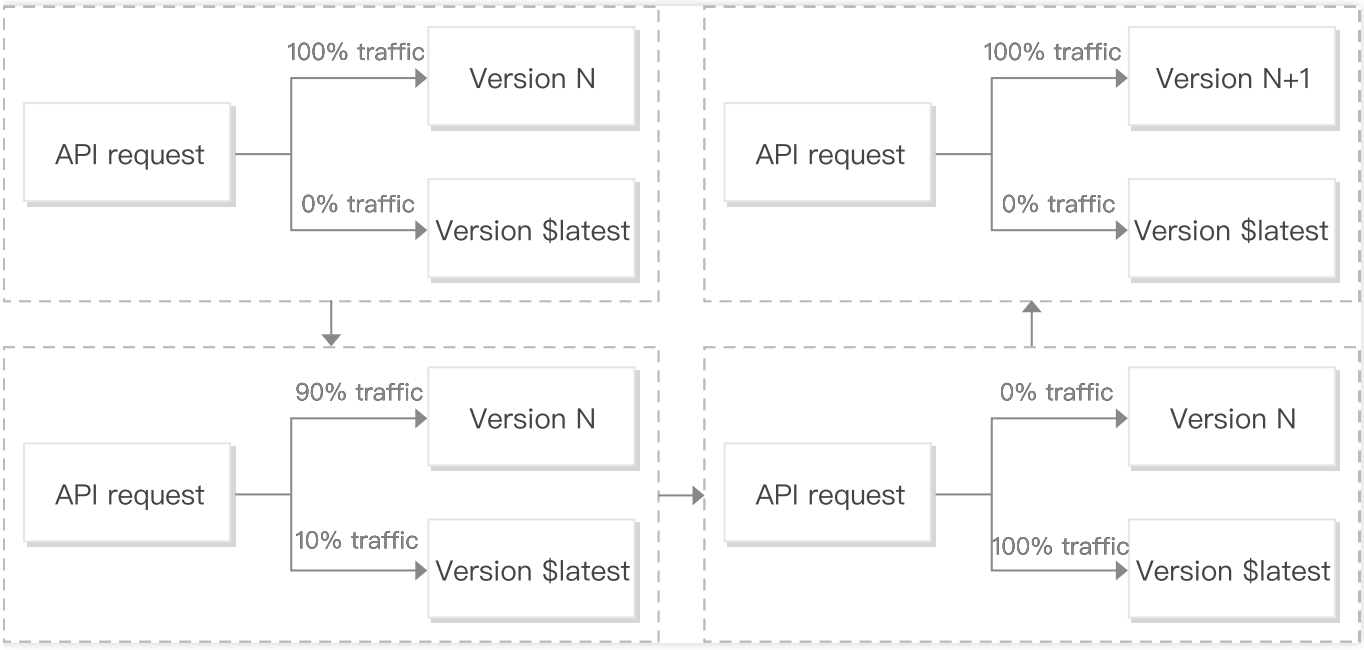
灰度发布（又名金丝雀发布）是指在黑与白之间，能够平滑过渡的一种发布方式。

Serverless 应用的灰度发布是配置云函数别名的流量规则，针对别名中两个不同版本的云函数进行流量规则配置。Serverless Framework 支持的两种方式别名配置：**默认别名**和**自定义别名**。

默认别名

默认别名是配置云函数的 $default（默认流量）别名。该别名中固定有两个云函数版本，一个为 $latest 版本，一个为最后一次函数发布的版本。部署时配置的 traffic 参数为 $latest 版本流量占比，默认另一部分流量切到当前云函数最后一次发布的版本。

每次上线一个新功能，执行 `sls deploy` 会部署到 $latest 版本上。我们将切部分流量在 $latest 版本上进行观察，然后逐步将流量切到 $latest 版本。当流量切到100%时，我们会固化这个版本，并将流量全部切到固化后的版本。



命令说明

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。

函数发布版本

部署时发布项目下所有函数版本：

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

函数流量设置

部署后切换20%流量到 $latest 版本：

```
sls deploy --inputs traffic=0.2
```

- Serverless Framework 流量切换修改的是云函数别名为 $default 的流量规则。
- 每次配置针对的是 $latest，最后一次云函数发布的版本的配置。
- traffic 配置的值为 $latest 版本对应的流量占比，最后一次云函数发布的版本的流量占比为 1-$latest 流量占比。（如 traffic=0.2，实则配置 $default 的流量规则为 {$latest:0.2, 最后一次云函数发布的版本: 0.8}）
- 如果函数还未发任何固定版本，只存在$latest版本的函数情况下，traffic 无论如何设置，都会是 $latest:1.0。

