ALIBABA CLOUD

Alibaba Cloud

函数计算

开发工具

文档版本：20210604

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.Funcraft

1.1. 功能概览

Funcraft是一个支持Serverless应用部署的工具，帮助您便捷地管理函数计算、API网关、日志服务等资源。Funcraft通过一个资源配置文件*template.yml*，协助您进行开发、构建、部署等操作。本文详细介绍Funcraft工具的主要功能。
如果您需要使用旧版本的语法，详情请参见旧版本语法。

概览

Funcraft作为一个命令行工具，内置了多个子命令，例如 `fun config` 、 `fun local` 、 `fun deploy` 等。您可以通过以下文档了解Funcraft工具：

- 安装教程：按需在Mac、Linux或Windows上安装Funcraft工具。
- 示例：通过一个简单示例了解如何使用Funcraft工具。
- Funcraft：主要介绍Funcraft工具的规范细节及基本功能。
- Funcraft基本功能如下：
  - 配置Funcraft：介绍Funcraft工具的配置方式。
  - fun init：介绍如何使用 `fun init` 命令指定的模板，快速地创建函数计算应用、体验和开发函数计算的相关业务，您也可以自定义自己的模板。
  - 使用fun local进行本地调试：介绍如何使用 `fun local` 命令将函数计算中的函数在本地完全模拟运行，并提供单步调试的功能，旨在弥补函数计算相对于传统应用开发体验上的短板，并为您提供一种排查函数计算问题的新途径。
  - 使用fun install安装第三方依赖：介绍如何使用 `fun install` 命令安装PIP和APT依赖，提供了命令行接口和Funfile描述文件两种形式。
  - 使用fun build构建函数代码包：介绍如何使用 `fun build` 命令完成从源码到交付产物的构建过程。
  - 使用fun nas管理NAS文件：介绍如何使用 `fun nas` 命令管理*Nas*文件以及上传本地文件*Nas*。
  - 使用fun deploy进行应用部署：介绍如何使用 `fun deploy` 命令部署函数。
- 系列文章：
  - 部署：快速部署云服务地系列文章，详情请参见以下文档：
    - 开发函数计算的正确姿势——部署API网关：介绍如何使用Funcraft工具部署API网关+函数计算应用。
    - 开发函数计算的正确姿势——OSS触发器：介绍如何使用Funcraft工具部署OSS触发器+函数计算应用。
    - 开发函数计算的正确姿势——CDN触发器：介绍如何使用Funcraft工具部署CDN触发器+函数计算应用。

- 实战：使用 `fun init` 、 `fun local` 、 `fun install` 、 `fun deploy` 等命令，快速开发一款Serverless应用的系列文章，详情请参见以下文档：
  - 开发函数计算的正确姿势——爬虫：介绍如何使用Funcraft工具开发一个Serverless应用。
  - 开发函数计算的正确姿势——排查超时问题：介绍如何使用Funcraft工具排查各种问题。
  - 开发函数计算的正确姿势——开发WordPress应用：介绍如何使用Funcraft工具本地开发WordPress Web应用。
  - 开发函数计算的正确姿势——开发NAS文件管理应用：介绍如何使用 `fun local` 命令开发一个NAS文件管理Web应用。
  - 开发函数计算的正确姿势——网页截图服务：介绍如何通过函数计算快速开发网页截图服务。
  - 五分钟教你如何用函数计算部署钉钉群发机器人：介绍如何通过函数计算快速搭建钉钉群发机器人。
  - 五分钟上线——函数计算Word转PDF云服务：介绍如何基于FC-Libreoffice项目快速搭建一个Word转PDF的云服务。
  - 用函数计算搭建微服务——云客服访客名片：介绍如何通过函数计算搭建云客服访客名片的微服务。
  - 三分钟学会如何在函数计算中使用Puppeteer：介绍如何在函数计算中使用Puppeteer，快速构建弹性服务完成各种功能。
  - 函数计算部署机器学习遇到的问题和解法：介绍当机器学习部署到函数计算时，可能会遇到的一些问题及解决方案。
- 自定义模板：使用 `fun init` 快速构建项目的系列文章，详情请参见以下文档：
  - fun init的基本使用方法
  - fun init的模板开发
  - 第三方模板：您可以直接通过 `fun init` 的方式快速初始化一个项目，详情请参见以下文档：
    - Puppeteer模板项目：介绍如何基于Puppeteer在函数计算上部署截图服务。
    - 函数计算R语言模板项目：介绍如何基于函数计算运行R语言的模板项目。
    - 开发函数计算的正确姿势——支持ES6语法和Webpack压缩：介绍如何快速搭建函数计算Node.js项目骨架，支持将ES6代码编译成ES5。
    - 函数计算实现OSS上传较小ZIP压缩文件的自动解压模板项目：介绍如何快速搭建OSS上传较小ZIP压缩文件自动解压的模板项目。
    - 函数计算实现OSS上传较大ZIP压缩文件的自动解压模板项目：介绍如何快速搭建OSS上传较大ZIP压缩文件自动解压的模板项目。
    - 函数计算TensorFlow CharRNN五言绝句古诗模板：介绍如何将TensorFlow CharRNN训练成自动写五言绝句古诗模型，然后部署到函数计算中。
    - 函数计算Selenium Chrome Java模板项目：介绍如何使用压缩比更高的Brotli算法压缩Selenium Chrome Java的模板项目。
    - 函数计算Couchbase模板项目：介绍如何在函数计算平台Node.js 8环境下安装并配置Couchbase。
    - Express项目接入函数计算：介绍如何快速的将Express项目接入函数计算中。
    - Next.js应用接入到函数计算项目：介绍如何三分钟内让Next.js应用在函数计算中运行。
    - ES6代码编译成ES5项目模板：介绍如何通过Webpack将ES6代码编译成ES5，并且打包压缩成一个*js*文件，然后将*js*文件上传到函数计算中运行。

- 依赖安装：使用 `fun install` 命令快速安装函数依赖的系列文章，详情请参见以下文档：
  - fun install的基本使用方法
  - 使用fun install升级系统动态链接库：介绍如何通过 `fun install` 命令升级系统动态链接库。
  - Install的原理，详情请参见以下文档：
    - 函数计算安装依赖库方法小结：介绍不同语言依赖安装的方法。
    - 函数计算Python连接SQL Server小结：介绍如何通过FC-Docker安装pymssql库，与 `fun install` 的底层实现思路是类似的。
    - 手把手教您将libreoffice移植到函数计算平台：介绍如何使用FC-Docker将Libreoffice移植到函数计算中。
- 本地运行与调试： `fun local` 在本地运行、调试函数，以及排查Bug技巧的系列文章，详情请参见以下文档：
  - fun local的基本使用方法
  - 开发函数计算的正确姿势——HTTP Trigger本地运行调试：介绍如何使用 `fun local` 在本地运行、单步调试配置HTTP Trigger函数。
  - 开发函数计算的正确姿势——本地运行、调试、发布NAS函数：介绍如何使用 `fun local` 在本地运行、单步调试配置NAS服务的函数。
  - 开发函数计算的正确姿势——API本地运行调试：介绍如何使用 `fun local` 通过API在本地运行、单步调试函数。
- 语法校验：Funcraft提供了比较强大的语法校验功能，让您通过精准的报错信息快速地将其修正。
  开发函数计算的正确姿势——fun validate语法校验排错指南：您可以通过报错信息修正*template.yml*文件中的错误语法。

- Funcraft的常见问题与解答
- 更多示例

反馈

如果在使用中遇到问题，您可以在这里反馈。

参考

- 使用函数计算做为API后端服务
- 函数计算
- API Gateway
- Funcraft发布2.0新版本
- 函数计算工具链新成员 ——fun local发布
- 三十分钟快速搭建Serverless网盘服务
- FC Docker

开源许可

The MIT License

1.2. 安装命令行工具

Funcraft是函数计算提供的一种命令行工具，通过该工具，您可以便捷地管理函数计算、API网关、日志服务等资源。通过一个资源配置文件*template.yml*，Funcraft工具即可协助您进行开发、构建、部署操作。本文介绍安装Funcraft工具的三种方式。

步骤一：安装Funcraft工具

您可通过以下三种方式安装Funcraft工具：

- 通过npm包管理安装：适用于已经预装了npm的Windows、MacOS、Linux的操作系统。
  i. 在Windows、MacOS、Linux操作系统中执行以下命令安装Funcraft工具。

  ```
  npm install @alicloud/fun -g
  ```

  > 说明
  > - 如果在Linux或MacOS操作系统下执行该命令报错且报错信息为 `Error: EACCES: permission denied` ，请执行命令 `sudo npm install @alicloud/fun -g` 。
  > - 如果安装过程较慢，可以考虑使用淘宝npm源，安装命令为 `npm --registry=https://registry.npm.taobao.org install @alicloud/fun -g` 。

  ii. 安装完成之后，执行以下命令查看版本信息。

  ```
  fun --version
  ```

- 通过下载二进制安装：适用于Windows、MacOS、Linux操作系统。流程如下：
  i. 打开Releases页面，在最新的版本中选择一个对应系统的Release压缩包链接，单击该链接即可直接下载。
  ii. 下载到本地解压后，即可直接使用。

  针对不同操作系统的具体步骤如下：
  - Windows操作系统
    a. 在Releases页面，找到最新的发布版本并下载*fun-*-win.exe.zip*文件， `*` 表示版本号，例如v3.6.20。
    b. 解压文件*fun-v3.6.20-win.exe.zip*得到*fun-v3.6.20.win.exe*文件，重命名为*fun.exe*。
    c. 将*fun.exe*文件增添到系统变量Path目录即可，例如*C:\WINDOWS\System32*。
    d. 打开命令终端，执行 `fun.exe --version` ，查看返回版本号以验证是否安装成功。

- Linux操作系统
  a. 打开Release页面，在Release页面查看Funcraft工具的最新版本，例如Release 3.6.20，表示最新版本为v3.6.20。
  b. 打开终端，例如Bash或Zsh，执行以下命令下载Funcraft工具。

     > 注意 如果Funcraft工具的最新版本有变化，请把执行命令中 http://funcruft-release.oss-accelerate.aliyuncs.com/fun/fun-v3.6.20-linux.zip 的 v3.6.20 更替为最新版本。

     ```
     curl -o fun-linux.zip http://funcruft-release.oss-accelerate.aliyuncs.com/fun/fun-v3.6.20-linux.zip
     ```

     预期输出：

     ```
     % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                    Dload  Upload   Total   Spent    Left  Speed
     100 32.2M  100 32.2M    0     0  2606k      0  0:00:12  0:00:12 --:--:-- 2376k
     ```

  c. 执行以下命令，解压ZIP文件。

     ```
     unzip fun-linux.zip
     ```

     预期输出：

     ```
     Archive:  fun-v3.6.20-linux.zip
      inflating: fun-v3.6.20-linux
     ```

  d. 执行以下命令，将Funcraft工具的路径增添到环境变量PATH目录中。

     > 注意 如果Funcraft工具的最新版本有变化，请把执行命令中的 v3.6.20 更替为最新版本。

     ```
     mv fun-v3.6.20-linux /usr/local/bin/fun
     ```

  e. 执行以下命令，验证Funcraft工具是否安装成功。

     ```
     fun --version
     ```

     预期输出：

     ```
     3.6.20
     ```

- MacOS操作系统
  a. 打开Release页面，在Release页面查看Funcraft工具的最新版本，例如Release 3.6.20，表示最新版本为v3.6.20。
  b. 打开终端，例如Bash或Zsh，执行以下命令下载Funcraft工具。

> 注意 如果Funcraft工具的最新版本有变化，请把执行命令中 http://funcruft-release.oss-accelerate.aliyuncs.com/fun/fun-v3.6.20-macos.zip 的 v3.6.20 更替为最新版本。

```
curl -o fun-macos.zip http://funcruft-release.oss-accelerate.aliyuncs.com/fun/fun-v3.6.20-macos.zip
```

预期输出：

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 47.8M  100 47.8M    0     0   724k      0  0:01:07  0:01:07 --:--:-- 3727k
```

  c. 执行以下命令，解压ZIP文件。

```
unzip fun-macos.zip
```

预期输出：

```
Archive:  fun-macos.zip
 inflating: fun-v3.6.20-macos
```

  d. 执行以下命令，将Funcraft工具的路径增添到环境变量PATH目录中。

> 注意 如果Funcraft工具的最新版本有变化，请把执行命令中的 v3.6.20 更替为最新版本。

```
mv fun-v3.6.20-macos /usr/local/bin/fun
```

  e. 执行以下命令，验证Funcraft工具是否安装成功。

```
fun --version
```

预期输出：

```
3.6.20
```

- 通过Homebrew包管理器安装：适用于MacOS操作系统，且更符合MacOS开发者习惯。
  执行以下命令安装Funcraft工具。

```
brew tap vangie/formula
brew install fun
```

（可选）步骤二：安装Docker

如果您需要通过Funcraft工具进行依赖编译和安装、本地运行调试，因为这些操作涉及到**fun install**、**fun build**、**fun local**等命令的功能，所以您需在开发环境中安装Docker。

- Windows操作系统
  请参见官方教程。如果遇到网络问题，可以下载阿里云提供的Docker For Windows。
- MacOS操作系统
  请参见官方教程。如果遇到网络问题，可以下载阿里云提供的Docker For MacOS。
- Linux操作系统

请参见官方教程。如果遇到网络问题，请参见阿里云Docker CE镜像源站下载。

如果您需要在其他平台或体系结构上安装Docker，请参见Dockerhub。

后续步骤

配置Funcraft

1.3. 配置Funcraft

在使用fun命令前，需要先配置Funcraft。本文介绍配置Funcraft的具体步骤。

背景信息

Funcraft有三种配置方式，通过这三种方式配置Funcraft的生效优先级按以下顺序依次递减：

- 通过.env文件配置
- 通过fun config命令配置
- 通过环境变量配置

使用场景示例

例如，您使用**fun config**命令配置了region为cn-beijing，但是有一个特殊项目需要部署到cn-shanghai，那么您可以单独在这个项目创建一个*.env*文件，并在该文件中写入内容 `REGION=cn-shanghai` ，然后直接执行**fun deploy**命令即可将项目部署至cn-shanghai地域，以此满足您在不同项目部署不同地域的需求。

通过.env文件配置

1. 在项目目录即*template.yml*文件所在目录下，创建一个名为*.env*的文件。

   > **说明** 建议您将*.env*放到*.gitignore*中，避免泄漏重要的账户信息。

2. 在*.env*文件录入以下配置。

```
ACCOUNT_ID=xxxxxxxx
REGION=cn-shanghai
ACCESS_KEY_ID=xxxxxxxxxxx
ACCESS_KEY_SECRET=xxxxxxxxxx
FC_ENDPOINT=https://{accountid}.{region}.fc.aliyuncs.com
TIMEOUT=10
RETRIES=3
```

通过fun config命令配置

1. 执行以下命令。