操作步骤

当一个功能测试完毕，需要进行灰度发布，操作如下：

1. 配置生产环境信息到 .env 文件（STAGE=prod 为生产环境）：

```
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100%流量到版本 N+1：

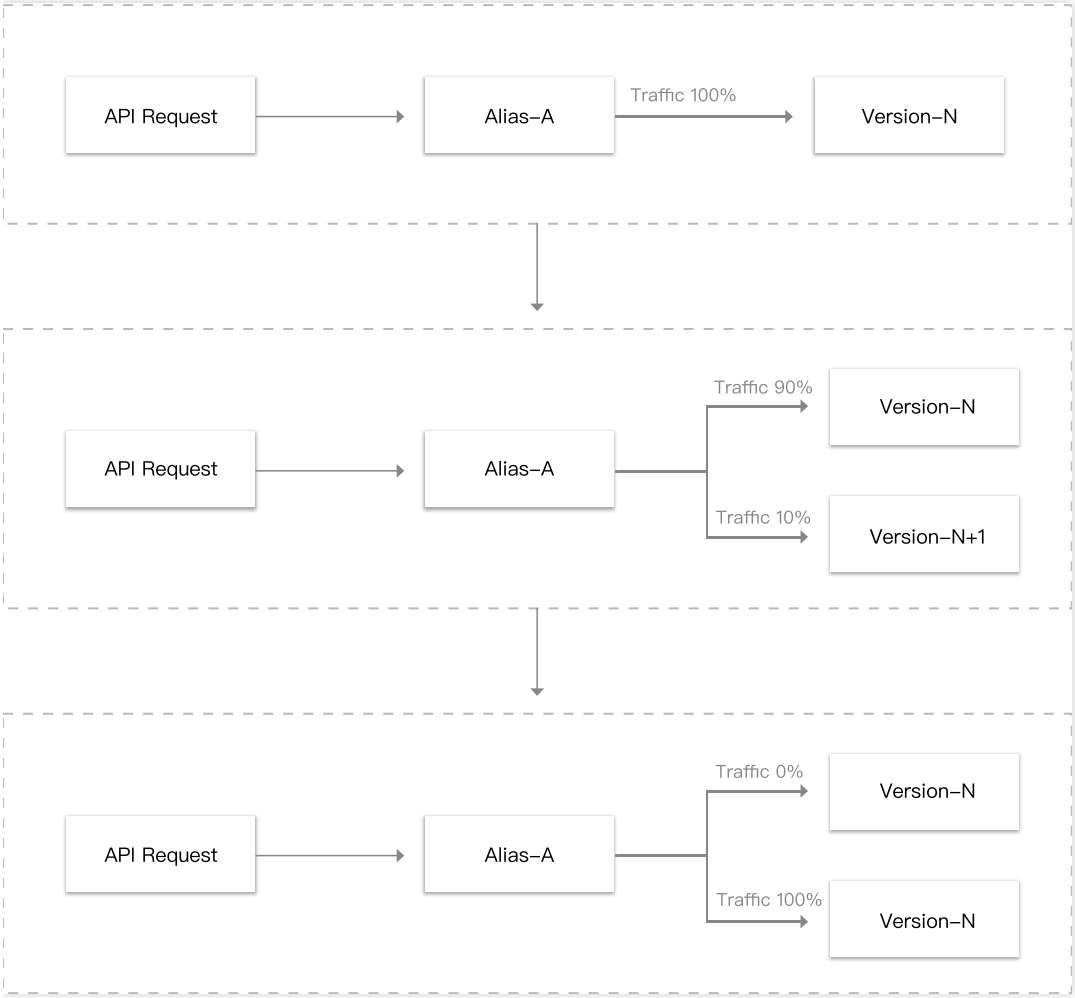
```
sls deploy --inputs publish=true traffic=0
```

自定义别名

自定义别名可以通过命令创建别名，配置指定两个云函数版本配置流量比。

使用自定义别名进行灰度发布时，先将新功能发布到一个新版本上，然后修改别名配置，切部分流量在该版本上进行观察，最后逐步将流量切到该版本。

自定义别名提供了灵活的版本切换，配置相对于默认别名的方式更复杂，适用于对灰度发布能力要求比较高的业务场景。**目前自定义别名只支持云函数组件。**



命令说明

函数发布版本

不部署直接给函数 my-function 发版本：

```
sls publish-ver --inputs function=my-function
```

创建别名

给云函数 my-function 创建别名 routing-alias，路由规则为版本1流量为50%，版本2流量为50%：

```
sls create-alias --inputs name=routing-alias function=my-function version=1
config='{"weights":{"2":0.5}}'
```

更新别名

更新云函数 my-function 别名 routing-alias 的流量规则为版本1流量为10%，版本2流量为90%：

```
sls update-alias --inputs name=routing-alias function=my-function version=1 confi
g='{"weights":{"2":0.9}}'
```

列举别名

列举云函数 my-function 别名 routing-alias：

```
sls list-alias --inputs function=my-function
```

删除别名

删除云函数 my-function 的别名 routing-alias：

```
sls delete-alias --inputs name=routing-alias function=my-function
```

操作步骤

当一个功能测试完毕，需要进行灰度发布，操作如下：

1. 配置生产环境信息到 .env 文件（STAGE=prod 为生产环境）：

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

2. 创建别名 alias-prod，配置 alias-prod 的流量规则（假设目前线上稳定版本为 N）：

```
sls create-alias --inputs function=my-function name=alias-prod version=n config
='{"weights":{"$LATEST":0}}'
```

3. 配置 my-function 函数的 serverless.yml 中触发器对应的别名引用：

```
events: # 触发器
- timer: # 定时触发器
name: #触发器名称，默认timer-${name}-${stage}
parameters:
qualifier: alias-prod #配置别名为alias-prod
cronExpression: '*/5 * * * *' # 每5秒触发一次
enable: true
argument: argument # 额外的参数
```

4. 部署到线上环境 $latest，并发布该新版本（假设函数名为 my-function，发布后的新版本为 N+1）：

```
sls deploy
sls publish-ver --inputs function=my-function
```

5. 配置云函数别名流量规则，切换10%的流量到 N+1版本（假设原线上版本为 N，云函数中的别名为 alias-prod）：

```
sls update-alias --inputs function=my-function name=alais-prod version=n config
='{"weights":{"n+1":0.1}}'
```

6. 持续观察监控，稳定后切换100%到版本 N+1 上：

```
sls update-alias --inputs function=my-function name=alais-prod version=n config
='{"weights":{"n+1":1}}'
```

层部署使用指引

最近更新时间：2022-07-22 14:39:08

操作场景

由于云函数限制，目前只支持上传小于 50MB 的代码包，当您的项目过大时，您可以将依赖放在层中而不是部署包中，可确保部署包保持较小的体积。层的具体使用请参考 层管理相关操作。

操作步骤

创建层

新建层并上传依赖，您可以通过以下两种方式操作：

- 通过 Serverless 应用控制台 直接创建
- 使用 Serverless Framework 的 Layer 组件（参考 Layer 组件）

使用层

您可以通过控制台配置和本地配置两种方法，在项目配置中使用层部署，具体如下：

控制台配置

- 对于 Node.js 框架应用，Serverless Framework 会自动为您创建名为 `${appName}-layer` 的层，并自动帮您把应用的依赖项 node_modules 上传到该层中。
- 导入已有项目时，您也可以选择使用新建层或已有层完成部署，选择新建层时，Serverless Framework 会自动帮您把应用的依赖项 node_modules 上传到该层中。

> 说明：
> 新建层操作仅支持 Node.js 框架，其它框架使用层时，请确保已经完成层的创建并已经把相关依赖项上传到层中。

通过 Layer 组件配置

1. 此处以 Next.js 组件为例，调整本地项目目录，新增 layer 文件夹，并创建 **serverless.yml** 文件，完成层的名称与版本配置，yml 模板如下：

```
app: appDemo
stage: dev
component: layer
name: layerDemo
inputs:
name: test
region: ap-guangzhou
src: ../node_modules #需要上传的目标文件路径
runtimes:
- Nodejs10.14
```

查看详细配置，请参考 layer 组件全量配置文档。

更新后的项目目录结构如下：