```
fun config
```

2. 按照提示依次配置Account ID、AccessKey ID、AccessKey Secret、Default Region Name。其中Account ID、AccessKey ID可以在函数计算控制台**概览**页的**常用信息**区域获得，如下图所示。

完成配置操作后，Funcraft会将配置保存到您目录下的*.fcli/config.yaml*文件中。

通过环境变量配置

针对不同的平台，通过环境变量配置Funcraft的步骤不同，通过环境变量进行配置时，可选配置项与*.env*相同。

1.4. 使用fun init初始化应用

`fun init` 作为Funcraft的子命令用于指定应用模板，您可以使用函数计算提供的常用模板，也可以进行自定义应用模板，快速创建函数计算应用。本文介绍如何通过 `fun init` 快速体验和开发函数计算的应用。

功能介绍

您可以执行以下代码获取 `fun init` 的功能介绍。

```
fun init --help
```

返回结果如下：

```
Usage: fun init [options] [template]
Initialize a new project based on a template. A template can be a folder containing template metadata and boile
rplate files, a name of a pre-built template, or a url that resolves to a template. You can find more information a
bout template at https://yq.aliyun.com/articles/674364.
Options:
 -o, --output-dir [path]  Where to output the initialized app into (default: ".")
 -n, --name [name]      The name of your project to be generated as a folder (default: "")
 -m, --merge [merge]    Merge into the template.[yml|yaml] file if it already exist (default: false)
 --no-input           Disable prompting and accept default values defined template config
 -V, --var [vars]       Template variable
 -h, --help           display help for command
 Examples:
  $ fun init
  $ fun init event-nodejs8
  $ fun init foo/bar
  $ fun init gh:foo/bar
  $ fun init gl:foo/bar
  $ fun init bb:foo/bar
  $ fun init github:foo/bar
  $ fun init gitlab:foo/bar
  $ fun init bitbucket:foo/bar
  $ fun init git+ssh://git@github.com/foo/bar.git
  $ fun init hg+ssh://hg@bitbucket.org/bar/foo
  $ fun init git@github.com:foo/bar.git
  $ fun init https://github.com/foo/bar.git
  $ fun init /path/foo/bar
  $ fun init -n fun-app -V foo=bar /path/foo/bar
```

参数	默认值	是否必填	描述	示例
-o, --output-dir	.	否	初始化的应用程序的输出目录。	`fun init -o /path/foo/bar`

参数	默认值	是否必填	描述	示例
-n, --name	fun-app	否	应用程序所在文件夹的名称。	fun init -n <projectname>
-m, --merge	false	否	合并当前内容到YAML文件中。	fun init -m
--no-input	无	否	禁用提示并接受默认值定义模板配置。 如果您不想被提示输入模板变量，您可以通过该选项跳过输入提示，直接使用模板变量的默认值。	fun init --no-input
-V, --var	无	否	模板变量。模板中可能会有很多模板变量，在初始化模板的过程中，如果需要您填写某些模板的变量值，会提示您进行设置，您可以通过以下方式进行设置： • 自定义设置该变量。 • 回车使用默认值。 • 通过 -V, --var 选项设置模板的变量值，通过该选项设置好模板的变量值后，系统就不会再提示您输入了。 **说明** 选项格式：以等号分隔的键值对，键只能包含字母、数字和下划线（_），值可以是任意字符。	fun init -V foo=bar -V bar=baz
-h, --help	无	否	打印使用说明。	fun init -h

获取函数计算官方离线模板

获取应用模板

您可以通过以下途径获取应用模板：

- 函数计算官方模板，包括：
  - 离线模板：内嵌在Funcraft工具中的模板。
  - 在线模板：通过模板名称来指定模板位置，在Funcraft内部会将其转换为GitHub地址。
- Git或Mercurial仓库模板：从相应仓库中获取应用模板。
- 本地文件系统路径模板：从本地获取应用模板。

- 自定义模板：从GitHub上获取已上传的自定义模板。

  1. 执行以下命令指定应用所在目录。

     ```
     fun init -o /myapp
     ```

  2. 选择Funcraft内置的应用模板。

     ```
     ? Select a template to init (Use arrow keys or type to search)
     > event-nodejs12
       event-nodejs10
       event-nodejs8
       event-nodejs6
       event-python3
       event-python2.7
       event-java8
       event-php7.2
       event-dotnetcore2.1
       http-trigger-nodejs12
       http-trigger-nodejs10
       http-trigger-nodejs8
       http-trigger-nodejs6
       http-trigger-python3
       http-trigger-python2.7
       http-trigger-java8
     (Move up and down to reveal more choices)
     ```

     > **说明** 在指定目录下您既可以看到刚创建好的模板应用，还可以在代码文件中编写您的逻辑代码。

函数计算官方在线模板存放在GitHub中。

执行 `fun init <templates name>` 即可获取在线应用模板。

指定Git或Mercurial仓库模板，支持多种前缀的缩写形式。对于GitHub，可以进一步缩写为 `user/repo` 。您可以使用以下模板：

- `fun init foo/bar`
- `fun init fun init gh:foo/bar/bar`
- `fun initfun init gl:foo/barfoo/bar`
- `fun init fun init bb:foo/bar/bar`
- `fun fun init github:foo/barfoo/bar`
- `fun fun init gitlab:foo/barfoo/bar`
- `fun fun init bitbucket:foo/barfoo/bar`
- `fun ifun init git+ssh://git@github.com/foo/bar.gitnit foo/bar`
- `fun init hg+ssh://hg@bitbucket.org/bar/foo`
- `fun init git@github.com:foo/bar.git`
- `fun init https://github.com/foo/bar.git`
- `fun init /path/foo/bar`
- `fun init -n fun-app -V foo=bar /path/foo/bar`

  1. 克隆在线模板至本地。
  2. 执行以下代码获取本地应用模板。

```
fun init /path/foo/bar //替换为您的应用所在文件夹的路径。
```

如果现有的模板不能满足您的需求，您可以自定义模板，然后将模板上传到GitHub上。当您或其他人需要使用该模板时，可以从GitHub中获取该模板。

1. 自定义您的应用模板。
2. 将模板上传至GitHub。
3. 执行以下命令获取自定义模板。

```
fun init <template URL>
```

获取函数计算官方在线模板

获取Git或Mercurial仓库模板

获取本地模板

获取自定义模板

1.5. 使用fun local进行本地调试

`fun local` 作为Funcraft的一个子命令，您可以直接通过 `fun local` 命令进行本地调试。Funcraft可以将函数计算中的函数在本地完全模拟运行，并提供单步调试的功能，弥补了函数计算相对于传统应用开发体验上的短板，并为您提供一种排查问题的新途径。

背景信息

`fun local` 提供 `fun local invoke` 和 `fun local start` 两个子命令：

- `fun local invoke` 支持本地运行事件函数。
- `fun local start` 支持本地运行HTTP触发器函数以及事件函数：
  - 当使用Fun Local Start HTTP触发器函数时，允许通过浏览器直接触发函数运行。
  - 当使用Fun Local Start事件函数时，允许通过InvokeFunction API或者SDK进行调用，详情请参见InvokeFunction API和SDK列表。

`fun local` 现已集成到VSCode、IDEA、Pycharm等IDE的图形化插件中，相较于命令行，这些插件往往通过图形化的方式带来更好的使用体验。

- 如果您使用VSCode，建议您直接使用VSCode插件，详情请参见Aliyun Serverless VSCode Extension插件。
- 如果您使用IDEA或Pycharm，建议您直接使用Cloud Toolkit插件，详情请参见管理函数。

fun local invoke命令格式

您可以执行以下命令查看相关信息。

```
fun local invoke -h
```

返回结果如下。

```
Usage: invoke [options] <[service/]function>
Run your serverless application locally for quick development & testing.
Options:
  -d, --debug-port <port>  used for local debugging
  -c, --config <ide>       print out ide debug configuration. Options are VSCode
  -e, --event <path>       event file containing event data passed to the function
  -h, --help               output usage information
```

- 本地运行函数
  运行函数的命令格式如下。

  ```
  fun local invoke [options] <[service/]function>
  ```

  其中service可以省略。从调用方式上，可以理解为 `fun local invoke` 支持通过函数名调用，或者支持服务名/函数名方式调用，示例如下：

  ```
  fun local invoke function
  ```

  ```
  fun local invoke service/function
  ```

  > **说明**
  > - 如果您需要精准匹配，可以使用服务名/函数名的方式。
  > - 如果*template.yml*中包含多个服务，而多个服务中包含相同名称的函数时，通过函数名的方式调用，Funcraft只会运行第一个名称匹配的函数。

  以本地运行Java类型的函数为例。

  > **说明** Java不同于解释型的语言，在运行函数前需要编译函数。

  假设待运行的事件函数名称为Java8。具体操作步骤如下：

  i. 执行以下命令编译函数。

  ```
  fun build java8
  ```

  返回结果如下：

  ```
  using template: template.yml
  start building function dependencies without docker
  building localdemo/java8
  running task flow MavenTaskFlow
  running task: MavenCompileTask
  running task: MavenCopyDependencies
  running task: CopyMavenArtifacts
  Build Success
  Built artifacts: .fun/build/artifacts
  Built template: .fun/build/artifacts/template.yml
  Tips for next step======================
  * Invoke Event Function: fun local invoke
  * Invoke Http Function: fun local start
  * Deploy Resources: fun deploy
  ```

  ii. 执行以下命令运行函数：

```
fun local invoke
```

- 本地调试
  `fun local invoke` 支持 `-d, —debug-port` 选项，可以对函数进行本地单步调试。 `fun local` 涉及到的调试技术全部是基于各个语言通用的调试协议实现的，使用对应语言的远程调试方法都可以进行调试。
  - 本地调试Nodejs、Python类型的函数
    对于Nodejs 6、Nodejs 8、Python 2.7、Python 3、Java 8类型的函数，调试方法基本一致。下面以Nodejs 8为例，调试端口为3000。
    a. 执行以下命令调试函数并输出VSCode的配置信息。

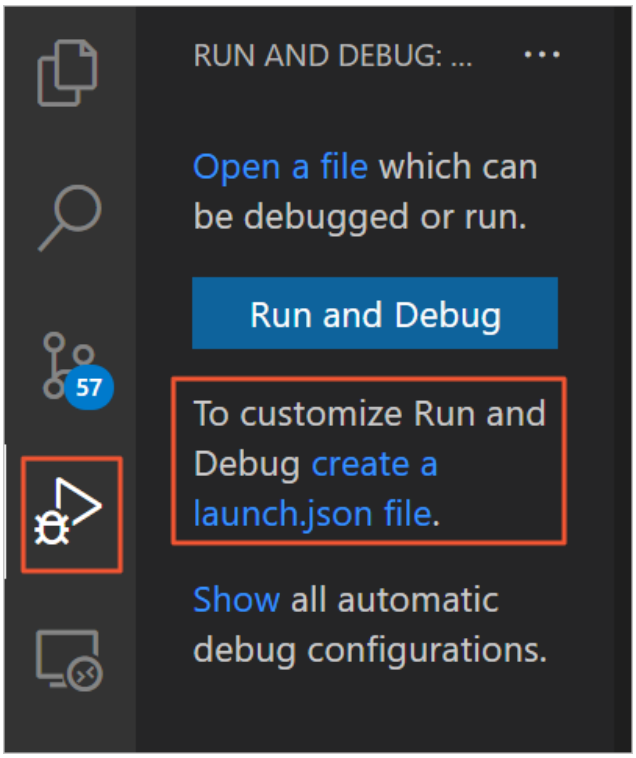
    ```
    fun local invoke -d 3000 --config VSCode nodejs8
    ```

    返回结果如下：

    ```
    skip pulling images ...
    you can paste these config to .vscode/launch.json, and then attach to your running function
    ////////////////// config begin //////////////////
    {
      "version": "0.2.0",
      "configurations": [
        {
          "name": "fc/localdemo/nodejs8",
          "type": "node",
          "request": "attach",
          "address": "localhost",
          "port": 3000,
          "localRoot": "/Users/tan/code/fun/examples/local/nodejs8",
          "remoteRoot": "/code",
          "protocol": "inspect",
          "stopOnEntry": false
        }
      ]
    }
    ////////////////// config end //////////////////
    Debugger listening on ws://0.0.0.0:3000/b65c288b-bd6a-4791-849b-b03e0d16b0ce
    For help see https://nodejs.org/en/docs/inspector
    ```

    b. 使用上一步骤中返回的信息配置VSCode。
    首次使用VSCode本地调试函数，需要配置VSCode，如果已经配置则无需再配置。

a. 选择 > **create a launch.json file**，创建*launch.json*文件。

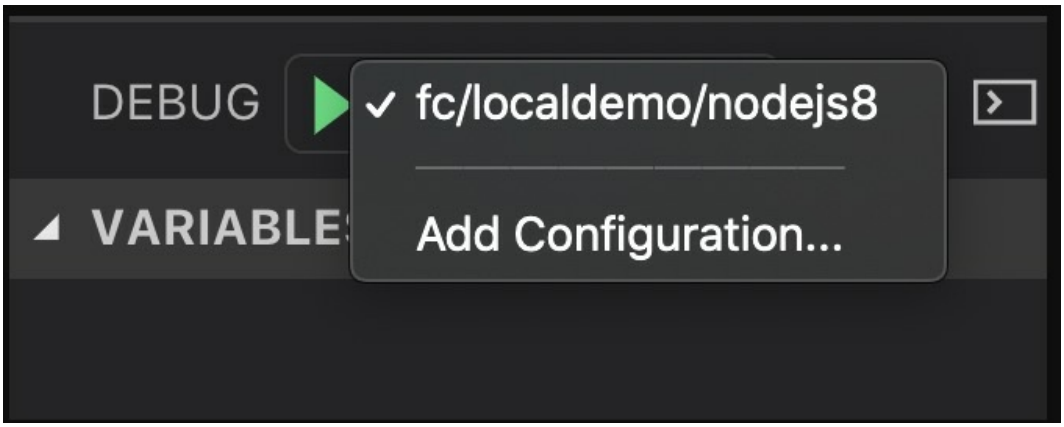


b. 复制步骤一日志中 config begin 与 config end 之间的配置到*launch.json*中。

launch.json

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "fc/localdemo/nodejs8",
            "type": "node",
            "request": "attach",
            "address": "localhost",
            "port": 3000,
            "localRoot": "/Users/tan/code/fun/examples/local/nodejs8",
            "remoteRoot": "/code",
            "protocol": "inspector",
            "stopOnEntry": false
        }
    ]
}
```