```
.
├── node_modules
├── src
| ├── serverless.yml # 函数配置文件
| └── index.js # 入口函数
├── layer
| └── serverless.yml # layer 配置文件
└── .env # 环境变量文件
```

2. 打开项目配置文件，增加 layer 配置项，并引用 layer 组件的输出作为项目配置文件的输入，模板如下：

```
app: appDemo
stage: dev
component: nextjs
name: nextjsDemo
inputs:
src:
src: ./
exclude:
- .env

region: ap-guangzhou
runtime: Nodejs10.15
apigatewayConf:
protocols:
- http
- https
environment: release
```

```
layers:
- name: ${output:${stage}:${app}:layerDemo.name} # layer名称
version: ${output:${stage}:${app}:layerDemo.version} # 版本
```

引用格式请参考 变量引用说明。

3. 在项目根目录下，执行 `sls deploy`，即可完成 Layer 的创建，并将 Layer 组件的输出作为 Next.js 组件的输入完成层的配置。

自定义域名及 HTTPS 访问配置

最近更新时间：2021-06-28 14:42:33

操作场景

通过 Serverless Component 快速构建一个 Serverless Web 网站服务后，如果您希望配置自定义域名及支持 HTTPS 的访问，则可以按照本文提供的两种方案快速配置。

前提条件

- 已经部署了网站服务，获取了 COS/API 网关的网站托管地址。具体部署方法参考 快速部署 Hexo 博客。
- 已拥有自定义域名（例如 www.example.com）。
- 如果需要 HTTPS 访问，可以申请证书并且 获得证书 ID （例如：certificateId : axE1bo3）。

方案一：通过 CDN 加速配置支持自定义域名的 HTTPS 访问

配置前，需要确保账号实名并已经 开通 CDN 服务。

增加配置

在 `serverless.yml` 中，增加 CDN 自定义域名配置：

```
# serverless.yml
component: website
name: myWebsite
app: websiteApp
stage: dev
inputs:
src:
src: ./public
index: index.html
error: index.html
region: ap-guangzhou
bucketName: my-hexo-bucket
protocol: https
# 新增的 CDN 自定义域名配置
hosts:
- host: www.example.com # 希望配置的自定义域名
https:
```

```
switch: on
http2: off
certInfo:
certId: 'abc'
# certificate: 'xxx'
# privateKey: 'xxx'
```

查看完整配置项说明 >>

部署服务

再次通过 `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。

> 说明：
> `sls` 是 `serverless` 命令的简写。

```
$ sls deploy

myWebsite:
url: https://my-hexo-bucket-1250000000.cos-website.ap-guangzhou.myqcloud.com
env:
host:
- https://www.example.com (CNAME: www.example.com.cdn.dnsv1.com)
17s › myWebsite › done
```

添加 CNAME

部署完成后，在命令行的输出中可以查看到一个以 `.cdn.dnsv1.com` 为后缀的 CNAME 域名。参考 CNAME 配置文档，在 DNS 服务商处设置好对应的 CNAME 并生效后，即可访问自定义 HTTPS 域名。

方案二：对 API 网关域名进行自定义域名配置

增加配置

在 `serverless.yml` 中，增加 API 网关自定义域名配置。本文以 egg.js 框架为例，配置如下：

```
# serverless.yml
component: apigateway # (必填) 组件名称，此处为 apigateway
name: restApi # (必填) 实例名称
org: orgDemo # (可选) 用于记录组织信息，默认值为您的腾讯云账户 appid
app: appDemo # (可选) 该应用名称
```

```
stage: dev # (可选) 用于区分环境信息，默认值为 dev
inputs:
region: ap-shanghai
protocols:
- http
- https
serviceName: serverless
environment: release
customDomains:
- domain: www.example.com
# 如要添加https，需先行在腾讯云-SSL证书进行认证获取cettificateId
certificateId: abcdefg
protocols:
- http
- https
endpoints:
- path: /users
method: POST
function:
functionName: myFunction # 网关所连接函数名
```

查看完整配置项说明>>

部署服务

再次通过 `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。

> 说明：
> `sls` 是 `serverless` 命令的简写。

```
$ sls deploy
restApi:
protocols:
- http
- https
subDomain: service-lqhc88sr-1250000000.sh.apigw.tencentcs.com
environment: release
region: ap-shanghai
serviceId: service-lqhc88sr
apis:
-
path: /users
method: POST
```

```
apiId: api-e902tx1q
customDomains:
- www.example.com (CNAME: service-lqhc88sr-1250000000.sh.apigw.tencentcs.com)
8s › restApi › done
```

添加 CNAME 记录

部署完成后，在命令行的输出中可以查看到一个以 `.apigw.tencentcs.com` 为后缀的 CNAME 域名。在 DNS 服务商处设置好对应的 CNAME 并生效后，即可访问自定义 HTTPS 域名。

开发与复用应用模板

最近更新时间：2022-10-21 15:46:49

操作场景

Serverless Cloud 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，本教程将指导您如何使用已有组件，构建您自己的多组件 Serverless 应用模板。

前提条件

已 安装 Serverless Cloud Framework，并保证您的 Serverless Cloud Framework 不低于1.0.2版本：

```
$ scf -v
```

组件全量配置文档

- 基础组件列表
- 框架组件列表

操作步骤

此处以部署一个使用 **Layer + Egg 框架项目**为例，指导您如何在项目中引入多个组件，并快速完成部署，步骤如下：

步骤1：创建项目

新建项目 `app-demo` 并进入该目录下：

```
$ mkdir app-demo && cd app-demo
```

步骤2：构建 Egg 项目

1. 在 `app-demo` 目录下，新建 `src` 文件夹，并在文件夹中新建 Egg 项目：

```
$ mkdir src && cd src
$ npm init egg --type=simple
```

```
$ npm i
```

2. 在 `src` 目录下，编写配置文件 `serverless.yml` ：

```
$ touch serverless.yml
```

egg 组件的 yml 文件示例如下（全量配置文件可参考 Eggjs 组件全量配置）：

```
# serverless.yml
app: app-demo #应用名称，同一个应用下每个组件的 app、stage、org 参数必须保持一一致
stage: dev
component: egg
name: app-demo-egg # （必填）创建的实例名称
inputs:
src:
src: ./ # 需要上传的项目路径
exclude: # 除去 node_modules 以及 .env 文件
- .env
- node_modules
region: ap-guangzhou
functionName: eggDemo # 函数配置
runtime: Nodejs10.15
apigatewayConf:
protocols: # API 网关触发器配置，默认新创建网关
- http
- https
environment: release
```

注意：

- 同一个应用下，每一个组件创建的资源的 **app、stage、org** 参数必须保持一致，**name** 参数必须唯一。
- Egg 组件实际上创建的是一个 API 网关触发器 + 云函数资源，此处可根据您的实际开发场景，选择不同组件，配置方法相似，详情请参考 组件全量配置。

步骤3：创建层

回到 `app-demo` 根目录下，新建 `layer` 文件夹，并在里面新建 layer 配置文件 `serverless.yml` ：

```
$ cd ..
$ mkdir layer && cd layer
```

```
$ touch serverless.yml
```

`serverless.yml` 可以按照如下模板配置（更多配置请参考 Layer 组件全量配置）：

```
# serverless.yml
app: app-demo #应用名称，同一个应用下每个组件的 app、stage、org 参数必须保持一致
stage: dev
component: layer
name: app-demo-layer # （必填）创建的实例名称
inputs:
region: ap-guangzhou
src:
src: ../src/node_modules # 您想要上传到层的项目路径，此处以 node_modules 为例
targetDir: /node_modules # 上传后的文件打包目录
runtimes:
- Nodejs10.15
```

注意：

- 同一个应用下，每一个组件创建的资源的 **app、stage、org** 参数必须保持一致，**name** 参数必须唯一。
- layer 组件也支持从 COS 桶导入项目，详情参考Layer组件全量配置，填写 `bucket` 参数时注意不要带 `-${appid}`，组件会自动为您添加。

步骤4：组织资源关系

同一个应用内，用户可以根据各个资源的依赖关系组织资源的创建顺序，以该项目为例，用户需要先创建好 layer，再在 Egg 项目里使用该 layer，因此需要保证资源的创建顺序为 **layer -> eggjs 应用**，具体操作如下：

修改 Egg 项目的 yml 配置文件，在层配置部分按以下语法进行配置，引用 Layer 组件的部署输出作为 Egg 项目的部署输入，即可保证 Layer 组件一定在 Egg 项目之前完成创建：