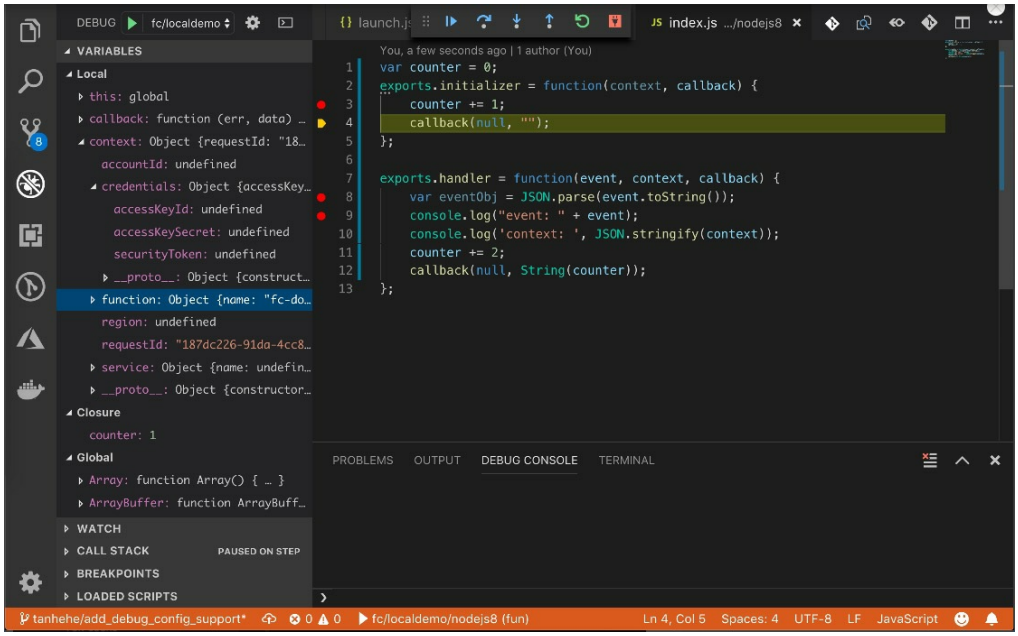
完成上面配置后，在Debug视图可以看到配置的函数列表。



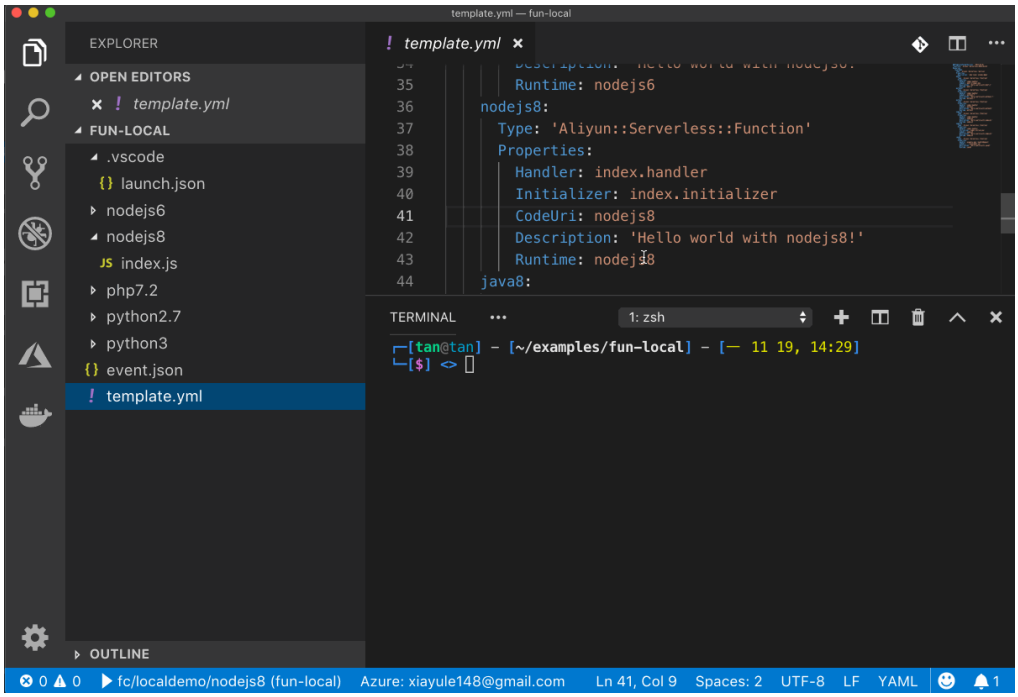
c. 使用VSCode调试。

a. 在VSCode编辑器侧边栏单击**设置断点**。

b. 然后单击**开始调试**按钮，即可开始调试。



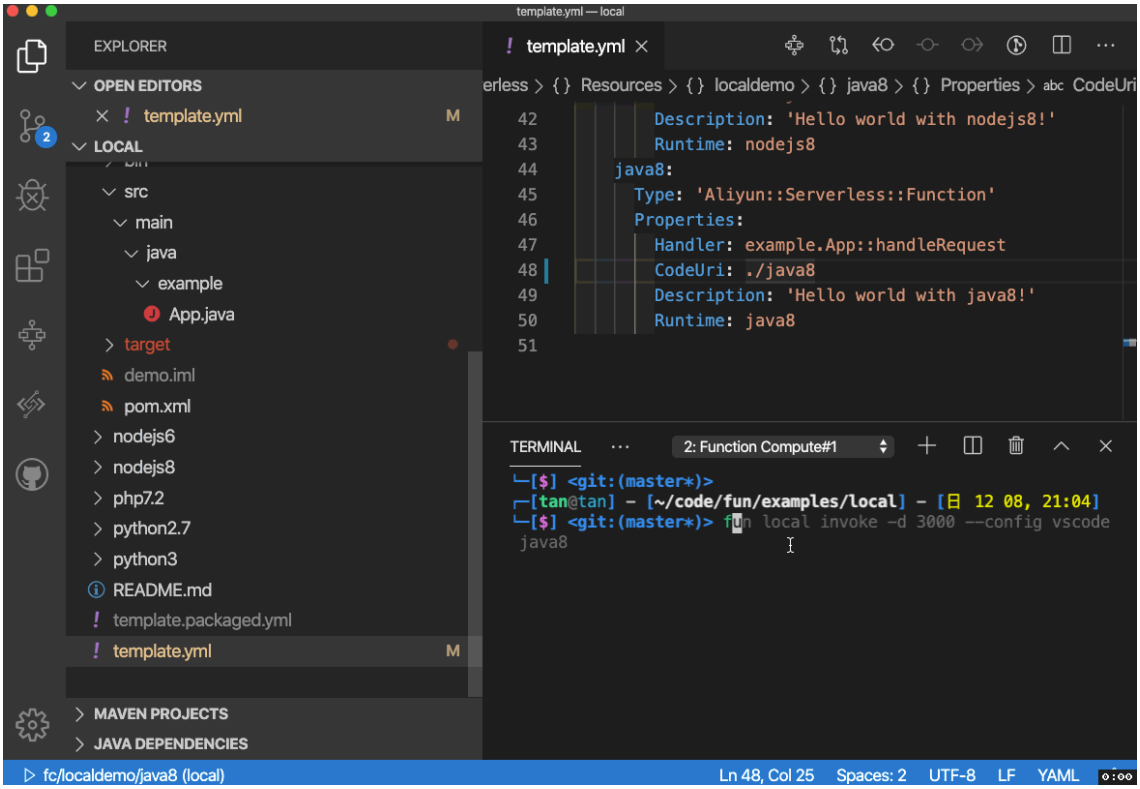
以下是一个Node.js 8函数本地单步调试的流程例子：



- 本地调试Java类型的函数

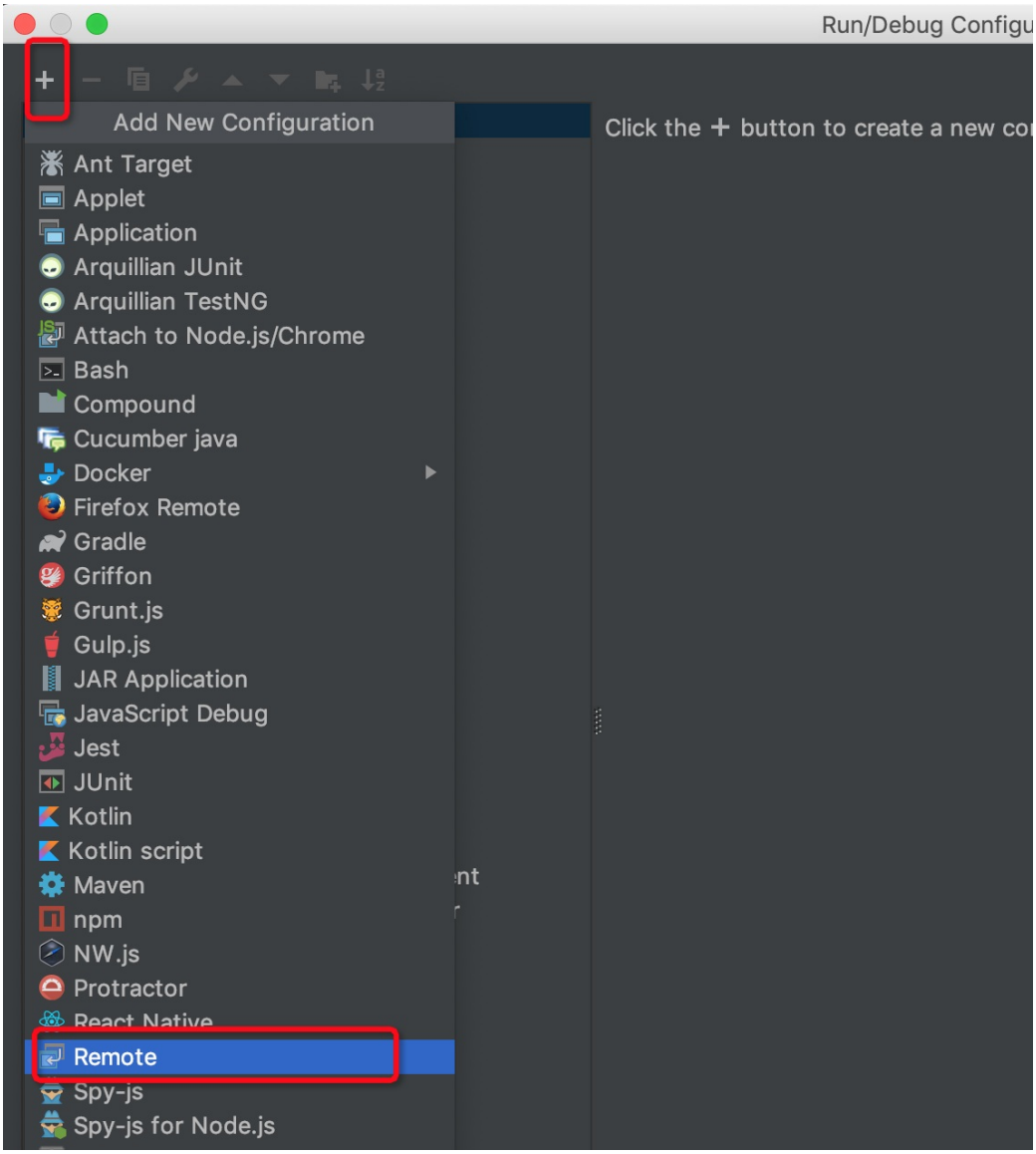
- 使用VSCode调试Java
  使用VSCode调试Java时，您可以通过以下方式安装Language Support for Java(TM) by Red Hat和Debugger for Java两个插件：
  - 在VSCode软件中搜索Language Support for Java(TM) by Red Hat和Debugger for Java进行安装。
  - 使用VSCode的插件市场安装插件比较简单，请参见操作介绍。

  您可以通过以下视频了解如何使用VSCode调试Java类型的函数。

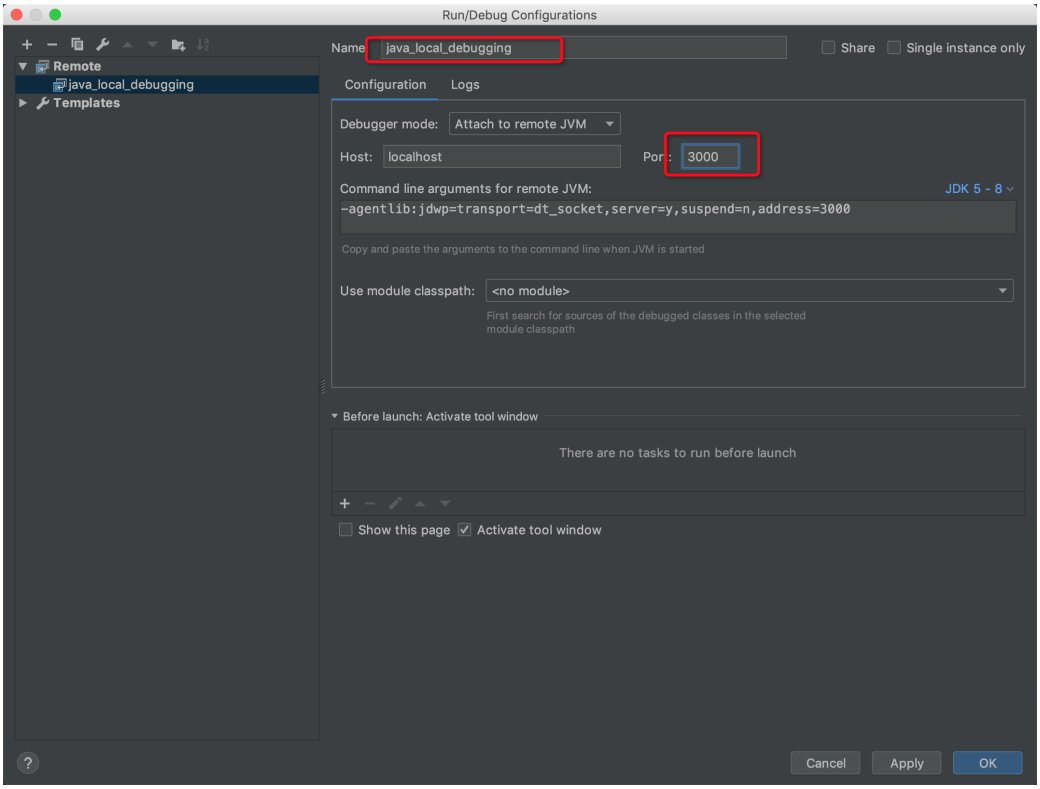


- 使用IDEA调试Java
  a. 配置Remote Debugging
     a. 在菜单栏选择**Run > Edit Configurations...**。

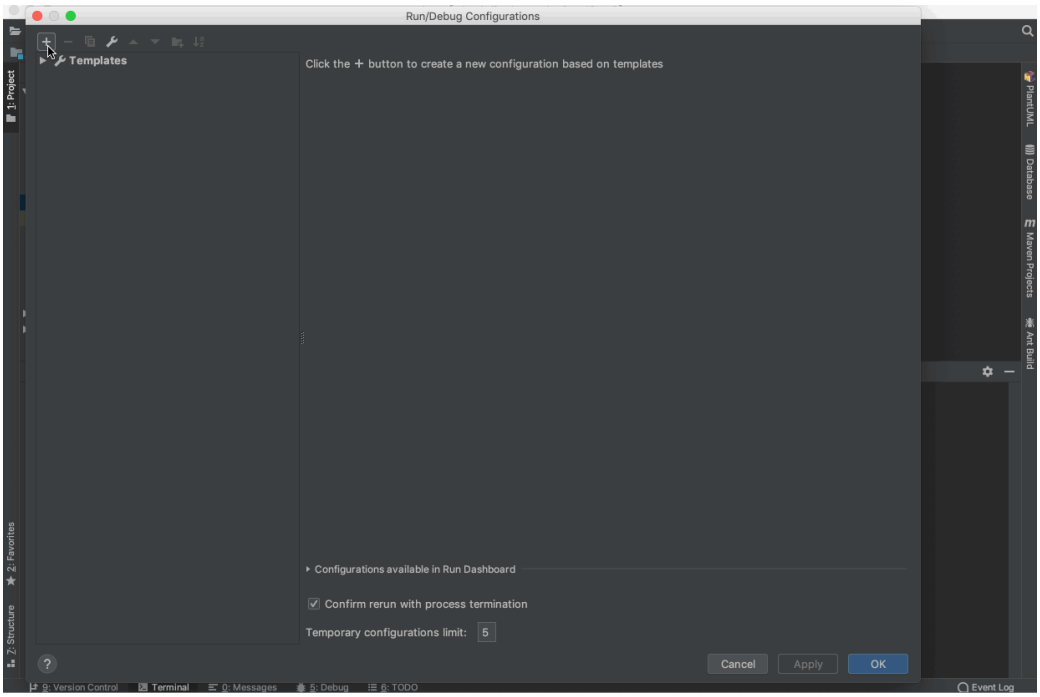
b. 单击左上角的加号图标，新建一个Remote Debugging。



c. 自定义设置**Name**，并将端口号**Por**设置为3000。



您可以通过以下视频了解如何配置IDEA Remote Debugging。



b. 使用IDEA调试本地事件函数

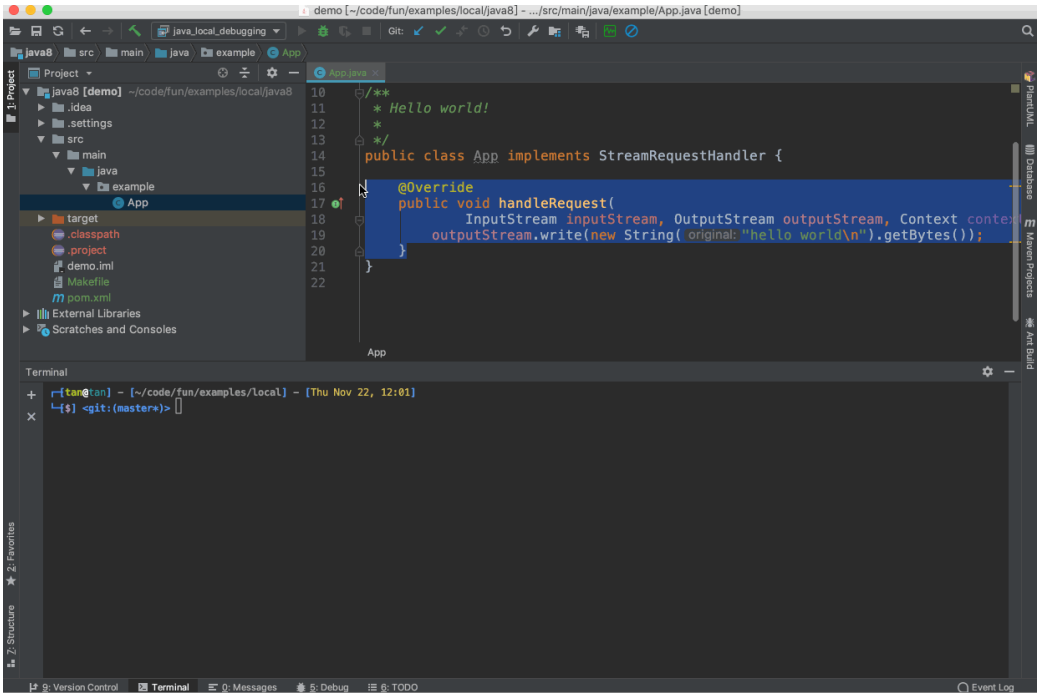
a. 您可以使用Debug的方式运行Java函数。

```
fun local invoke -d 3000 java8
```

b. 当函数停止时，您就可以使用IDEA连接并通过以下方式开始调试。

- 在菜单栏选择**Run > Debug...**。
- 在工具栏中单击**Debug**，即可开始调试。

您可以通过以下视频了解如何使用IDEA运行Remote Debugging。



- 本地调试PHP类型的函数

a. 执行以下命令启动函数。

```
fun local invoke -d 3000 --config VSCode php72
```

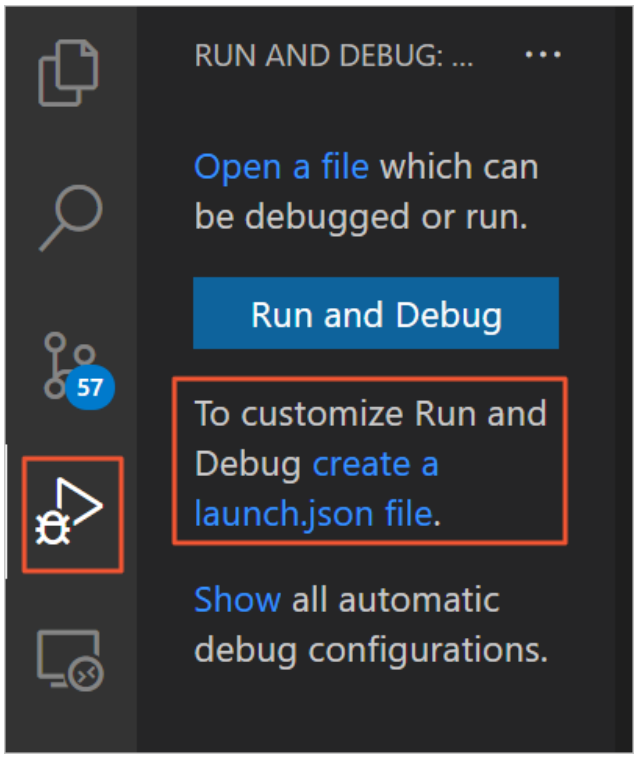
执行结果如下：

```
skip pulling images ...
you can paste these config to .vscode/launch.json, and then attach to your running function
////////////////// config begin //////////////////
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "fc/localdemo/php72",
      "type": "php",
      "request": "launch",
      "port": 3000,
      "stopOnEntry": false,
      "pathMappings": {
        "/code": "/Users/tan/code/fun/examples/local/php7.2"
      },
      "ignore": [
        "/var/fc/runtime/**"
      ]
    }
  ]
}
////////////////// config end //////////////////
FunctionCompute php7.2 runtime inited.
FC Invoke Start RequestId: 6e8f7ed7-653d-4a6a-94cc-1ef0d028e4b4
FC Invoke End RequestId: 6e8f7ed7-653d-4a6a-94cc-1ef0d028e4b4
hello world
RequestId: 6e8f7ed7-653d-4a6a-94cc-1ef0d028e4b4      Billed Duration: 48 ms      Memory Size: 1998
MB      Max Memory Used: 58 MB
```

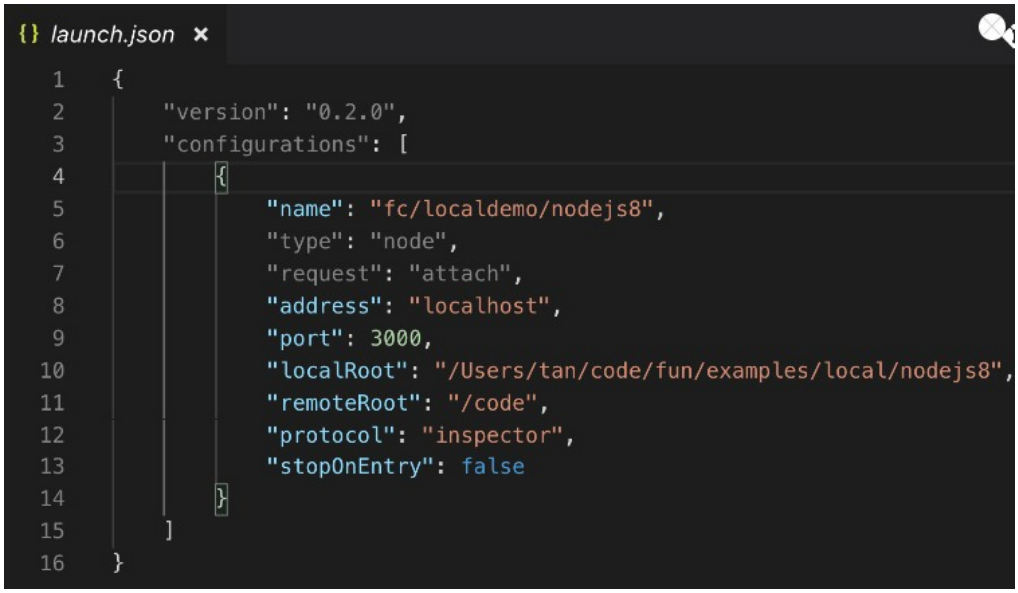
b. 您可以按照下面的方式进行调试PHP的调试器：

a. 使用上一步骤中返回的信息配置VSCode。
首次使用VSCode本地调试函数，需要配置VSCode，如果已经配置则无需再配置。

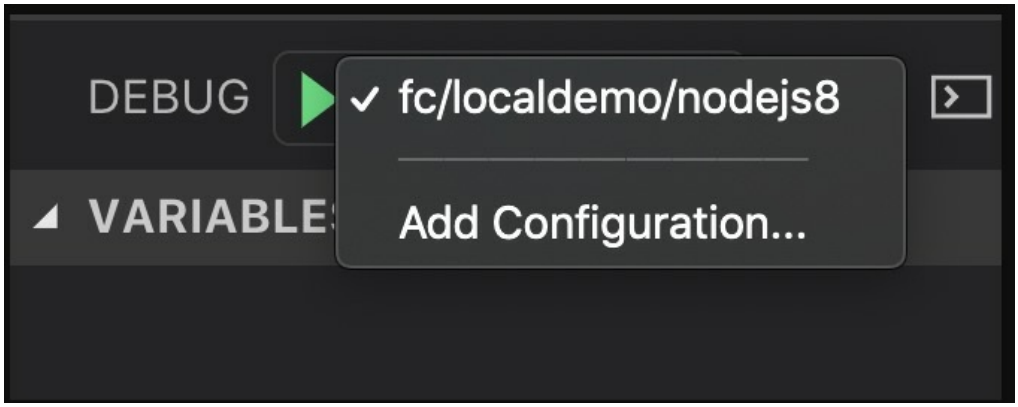
a. 选择 > create a launch.json file，创建*launch.json*文件。



b. 复制步骤一日志中 `config begin` 与 `config end` 之间的配置到*launch.json*中。



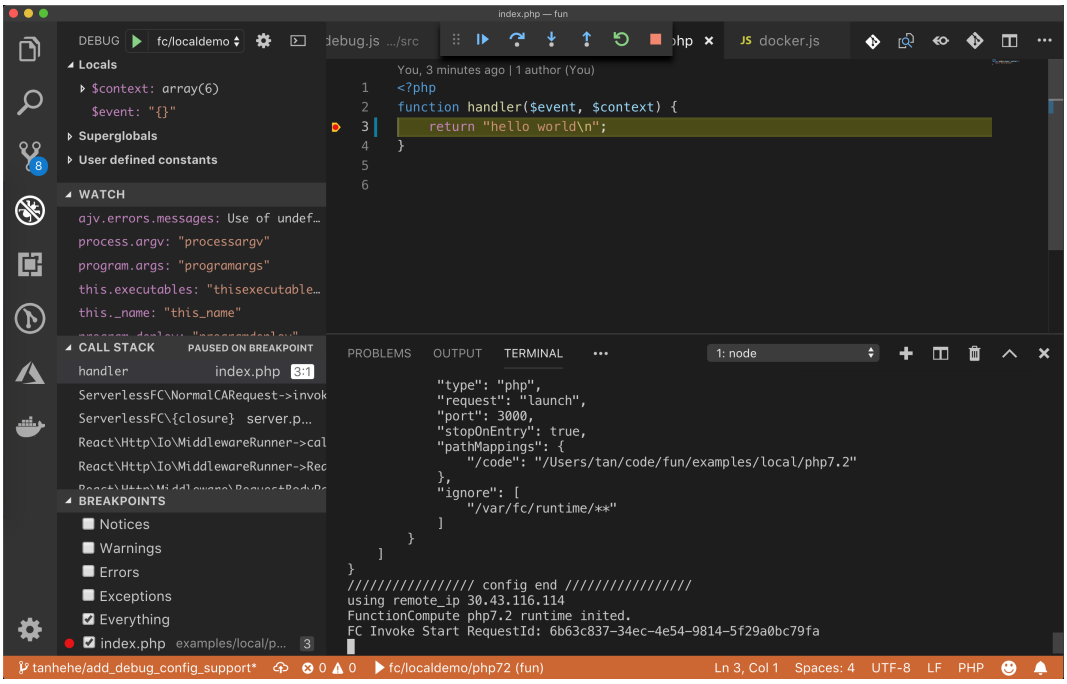
完成上面配置后，在Debug视图可以看到配置的函数列表。



b. 使用VSCode调试。在VSCode编辑器侧边栏，单击**设置断点**然后单击**开始调试**按钮，即开始调试。

c. 执行以下代码再次调试运行函数。

```
fun local invoke -d 3000 php72
```



- Event事件源
  函数计算提供了丰富的触发器，包括对象存储触发器、日志服务触发器、CDN事件触发器等。在本地无论是运行还是调试函数，为了能够完全模拟线上环境，您需要构造触发事件。
  以下以JSON为例，假设触发事件内容为：

  ```
  {
    "testKey": "testValue"
  }
  ```

  您可以通过以下三种途径将事件内容传给函数执行：
  - 管道：`echo '{"testKey": "testValue"}' | fun local invoke nodejs8`。
  - 文件：将JSON内容写入到文件，自定义文件名，例如`event.json`。然后通过`-e`命令指定文件名。

    ```
    fun local invoke -e event.json nodejs8
    ```

  - 重定向：`fun local invoke nodejs8 < event.json` 或 `fun local invoke nodejs8 <<< '{"testKey": "testValue"}'` 等，详情请参见文章。