```
$ cd ../src
```

在 `serverless.yml` 里，`inputs` 部分增加 layer 配置：

```
inputs:
src:
src: ./
exclude:
- .env
- node_modules
region: ap-guangzhou
```

```
functionName: eggDemo
runtime: Nodejs10.15
layers: # 增加 layer 配置
- name: ${output:${stage}:${app}:app-demo-layer.name} # layer名称
version: ${output:${stage}:${app}:app-demo-layer.version} # 版本
apigatewayConf:
protocols:
- http
- https
environment: release
```

变量引用格式请参考 变量引用说明。

此时已完成 Serverless 应用的构建，项目目录结构如下：

```
./app-demo
├── layer
| └── serverless.yml # layer 配置文件
├── src
| ├── serverless.yml # egg 组件配置文件
| ├── node_modules # 项目依赖文件
| ├── ...
| └── app # 项目路由文件
└── .env # 环境变量文件
```

步骤5：部署应用

在项目根目录下，执行 `scf deploy`，即可完成 Layer 创建，并将 Layer 组件的输出作为 Egg 组件的输入，完成 Egg 框架上云。

```
$ scf deploy
serverless-cloud-framework
app-demo-layer:
region: ap-guangzhou
name: layer_component_xxx
bucket: scf-layer-ap-guangzhou-code
object: layer_component_xxx.zip
description: Layer created by serverless component
runtimes:
- Nodejs10.15
version: 3
vendorMessage: null
app-demo-egg:
region: ap-guangzhou
scf:
functionName: eggDemo
```

```
runtime: Nodejs10.15
namespace: default
lastVersion: $LATEST
traffic: 1
apigw:
serviceId: service-xxxx
subDomain: service-xxx.gz.apigw.tencentcs.com
environment: release
url: https://service-xxx.gz.apigw.tencentcs.com/release/
vendorMessage: null
76s › app-demo › "deploy" ran for 2 apps successfully.
```

点击 `apigw` 输出的 URL，即可访问您已经创建好的应用，执行 `sls info`，可以查看部署的实例状态，执行 `sls remove`，可以快速移除应用。

步骤6：发布应用模板

完成模板构建后，Serverless Cloud Framework 支持您将自己的 Serverless 项目模板发布在 Serverless Registry 应用中心中，提供给团队和他人使用。

1. 创建配置文件

根目录下，新建 `serverless.template.yml` 文件，此时项目目录结构如下：

```
./app-demo
├── layer
| └── serverless.yml # layer 配置文件
├── src
| ├── serverless.yml # egg 组件配置文件
| ├── node_modules # 项目依赖文件
| ├── ...
| └── app # 项目路由文件
├── .env # 环境变量文件
└── serverless.template.yml # 模板项目描述文件
```

2. 配置项目模板文件并发布

```
# serverless.template.yml
name: app-demo # 项目模板的名字，模板唯一标识，不可重复
displayName: 基于 layer 创建的 eggjs 项目模板 #项目模板展示在控制台的名称（中文）
author: Tencent Cloud, Inc. # 作者的名字
org: Tencent Cloud, Inc. # 组织名称，可选
type: template #项目类型，可填 template 或 component，此处为模板
description: Deploy an egg application with layer. # 描述您的项目模板
description-i18n:
zh-cn: 基于 layer 创建的 eggjs 项目模板 # 中文描述
```

```
keywords: tencent, serverless, eggjs, layer # 关键字
repo: # 源代码,通常为您的 github repo
readme: # 详细的说明文件, 通常为您的 github repo README 文件
license: MIT # 版权声明
src: # 描述项目中的哪些文件需要作为模板发布
src: ./ # 指定具体的相对目录, 此目录下的文件将作为模板发布
exclude: #描述在指定的目录内哪些文件应该被排除
# 通常希望排除
# 1. 包含 secrets 的文件
# 2. .git git 源代码管理的相关文件
# 3. node_modules 等第三方依赖文件
- .env
- '**/node_modules'
- '**/package-lock.json'
```

`serverless.template.yml` 文件配置完成后，便可以使用发布命令 `scf publish` 将此项目作为模板发布到应用中心。

```
$ scf publish
serverless ⚡registry
Publishing "app-demo@0.0.0"...
Serverless › Successfully published app-demo
```

3. 复用模板

完成发布后，其他人可通过 `scf init` 指令，快速下载您的模板并进行项目复用。

```
$ scf init app-demo --name example
$ cd example
$ npm install
```

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，引用语法如下：`${org}`、`${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
```