`fun local start` 命令格式

您可以执行以下代码查看相关的帮助信息。

```
fun local start
```

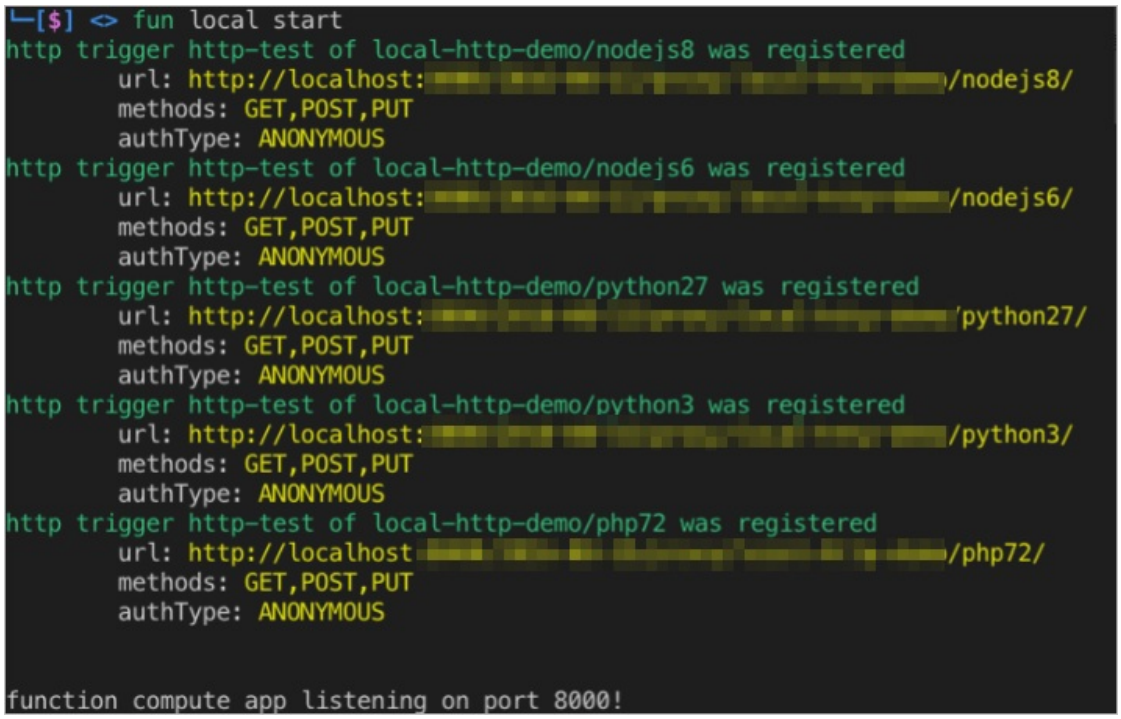
返回结果如下：

```
Usage: fun local start [options]
  Allows you to run the Function Compute application locally for quick development & testing.
  It will start an http server locally to receive requests for http triggers and apis.
  It scans all functions in template.yml. If the resource type is HTTP, it will be registered to this http server, whi
ch can be triggered by the browser or some http tools.
  For other types of functions, they will be registered as apis, which can be called by sdk in each language or dir
ectly via api.
  Function Compute will look up the code by CodeUri in template.yml.
  For interpreted languages, such as node, python, php, the modified code will take effect immediately, withou
t restarting the http server.
  For compiled languages such as java, we recommend you set CodeUri to the compiled or packaged localtion.
  Once compiled or packaged result changed, the modified code will take effect without restarting the http ser
ver.
 Options:
  -d, --debug-port <port>    specify the sandboxed container starting in debug mode, and exposing this port on
localhost
  -c, --config <ide/debugger>  output ide debug configuration. Options are vscode
  -h, --help             output usage information
```

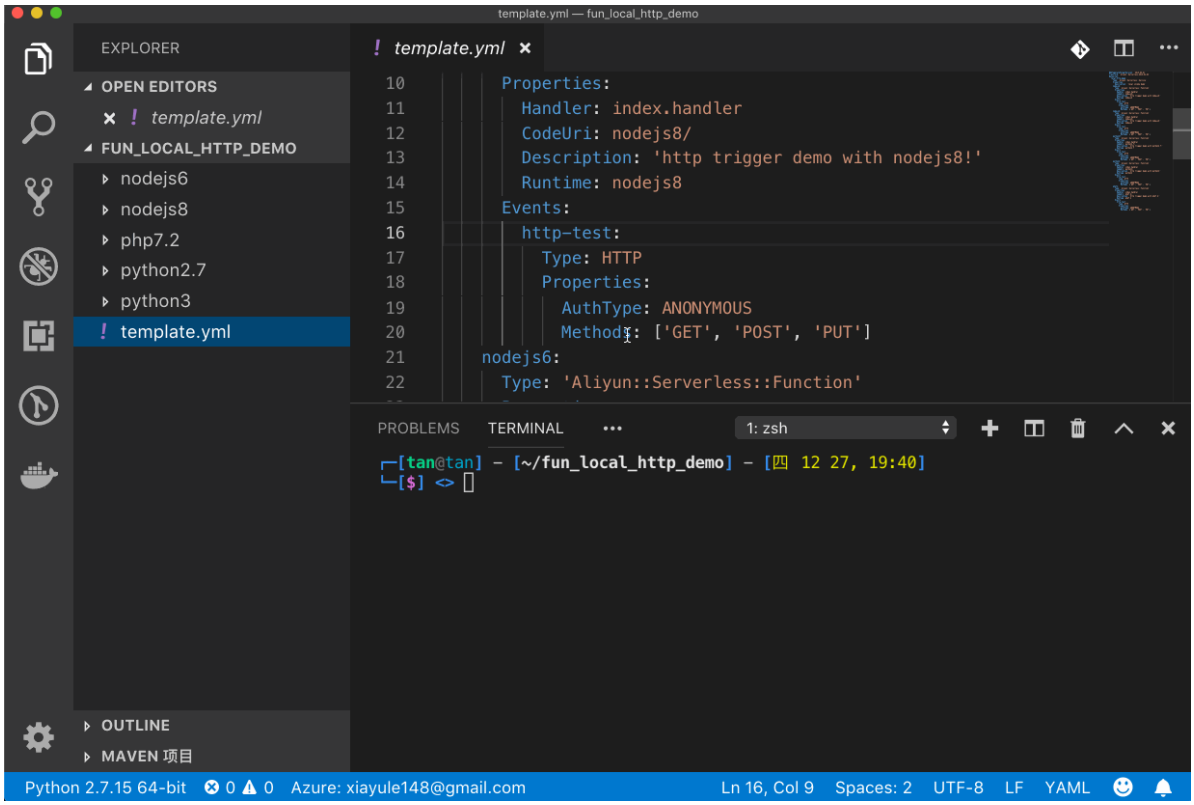
- 本地运行HTTP触发器函数
  您可以使用以下格式运行命令。其中options的取值请参见使用fun local进行本地调试，该参数可以省略。

  ```
  fun local start [options]
  ```

  执行 `fun local start` 后，Funcraft会首先启动一个HTTP Server提供HTTP的服务。然后Funcraft会扫描*template.yml*中描述的所有函数，并将所有配置HTTP触发器的函数，注册到HTTP Server中。注册成功后，您可以通过浏览器或者其他的HTTP工具打开任意一个URL访问。

您可以通过以下视频了解如何访问Node.js HTTP触发器。其他类型的Runtime例如Python、PHP使用体验类似。



- 本地调试HTTP触发器函数
  `fun local start` 支持 `-d`，`—debug-port` 选项。同时还支持 `-c`，`—config`，该命令可用于单步调试本地的HTTP函数。
  以下以Python为例，假设调试的函数名称为python3。其他类型与Python函数的调试过程一致。
  i. 您可以执行以下代码使用调试模式运行函数。

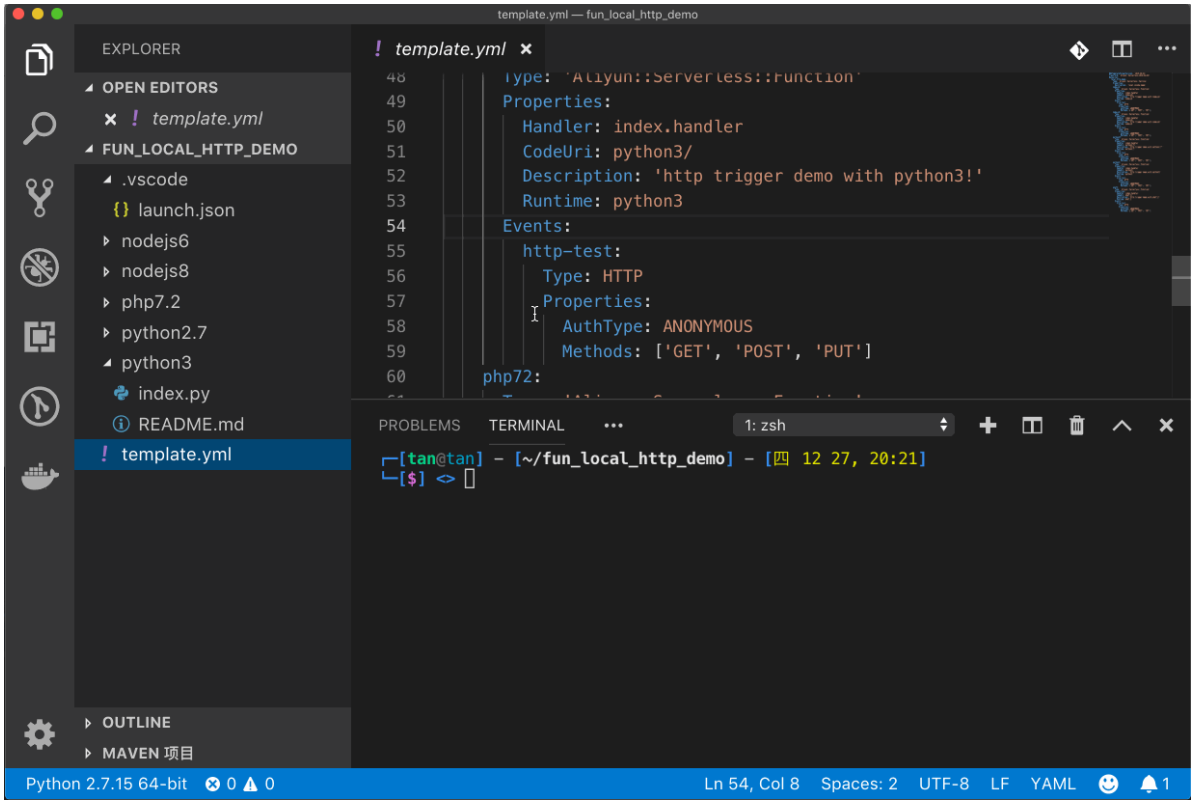
  ```
  fun local start —debug 3000 —config vscode
  ```

  ```
  └─[$] <> fun local start -d 3000 --config vscode
  http trigger http-test of local-http-demo/nodejs8 was registered
          url: http://localhost:[redacted]/nodejs8/
          methods: GET,POST,PUT
          authType: ANONYMOUS
  http trigger http-test of local-http-demo/nodejs6 was registered
          url: http://localhost:[redacted]/nodejs6/
          methods: GET,POST,PUT
          authType: ANONYMOUS
  http trigger http-test of local-http-demo/python27 was registered
          url: http://localhost:[redacted]/python27/
          methods: GET,POST,PUT
          authType: ANONYMOUS
  http trigger http-test of local-http-demo/python3 was registered
          url: http://localhost:[redacted]/python3/
  ```

  ii. 根据服务名、函数名或者HTTP触发器名称选择合适的URL，使用浏览器打开可以看到浏览器页面一直无响应，但在终端有日志输出。您需要将日志中的配置信息复制到VSCode调试器中，并在代码中下好断点，单击**开始调试**即可。

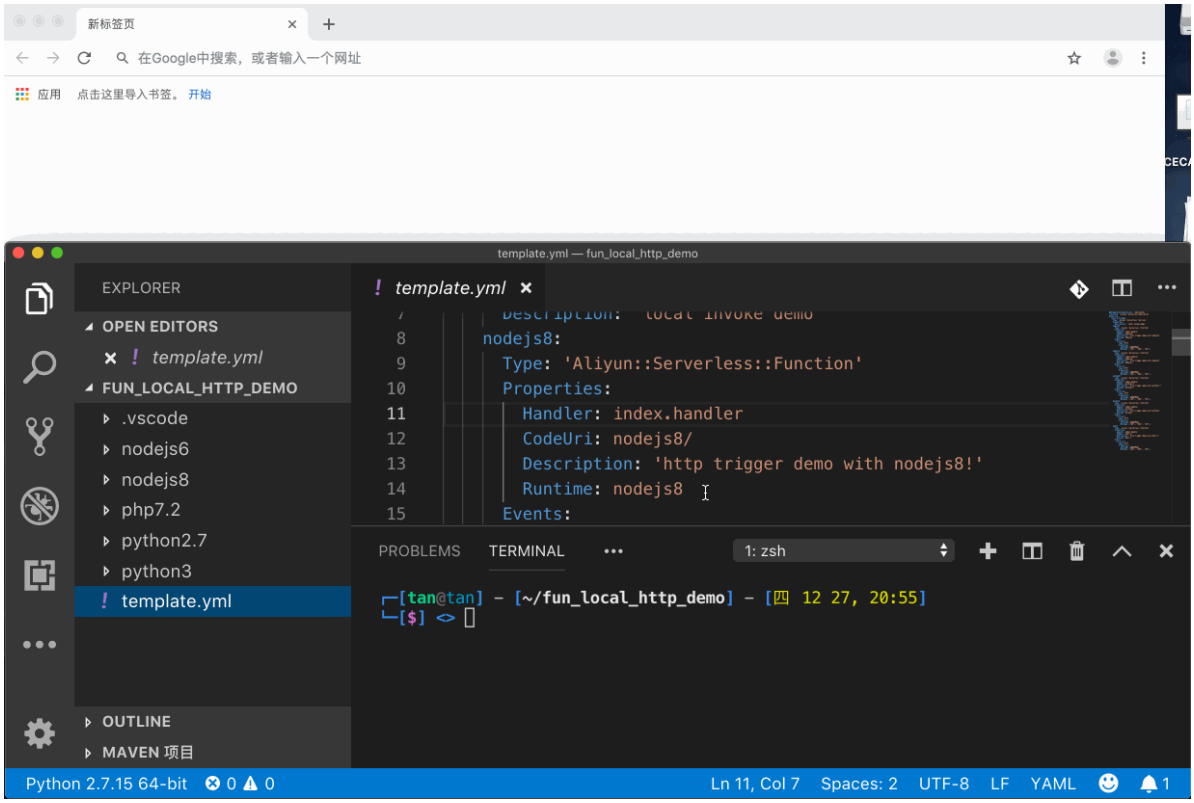
```
skip pulling image aliyunfc/runtime-python3.6:1.2.0...
you can paste these config to .vscode/launch.json, and then attach to your running function
////////////////// config begin //////////////////
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "fc/local-http-demo/python3",
      "type": "python",
      "request": "attach",
      "host": "localhost",
      "port": 3000,
      "pathMappings": [
        {
          "localRoot": "/Users/tan/fun_local_http_demo/python3",
          "remoteRoot": "/code"
        }
      ]
    }
  ]
}
////////////////// config end //////////////////
FunctionCompute python3 runtime inited.
FC Invoke Start RequestId: 04c57fba-cbe9-4c1f-8c57-f8e0b539fa08
```

您可以通过以下视频了解如何调试Python HTTP触发器的动态演示。其他类型的Runtime例如Node.js、PHP使用体验类似。



- 热加载
  本地运行、调试HTTP触发器时，支持热加载。

当通过 `fun local start` 启动本地服务后，即使修改了代码，也无需重启本地服务，可以直接刷新网页或者重新触发函数就可以运行变更后的函数。
您可以通过以下视频了解如何进行Nodejs的热加载。其他类型的Runtime例如PHP、Python类似。请提前在函数目录中执行 `npm install` 初始化函数依赖的Nodejs模块。



- API本地运行调试
  您可以使用以下格式运行命令。其中options可以省略。

  ```
  fun local start [options]
  ```

  执行 `fun local start` 命令后，Funcraft会首先启动一个HTTP Server，以提供HTTP的服务。然后Funcraft会扫描*template.yml*中描述的所有函数，并将所有没有配置HTTP触发器的函数，注册到HTTP Server中。注册成功后，就可以通过InvokeFunction接口或SDK进行调用。详情请参见 API定义和 SDK列表。
  您可以通过以下方式进行签名认证：
  - 通过API访问签名，实现签名认证，详情请参见签名认证。
  - 使用SDK进行调用，您可以通过以下命令将服务运行起来。

    ```
    fun local start
    ```

    返回结果如下。

```
api localdemo/php72 was registered
  url: http://localhost:8000/2016-08-15/services/localdemo/functions/php72/invocations/
api localdemo/python27 was registered
  url: http://localhost:8000/2016-08-15/services/localdemo/functions/python27/invocations/
api localdemo/python3 was registered
  url: http://localhost:8000/2016-08-15/services/localdemo/functions/python3/invocations/
api localdemo/nodejs6 was registered
  url: http://localhost:8000/2016-08-15/services/localdemo/functions/nodejs6/invocations/
api localdemo/nodejs8 was registered
  url: http://localhost:8000/2016-08-15/services/localdemo/functions/nodejs8/invocations/
api localdemo/java8 was registered
  url: http://localhost:8000/2016-08-15/services/localdemo/functions/java8/invocations/
function compute app listening on port 8000!
```

当服务运行成功后，您可以通过Python SDK进行调试。

a. 安装 FC Python SDK。

b. 执行以下代码安装依赖。

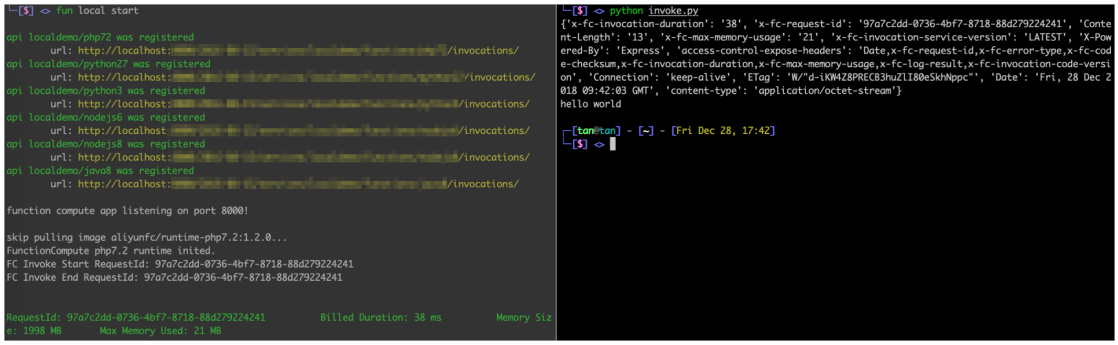
```
pip install aliyun-fc2
```

c. 编写代码。

```
import fc2
client = fc2.Client(endpoint='http://localhost:8000', accessKeyID='<your access key id>', accessKeySec
ret='your access key secret')
resp = client.invoke_function('localdemo', 'php72')
print resp.headers
print resp.data
```

> **说明** SDK代码中配置的AccessKeyID、AccessKey Secret要求与Funcraft配置一致，否则调用时，会导致签名认证失败。

执行结果如下图所示：



其他

- fun local invoke 访问宿主机上的服务
  如果您使用的是Docker-for-Mac或Docker-for-Windows 18.03+，您可以使用host.docker.internal作为IP访问本地服务。例如

```
client = fc2.Client(endpoint='http://host.docker.internal:8000', accessKeyID='xxx', accessKeySecret='xxx')
resp = client.invoke_function('localdemo', 'nodejs8', json.dumps({'code': 123}))
```

如果Docker版本小于18.03，您可以在不同类型的宿主机上执行以下命令查看宿主机的IP地址，然后配置到

Endpoint中即可。

- linnux or Mac：`ifconfig`
- Windows：`ipconfig`

- 环境变量
  在*template.yml*中配置的EnvironmentVariables会与线上行为一致，当函数运行时，您可以通过代码获取环境变量。更多信息请参见配置环境变量。

  > **说明** 您可以通过环境变量区分本地运行和线上运行，方便您进行一些特定的逻辑处理。

- Credentials
  您可以通过Credentials中存储的AccessKey ID、AccessKey Secret信息访问阿里云的其他服务。`fun local` 在本地运行函数时，会按照与 `fun deploy` 相同的策略寻找AccessKey ID、AccessKey Secret信息。
  关于函数计算Credentials的描述，您可以参考以下代码示例。以下代码根据本地、线上环境的不同，利用函数提供的Credentials配置OSS Client示例。

```
local = bool(os.getenv('local', ""))
if (local):
  print 'thank you for running function in local!!!!!!'
  auth = oss2.Auth(creds.access_key_id,
          creds.access_key_secret)
else:
  auth = oss2.StsAuth(creds.access_key_id,
            creds.access_key_secret,
            creds.security_token)
```

1.6. 安装第三方依赖

1.6.1. 使用fun install安装第三方依赖

本文介绍如何通过fun install命令及Funfile文本文件安装第三方依赖的操作步骤。

功能介绍

执行以下命令，获取 `fun install` 命令下的相关功能。

```
fun install -h
```

预期输出：

```
Usage: fun install [-f|--function <[service/]function>] [-r|--runtime <runtime>] [-p|--package-type <type>] [--save
] [-e|--env key=val ...] [packageNames...]
install dependencies which are described in fun.yml file.
Options:
 -f, --function <[service/]function>  Specify which function to execute installation task.
 -e, --env <env>              environment variable, ex. -e PATH=/code/bin (default: [])
 -d, --use-docker             Use docker container to install function dependencies
 -r, --runtime <runtime>          function runtime, avaliable choice is: nodejs6, nodejs8, nodejs10,
                      nodejs12, python2.7, python3, java8, php7.2, dotnetcore2.1, custom
 -p, --package-type <type>        avaliable package type option: pip, apt, npm.
 --save                   add task to fun.yml file.
 --registry <npm-url>             Configure npm to use any compatible registry, and even run your own
                      registry.
 --index-url <pip-url>            Base URL of Python Package Index (default https://pypi.org/simple). This should po
int to a repository compliant with PEP 503 (the simple repository API) or a local
                      directory laid out in the same format.
 -h, --help                   display help for command
Commands:
 init                     initialize fun.yml file.
 env                      print environment varables.
 sbox [options]               Start a local sandbox for installation dependencies or
                      configuration
```

fun install命令安装依赖

在项目目录中执行以下命令，安装Python环境下的PIP包。

```
fun install --runtime python2.7 --package-type pip tensorflow
```

预期输出：

```
skip pulling image aliyunfc/runtime-python2.7:build-1.2.0...
Task => [UNNAMED]
   => PYTHONUSERBASE=/code/.fun/python pip install --user tensorflow
```

参数说明如下：

- `--runtime`：指定运行的环境，您可以按需修改相应的运行环境。如果*fun.yml*里已经声明了运行环境，说明文件已经被初始化，您可以省略该选项。
- `--package-type`：指定安装依赖的类型。
- `tensorflow`：自定义PIP包的文件名称，本文以`tensorflow`为例。
- `PYTHONUSERBASE=/code/.fun/python pip install --user tensorflow`：命令在FC-Docker提供的容器内部执行，执行的命令会逐行打印出来。

> **注意** APT包依赖的安装方式及其参数和PIP包依赖类似，如果您需要安装APT包的依赖，您需要将`pip`修改为`apt`，APT的包名使用`apt-get`可以安装的DEB格式的包名即可。

PIP包安装成功后，项目目录中会生成一个*.fun*目录。

- *.fun/python/bin*：可执行文件被放置的目录。
- *.fun/python/lib/python2.7/site-packages*：库文件放置的目录。

文件存放结构如下所示：

```
.fun
└── python
    ├── bin
    │   ├── freeze_graph
    │   ├── markdown_py
    │   ├── pbr
    │   ├── saved_model_cli
    │   ├── tensorboard
    │   ├── tflite_convert
    │   ├── toco
    │   └── toco_from_protos
    └── lib
        └── python2.7
            └── site-packages
                ├── tensorboard
                ├── tensorboard-1.12.2.dist-info
                ├── tensorflow
                ├── tensorflow-1.12.0.dist-info
                ├── termcolor-1.1.0.dist-info
                ...
```

与 `pip install -t . <package-name>` 方式相比，使用 `fun install` 安装文件的优点如下：

- 存放位置更有组织性。
- 依赖文件和代码文件分离开，便于借助OSS或NAS初始化依赖文件。

安装成功后，库文件可能无法被程序找到，您需要自定义环境变量，关于如何配置函数计算的环境变量，请参见配置环境变量。您也可以执行以下命令获取相应的环境变量。

```
fun install env
```

预期输出：

```
LD_LIBRARY_PATH=/code/.fun/root/usr/local/lib:/code/.fun/root/usr/lib:/code/.fun/root/usr/lib/x86_64-linux-gnu:/code/.fun/root/usr/lib64:/code/.fun/root/lib:/code/.fun/root/lib/x86_64-linux-gnu:/code/.fun/root/python/lib/python2.7/site-packages:/code/.fun/root/python/lib/python3.6/site-packages:/code:/code/lib:/usr/local/lib
PATH=/code/.fun/root/usr/local/bin:/code/.fun/root/usr/local/sbin:/code/.fun/root/usr/bin:/code/.fun/root/usr/sbin:/code/.fun/root/sbin:/code/.fun/root/bin:/code:/code/node_modules/.bin:/code/.fun/python/bin:/code/.fun/node_modules/.bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/sbin:/bin
NODE_PATH=/code/node_modules:/usr/local/lib/node_modules
PYTHONUSERBASE=/code/.fun/python
```

> **说明** 如果您是使用 `fun local` 和 `fun deploy` 进行调试和部署的，您无需配置环境变量，Funcraft工具已经为您设定好了。

- 保存文件
  如果您需要将命令永久保存在*Funfile*文件中。您可以执行以下命令。

  ```
  fun install --runtime python2.7 --package-type pip --save tensorflow
  ```

  预期输出：

```
skip pulling image aliyunfc/runtime-python2.7:build-1.2.0...
Task => [UNNAMED]
=> PYTHONUSERBASE=/code/.fun/python pip install --user tensorflow
```

- 查看日志
  如果您需要查看详细日志。您可以执行以下命令。

  ```
  fun install -v
  ```

  预期输出：

  ```
  skip pulling image aliyunfc/runtime-python3.6:build-1.2.0...
  Task => [UNNAMED]
    => apt-get update (if need)
  Ign http://mirrors.aliyun.com stretch InRelease
  Get:1 http://mirrors.aliyun.com stretch-updates InRelease [91.0 kB]
  Get:2 http://mirrors.aliyun.com stretch-backports InRelease [91.8 kB]
  Get:3 http://mirrors.aliyun.com stretch/updates InRelease [94.3 kB]
  Hit http://mirrors.aliyun.com stretch Release.gpg
  Hit http://mirrors.aliyun.com stretch Release
  Get:4 http://mirrors.aliyun.com stretch-updates/main Sources [3911 B]
  ....
  ```

操作步骤

安装结果

（可选）相关操作

更多信息

执行 `fun install` 命令后安装的依赖会被安装到代码目录中，同时您也可以通过 `fun local` 的相关命令进行本地调试，也可以通过 `fun deploy` 一键部署到云端。相关信息，请参见使用fun local进行本地调试及使用fun deploy部署应用。

1.6.2. 使用Funfile文件安装第三方依赖

Funfile文件中包含了一系列命令，这些命令指定了构造函数计算交付物的步骤。Funcraft工具会执行这些命令，最终产出交付物，例如完成项目开发及打包编译后需要发布的内容。本文介绍如何通过Funfile文件安装第三方依赖。

前提条件

您已完成以下操作：

- 安装并配置Funcraft工具
- 安装并配置Docker

Funfile支持的语法

*Dockerfile*支持的命令，例如**COPY**、**RUN**、**ENV**、**USER**和**WORKDIR**等，*Funfile*也同样支持。Dockerfile用于构造Docker镜像，*Funfile*可以用于构造函数计算的交付产物。您可以通过安装rlang了解*Funfile*支持的一些语法命令，详细信息，请参见rlang。

相较于*Dockerfile*命令，*Funfile*只需要在原来的 `apt-get` 及 `pip` 安装包的命令前加 `fun-install` 前缀即可。

使用Funfile的优势

- 标准化：通过Funfile文件描述安装依赖的详细步骤，有利于标准化开发流程。
- 便捷性：通过将Funfile文件构建而不是将具体的某些依赖提交到版本控制系统中，也更有利于函数代码的维护。
- 原生态：Funfile的编写，与Dockerfile的编写类似，却更接近原生体验。结合 `fun install sbox` ，您可以先在交互式环境中进行各种尝试，然后将最终安装步骤编写到*Funfile*文件内。
- 高效性：Funfile也支持Dockerfile的Cache功能，如果项目的改动对Funfile不影响，那么Funfile可以利用Cache直接跳过相应的命令执行过程。这有助于加速对Funfile的执行过程，提升开发效率。

操作步骤

1. 构建*Funfile*文件。
   i. 在项目目录中执行**fun install init**命令，初始化构建安装*Funfile*文件。
   ii. 选择函数的运行环境。

```
? Select a runtime (Use arrow keys)
> nodejs6
  nodejs8
  nodejs10
  nodejs12
  python2.7
  python3
  java8
(Move up and down to reveal more choices)
```

   以安装 `python2.7` 为例，介绍如何安装第三方依赖。当选择 `python2.7` 后，会在当前目录中生成一个名为*Funfile*的文件，内容如下：

```
RUNTIME python2.7
```

   > **说明** 由于*Funfile*是一个文本文件，您可以直接创建*Funfile*的文件，按需修改运行环境。

2. 使用*Funfile*文件安装pip依赖。
   i. 安装PIP依赖时需要修改*Funfile*文件，内容如下。

```
RUNTIME python2.7
RUN fun-install pip install flask
```

   > **说明** 如果需要安装APT依赖，您只需将 `pip` 修改为 `apt-get` 即可。

ii. 执行以下命令安装依赖。

```
fun install
```

预期输出：

```
using template: template.yml
start installing function dependencies without docker
building pyzbar-srv/pyzbar-fun
Funfile exist, Fun will use container to build forcely
Step 1/2 : FROM aliyunfc/runtime-python3.6:build-1.6.1
 ---> 2110e9fa9393
Step 2/2 : RUN fun-install pip install flask
 ---> Using cache
 ---> 992b0d65136d
sha256:992b0d65136dfa63b9566b8280881bb4cf22c4f1b68155713545e4b61ada8dd9
Successfully built 992b0d65136d
Successfully tagged fun-cache-e95a9bfa-a6fa-4d70-b5b4-0084613e2b8d:latest
copying function artifact to /Users/tan/code/fun/examples/install/pyzbar_example
could not find any manifest file for pyzbar-srv/pyzbar-fun, [install] stage for manifest will be skipped
Tips for next step
======================
* Invoke Event Function: fun local invoke
* Invoke Http Function: fun local start
* Build Http Function: fun build
* Deploy Resources: fun deploy
```

更多信息

执行fun install命令后会将依赖安装到代码目录中。同时，您可以通过fun local的相关命令进行本地调试，也可以通过fun deploy将项目部署到云端。

1.7. 使用fun build构建代码包

本文介绍如何通过 `fun build` 将源码构建成交付产物。

背景信息

源码并不等于交付产物，例如Java型编译语言，您写好Java型代码后，不能直接上传代码，您还需要进行编译及打包，编译及打包的结果属于交付产物。由于函数计算只能接受一个ZIP包或JAR包，所以您需要将代码交付产物及其依赖一同打包到一个JAR包中，然后交付。

使用 `fun build` 的构建过程可以理解为，Funcraft工具在函数代码目录中查找特定的*manifest*文件（清单文件）并根据*manifest*文件进行构造、下载依赖及编译。

目前Funcraft工具针对不同的语言分别支持一些主流的包管理器*manifest*文件。

- Funcraft定义的不限语言的*Funfile*文件。更多信息，请参见使用fun install安装第三方依赖。
- Java maven包管理器的*pom.xml*文件。
- Node.js npm包管理器的*package.json*文件。
- Python pip包管理器的*requirements.txt*文件。
- PHP composer包管理器的*composer.json*文件。

fun build功能介绍

执行以下命令获取 `fun build` 的相关功能介绍。

```
fun build -h
```

预期输出：

```
Usage: fun build [options] [[service/]function]
Build the dependencies.
Options:
 -d, --use-docker        Use docker container to build functions
 -t, --template [template]  The path of fun template file.
 -h, --help              display help for command
```

操作步骤

本文以Java8为例，介绍如何构建交付产物。

1. 在函数根目录下新建一个*template.yml*文件，例如*/tmp/code/template.yml*，具体信息，请参见template.yml，文件内容修改如下。

   ```
   ROSTemplateFormatVersion: '2015-09-01'
   Resources:
    Services:
     Type: 'Aliyun::Serverless::Service'
     Properties:
      Policies:
       - AliyunOSSFullAccess
     Function:
      Type: 'Aliyun::Serverless::Function'
      Properties:
       Handler: main.main_handler
       Runtime: java8
       CodeUri: './'
   ```

   *template.yml*的含义如下：声明一个名为 `Services` 的服务，在这个服务下，声明一个名为 `Function` 的函数，配置函数入口为 `main.main_handler` ，函数的Runtime为 `java8` ，并指定 `CodeUri` 为当前目录。在部署时，Funcraft会将 `CodeUri` 指定的目录打包上传。
   相关信息，请参见：

   - 函数计算服务属性
   - 函数计算函数属性
   - CodeUri说明
   - 更多的配置规则

2. 您可以通过以下方式构建源代码：
   - 直接执行以下命令构建。

     ```
     fun build
     ```

   - 指定函数名称进行构建， `<your functionName>` 需替换为您的函数的名称，例如 `Function` 。

     ```
     fun build <your functionName>
     ```

   - 指定函数及服务名称进行构建， `<your serviceName>/<your functionName>` 需替换为您的服务及函数的名称，例如 `Services/Function` 。

     ```
     fun build <your serviceName>/<your functionName>
     ```

     以上三种构建方式的预期输出一致，如下所示。

```
using template: template.yml
start building function dependencies without docker
building Services/Function
Build Success
Built artifacts: .fun\build\artifacts
Built template: .fun\build\artifacts\template.yml
Tips for next step
======================
* Invoke Event Function: fun local invoke
* Invoke Http Function: fun local start
* Deploy Resources: fun deploy
```

> **说明**
>
> - 默认情况下，`fun build`可以直接在宿主机上构建，如果想避免因为环境差异带来部署后的不兼容问题，可以在构建时指定-d或--use-docker参数，表示会在本地使用函数计算模拟环境进行编译。
> - 如果*template.yml*不在当前目录下，您可以通过`fun build -t templatePath`的方式指定。
> - `fun build`是将*template.yml*中列出的函数进行构建，如果不指定具体函数，则默认编译所有函数。

更多信息

您可以通过以下方式进行构建或其他操作：

- 通过以下命令对*template.yml*文件中的所有函数进行构建。

  ```
  fun build
  ```

- 通过以下命令以本地模拟环境对所有函数进行构建。

  ```
  fun build --use-docker
  ```

- 通过以下命令进行构建及调用。

  ```
  fun build && fun local invoke
  ```

- 通过以下命令进行构建及部署。

  ```
  fun build && fun deploy
  ```

构建示例

以下是一个初始化、构建、运行及修改Java的示例。

1.8. 使用fun nas管理NAS文件

本文介绍如何配置NAS及其依赖资源，以及在配置成功后如何通过 `fun nas` 的相关命令管理NAS文件。

背景信息

- 通过 `NasConfig: Auto` 一键配置NAS及其依赖的VPC资源
  在使用函数计算过程中，可以通过NAS来存放一些体积较大且不易变动的资源，配置NAS及其资源后，可以让指定服务下的函数访问NAS文件系统时如同访问本地文件系统一样，这使得即使在依赖比较大的场景下，依旧可以借助函数计算来快速开发一个Serverless应用。
  但是NAS在帮助函数计算解决大依赖问题的同时，由于其自身的配置比较复杂，也增加了函数计算的使用难度。为了简化NAS的使用体验，Funcraft工具添加了一个新的配置项 `NasConfig: Auto` ，通过该属性，您可以一键配置NAS及其依赖的VPC资源。
- 通过 `fun nas` 管理NAS文件以及上传本地NAS文件
  即使配置好了NAS服务，仍需面对上传本地资源到NAS文件系统中等问题，上传资源到NAS文件系统中通常有以下几种方案：
  - 购买一台ECS，然后挂载NAS，通过FTP或者网盘等实现将文件上传到NAS。
  - 使用在线迁移服务，将OSS资源迁移到NAS。更多信息，请参见如何将阿里云OSS的资源迁移至阿里云NAS。
  - 借助函数计算，开发一个Serverless的NAS文件管理应用。更多信息，请参见开发NAS文件管理应用。
  - 通过 `fun nas` 命令管理NAS文件及上传本地NAS文件。
    `fun nas` 本身实现思路是上述借助函数计算，开发一个Serverless的NAS文件管理应用的升级版，解决了以下问题，例如：
    - 需要手动部署一套服务。
    - 不支持目录上传。
    - 不支持大于6 MB的文件上传。

fun nas功能介绍

执行以下命令获取 `fun nas` 的功能介绍。

```
fun nas -h
```

预期输出：

```
Usage: fun nas [options] [command]
Operate your remote NAS file system locally.
Options:
 -h, --help     display help for command
Commands:
 info        Print nas config information, such as local temp directory of NAS.
 init        For each service with NAS config, create local NAS folder and deploy fun nas server service.
 sync        Synchronize the local NAS directory with the remote NAS file system.
 ls          List contents of remote NAS directory
 rm          Remove remote NAS file.
 cp          copy local file/folder to remote NAS
 help [command]  display help for command
```

配置NAS

本文主要介绍两种方式配置NAS，成功配置后您的函数可以访问指定的NAS资源，建议您使用方式二配置NAS。

使用控制台配置NAS及其依赖资源。

1. 登录VPC控制台，创建专有网络、交换机及安全组，更多信息，请参见专有网络和交换机概述。
2. 登录NAS控制台，创建文件系统及添加挂载点，更多信息，请参见入门引导。
3. 编写template.yml，复制上面步骤创建好的资源，并编写VpcConfig及NasConfig的配置后保存文件。

   ```
   ROSTemplateFormatVersion: '2015-09-01'
   Transform: 'Aliyun::Serverless-2018-04-03'
   Resources:
    nasDemo:
     Type: 'Aliyun::Serverless::Service'
     Properties:
      VpcConfig:
       VpcId: 'vpc-bp12hm92gdpcjtai7****'
       VSwitchIds: [ 'vsw-bp1gitru7oicyyb4u****' ]
       SecurityGroupId: 'sg-bp1243pi65bw4cjj****'
      NasConfig:
       UserId: 10003
       GroupId: 10003
       MountPoints:
         - ServerAddr: '012194b28f-ujc20.cn-hangzhou.nas.aliyuncs.com:/'
           MountDir: '/mnt/nas'
   ```

   完成上述步骤后，您就可以使用fun local和fun deploy进行部署和调试了。

使用*template.yml*文件及 `fun deploy` 命令配置NAS及其依赖资源。

> **说明** 如果您需要使用NAS及其依赖资源，但是又不想涉及专有网络、交换机及挂载点等，建议您使用方式二进行配置。

1. 在函数根目录下新建一个*template.yml*文件，具体信息，请参见template.yml，使用 `NasConfig: Auto` 服务的示例修改文件内容。

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 nasDemo:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   NasConfig: Auto
```

2. 在根目录中执行 `fun deploy` 命令。
预期输出：

```
Waiting for service nasDemo to be deployed...
  ... ...
  using 'VpcConfig: Auto', Fun will try to generate related vpc resources automatically
    ... ...
  generated auto VpcConfig done: {"vpcId":"vpc-bp1pcr7o8ksmgojt4****","vswitchIds":["vsw-bp13qdw9qe1i2it4y****"],"securityGroupId":"sg-bp161u0547x0lq5k****"}
  using 'NasConfig: Auto', Fun will try to generate related nas file system automatically
    ... ...
  generated auto NasConfig done: {"UserId":10003,"GroupId":10003,"MountPoints":[{"ServerAddr":"****c4a7ac-vrk7.cn-hangzhou.nas.aliyuncs.com:/nasDemo","MountDir":"/mnt/auto"}]}
  Checking if nas directories /mnt/auto/nasDemo exists, if not, it will be created automatically
  Checking nas directories done ["/mnt/auto/nasDemo"]
  ... ...
service nasDemo deploy success
```

> 说明
> - `NasConfig: Auto` 的部署效果是幂等的，即如果依赖的相关资源不存在，则会自动创建，如果相关资源已经存在，则会直接使用相关资源。
> - `fun local` 、 `fun deploy` 均支持 `NasConfig: Auto` 配置。
> - 使用 `NasConfig: Auto` 时，NAS文件系统会被自动挂载到函数计算运行环境的 `/mnt/auto` 目录中。

方式一

方式二

相关操作

您可以通过以下不同的命令，了解 `fun nas` 的相关功能。

您需要在函数根目录下新建一个*template.yml*文件，具体信息，请参见template.yml，使用 `NasConfig` 服务示例修改文件内容，如下所示。

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 nasDemo:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   NasConfig: Auto
```

- 在根目录中执行以下命令，初始化配置：

```
fun nas init
```

预期输出：

```
start fun nas init...
Waiting for service fun-nas-NasDemo to be deployed...
 ... ...
  using 'VpcConfig: Auto', Fun will try to generate related vpc resources automatically
 ... ...
  generated auto VpcConfig done: {"vpcId":"vpc-bp1pcr7o8ksmgojt4****","vswitchIds":["vsw-bp13qdw9qe
1i2it4y****"],"securityGroupId":"sg-bp161u0547x0lq5k****"}
  using 'NasConfig: Auto', Fun will try to generate related nas file system automatically
 ... ...
  generated auto NasConfig done: {"UserId":100**,"GroupId":100**,"MountPoints":[{"ServerAddr":"****c4a
6d9-gry74.cn-hangzhou.nas.aliyuncs.com:/fun-nas-NasDemo","MountDir":"/mnt/auto"}]}
  Checking if nas directories /mnt/auto/fun-nas-NasDemo exists, if not, it will be created automatically
  Checking nas directories done ["/mnt/auto/fun-nas-NasDemo"]
 ... ...
service fun-nas-NasDemo deploy success
Fun nas init Success
Tips for next step
======================
$ fun nas info    # Show NAS info
$ fun nas ls      # List NAS files
$ fun nas sync    # Synchronize files to nas
$ fun deploy
# Deploy Resources
```

> **说明** 您执行 `fun nas init` 命令后，NAS及VPC相关资源已成功创建，如果您修改了NAS或VPC的相关配置，您只需重新执行 `fun nas init` 或 `fun vpc init` 命令进行一次初始化配置即可。

- 在根目录中执行以下命令查看NAS目录。

```
fun nas info
```

预期输出：

```
Local NAS folder of service NasDemo includes:
/NasDemo/.fun/nas/auto-default/NasDemo
```

- 在根目录中执行以下命令同步本地NAS资源信息，如果本地资源有变更，您只需重新执行以下命令即可。

```
fun nas sync
```

预期输出：

```
Starting upload /NasDemo/.fun/nas/auto-default/NasDemo to nas://NasDemo:/mnt/auto/
NAS path checking...
zipping /NasDemo/.fun/nas/auto-default/NasDemo
 /NasDemo/.fun/nas/auto-default/.NasDemo.zip - zipped
generate tmpDir: /NasDemo/.fun/nas/auto-default/.fun_nas_tmp/42658c92eef63f2b3b13e5f46315ee7d
Split zip file to 1 small files
Uploading...
Upload /NasDemo/.fun/nas/auto-default/NasDemo to /mnt/auto/ done!
```

- 您可以在根目录中通过以下两种方式命令查看NAS服务中的文件。
  - 指定列出某个服务下的NAS路径内容：`fun nas ls nas://ServiceName/Path`。执行以下命令查看NAS服务中的资源。

    ```
    fun nas ls -a nas://NasDemo/mnt/auto
    ```

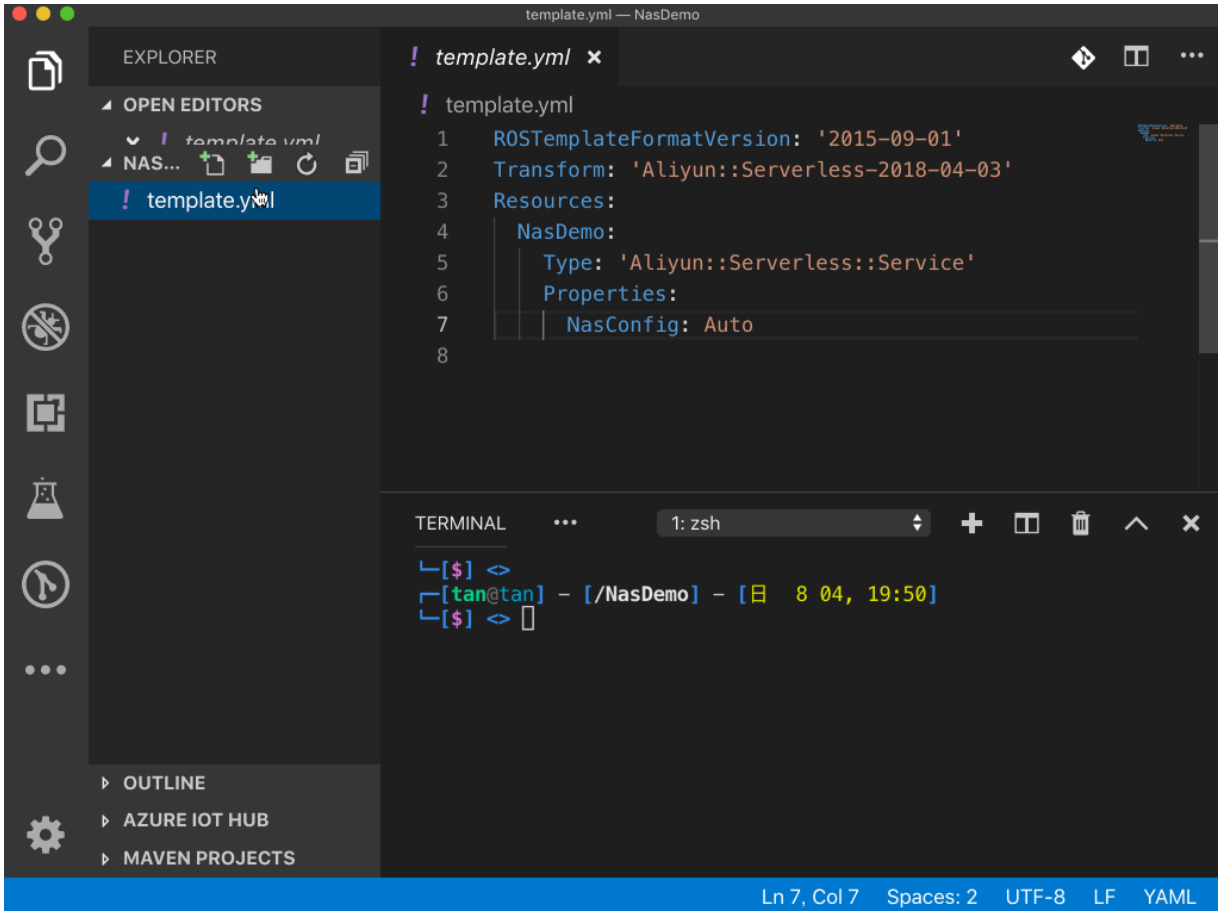
    预期输出：

    ```
    total 10
    drwxr-xr-x 4 user10003 10003 4096 Aug  4 11:38 .
    drwxr-xr-x 1 root      root  4096 Aug  4 11:37 ..
    drwxr-xr-x 4 user10003 10003 4096 Aug  4 11:38 .fun_nas_tmp
    -rw-r--r-- 1 user10003 10003   19 Aug  4 11:38 test
    drwxr-xr-x 2 user10003 10003 4096 Aug  4 11:38 testDir
    ```

  - 如果模板文件*template.yml*的文件中只包含一个服务，您可以将ServiceName省略掉，即`fun nas ls nas:///Path`，执行以下命令查看NAS服务中的资源。

    ```
    fun nas ls -a nas://mnt/auto
    ```

fun nas相关命令的动态演示



1.9. 使用fun deploy部署应用

本文介绍如何通过fun deploy命令或ROS方式将应用部署到云上。

前提条件

您已成功完成以下操作：

- 创建存储空间
- （可选）创建资源栈

背景信息

Funcraft工具引入了一种全新的部署方式即ROS部署，使用ROS的部署方式使部署具备更完善的资源状态管理机制、回滚机制等优点。

优势	描述
具备更完善的资源状态管理机制	通过ROS部署应用时，使Funcraft工具能够准确的感知资源的新增、更新、删除以及不变，通过感知这些资源的状态变化，Funcraft可以做出正确的决策，达到预期的行为。
具备回滚机制	在部署应用的过程中，任何资源的操作失败，都会导致整个资源栈的回滚。通过ROS部署应用，当部署失败时可以回滚到上一个正确的状态，解决部署过程中服务不可用的问题。
灵活定义模型	通常一个*template.yml*用来描述一个完整的应用，您看到的*template.yml*内容，就是您部署到线上应用的最终形态，通过ROS部署可以更加灵活的部署应用。
明确部署差异	通过ROS部署应用，您可以了解到当前的模板文件是否与线上的文件同步，因此可以快速决策是否直接将本地模板部署到云上。
支持更多的云资源	ROS纳入了更多的云资源的管理，覆盖了使用者的全部使用场景。
支持快速创建多套开发环境	当需要创建多套开发环境时，无需手动修改资源的唯一名称等标识。

ROS支持在模板文件里描述更多资源。除了原有Funcraft定义的资源外，ROS支持的资源也可以在Funcraft的模板文件里声明。更多信息，请参见资源编排服务ROS。例如通过以下方式声明一个对象存储的Bucket实例。

```
Bucket:
 Type: 'ALIYUN::OSS::Bucket'
 Properties:
  BucketName: 'oss-unzip-demo-bucket' //需替换成您真实的Bucket名称。
```

功能介绍

执行以下命令获取 `fun deploy` 的帮助信息。

```
fun deploy -h
```

预期输出：

```
Usage: fun deploy [options] [resource]
 Deploy a serverless application.
 use 'fun deploy' to deploy all resources
 use 'fun deploy serviceName' to deploy all functions under a service
 use 'fun deploy functionName' to deploy only a function resource
 with '--only-config' parameter, will only update resource config without updating the function code
 use '--parameter-override', A parameter structures that specify input parameters for your stack template.
                If you're updating a stack and you don't specify a parameter, the command uses the stack's existi
ng value.
                For new stacks, you must specify parameters that don't have a default value. Syntax: parameterk
ey=parametervalue.
Options:
 -t, --template [template]         The path of fun template file.
 -c, --only-config                 Update only configuration flags
 -p, --parameter-override <parameter>  A parameter structures that specify input parameters for your stack te
mplate.
 -y, --assume-yes                  Automatic yes to prompts. Assume "yes" as answer to all prompts and run non-inter
actively.
 --use-ros                         Deploy resources using ROS
 --use-nas                         Automatically upload local resources to NAS.
 --stack-name <stackName>          The name of the ROS stack
 --push-registry <pushRegistry>    Modify the image upload path
 -h, --help                        display help for command
```

fun deploy命令部署

1. 在项目目录中创建一个*index.js*文件，并将以下内容复制到*index.js*文件中保存。

```
var getRawBody = require('raw-body')
module.exports.handler = function (request, response, context) {
  // 获取请求体
  getRawBody(request, function (err, body) {
    var respBody = {
      headers: request.headers,
      url: request.url,
      path: request.path,
      queries: request.queries,
      method: request.method,
      clientIP: request.clientIP,
      body: body.toString()
    };
    response.setStatusCode(200);
    response.setHeader('content-type', 'application/json');
    response.send(JSON.stringify(respBody, null, 4));
  });
};
```

2. 在项目目录下创建一个*template.yml*文件，将以下内容复制到*template.yml*文件中保存。

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 local-http-test:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   Description: 'local invoke demo'
  nodejs8:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Handler: index.handler
    CodeUri: './'
    Description: 'http trigger demo with nodejs8!'
    Runtime: nodejs8
   Events:
    http-test:
     Type: HTTP
     Properties:
      AuthType: ANONYMOUS
      Methods: ['GET', 'POST', 'PUT']
```

> **说明** 如果您有需要忽略的文件或文件夹，例如不需要打包上传的函数代码ZIP包的文件，您可以在项目中创建*.funignore*文件，然后将不需要使用或需要忽略的文件名写入到*.funignore*文件中即可。

3. 代码以及模板文件编写完成后，执行以下命令将服务部署到线上环境。

```
fun deploy
```

预期输出：

```
using region: cn-shanghai
using accountId: ***********8320
using accessKeyId: ***********1EXB
using timeout: 10
Waiting for service local-http-test to be deployed...
    Waiting for function nodejs8 to be deployed...
        Waiting for packaging function nodejs8 code...
        package function nodejs8 code done
        Waiting for HTTP trigger http-test to be deployed...
        methods: GET
        url: https://***************.cn-shanghai.fc.aliyuncs.com/2016-08-15/proxy/local-http-test/nodejs8
/
        function http-test deploy success
    function nodejs8 deploy success
service local-http-test deploy success
```

ROS方式部署

1. 执行以下命令初始化示例项目。

```
fun init event-nodejs8 -n RosDemo
```

   - `event-nodejs8`：指定应用模板。

- `RosDemo`：自定义部署项目的文件名称。

2. 在项目目录中，执行以下命令将资源上传到对象存储中，实现项目打包。

```
fun package --oss-bucket <bucketname>
```

`<bucketname>` 需替换为Bucket的名称。

> 说明
> - Bucket的地域需和创建的函数在同一个地域内。
> - 由于ROS部署无法直接使用本地的代码资源，要求资源必须“云化”，需先执行fun package命令将资源上传到对象存储中。

3. 执行以下命令部署项目。

```
fun deploy --use-ros --stack-name <stackName>
```

参数说明：

- `--use-ros`：表示使用ROS的方式进行部署。
- `--stack-name`：表示使用ROS部署时使用的资源栈名称。资源栈指使用模板文件部署的一整套应用即云端资源集合的ID。即使同一个模板文件，也可以通过指定不同的资源栈名称，实现部署多套应用。当通过模板文件进行部署时，会将模板文件的资源属性与所指定的资源栈进行对比并计算差异，进而完成线上资源的创建、删除及更新等操作。

> 说明 `--stack-name <stackName>` 即可以自定义也可以省略还可以使用已成功创建的资源栈名称，当省略或自定义时系统会自动创建资源栈。

当执行成功后，您可以登录函数计算控制台查看到成功创建的服务及函数。

操作步骤

1.10. 部署API网关

本文介绍如何通过Funcraft部署API网关和函数计算应用，如果应用比较简单，您可以直接复制提供的示例的配置，如果提供的示例无法满足需求，您可以通过本文介绍的映射规则进行自定义配置。

背景信息

函数计算提供了HTTP触发器，并且支持本地调试，方便您通过函数计算开发Web应用，请参见HTTP触发器概述和使用fun local进行本地调试。但也存在一些场景，例如已经存在一个API网关+函数计算的应用，想要将该应用部署到不同的地域。如果通过手工操作，这个过程会十分的繁琐，且容易出错。通过Funcraft可以较容易的实现一键部署API网关+函数计算到所有地域。

在通过Funcraft发布API网关和函数计算应用前，建议您先了解Funcraft工具的相关文档，请参见功能概览。

> 说明 使用Funcraft配置API网关具有重复部署、多地域部署、协同开发、版本控制管理等优点，建议您优先使用Funcraft部署API网关。本文只提供核心的代码或配置，若您需要查看完整示例，请参见参考文档。

本文涉及的名词解释如下：

- 函数计算FC（Function Compute）：函数计算是事件驱动的全托管计算服务。使用函数计算，您无需采购与管理服务器等基础设施，只需编写并上传代码。函数计算准备计算资源，并以弹性伸缩的方式运行您的代码，而您只需为任务实际消耗进行付费。函数计算更多信息，请参见什么是函数计算。

- Funcraft：Funcraft是一个用于支持Serverless应用部署的工具，能帮助您便捷地管理函数计算、API网关、日志服务等资源。您可以通过一个资源配置文件*template.yml*进行开发、构建、部署等操作。Funcraft的更多文档，请参见介绍。
- API网关：API网关（API Gateway）提供高性能、高可用的API托管服务，帮助您对外开放其部署在ECS、容器服务等阿里云产品上的应用，提供完整的API发布，维护生命周期管理等。您只需进行简单地操作，即可快速、低成本、低风险地开放数据或服务。

简单示例

1. 您可以执行以下代码编写函数。函数既可以是一个简单的`hello world`，也可以是一段更复杂的程序涉及到的代码，更多信息，请参见使用函数计算做为API后端服务。

```
exports.handler = function(event, context, callback) {
 var response = {
   isBase64Encoded: false,
   statusCode: 200,
   body: 'hello world'
 };
 callback(null, response);
};
```

2. 您可以添加以下配置，配置API网关。

```
HelloworldGroup: # Api Group
 Type: 'Aliyun::Serverless::Api'
 Properties:
  StageName: RELEASE
  DefinitionBody:
   '/': # request path
    get: # http method
     x-aliyun-apigateway-api-name: hello_get # api name
     x-aliyun-apigateway-fc: # 当请求该API时，要触发的函数，ARN: acs:fc:::services/${fc.Arn}/functions/${hel
loworld.Arn}/
      timeout: 3000
```

> 说明 `x-aliyun-apigateway-fc`配置项中的`fc`以及`helloworld`，分别为服务名、函数名，请您根据实际配置的函数进行修改。

3. 您可以通过fun deploy命令完成发布，同时您也可以通过提示的URL完成访问。

```
Waiting for service fc to be deployed...
        Waiting for function apigateway to be deployed...
                Waiting for packaging function apigateway code...
                package function apigateway code done
        function apigateway deploy success
service fc deploy success

Waiting for api gateway HelloworldGroup to be deployed...
    URL: GET http://[redacted].com/
      stage: RELEASE, deployed, version: 20190128171519485
      stage: PRE, undeployed
      stage: TEST, undeployed
api gateway HelloworldGroup deploy success
```

使用进阶

如果您想要配置更多参数，例如HTTPS、超时时间等，您可以根据API网关的API文档进行更多的配置，Funcraft支持的参数包括ApiName、Visibility、RequestConfig、RequestParameter、ServiceParameter、ServiceParametersMap等。详细信息，请参见创建API。

如果您需要通过API文档，编写Funcraft配置文件，您需要了解关于API与Funcraft配置的映射规则。映射规则如下：

- 参数名需小写。
- 原来的驼峰需用 - 连接。
- 前缀需加上 x-aliyun-apigateway 。

例如对于以下API参数，映射关系分别为：

```
ApiName => x-aliyun-apigateway-api-name
Visibility => x-aliyun-apigateway-visibility
RequestParameters => x-aliyun-apigateway-request-parameters
ServiceParametersMap => x-aliyun-apigateway-service-parameters-map
```

如果您想对每个参数执行更细粒度的配置，例如在API文档中介绍的RequestParameter，表示RequestParameter数组的字符串，RequestParameter可进行配置的参数有ApiParameterName、Location、ParameterType、Required等。您可以参照以下示例将RequestParameter的参数转换成Funcraft配置。

```
ApiParameterName => apiParameterName
Location => location
```

映射示例

您可以通过以下配置将RequestParameter转换成Funcraft的配置。

```
x-aliyun-apigateway-request-parameters:
 - apiParameterName: 'token'
   location: 'Query'
   parameterType: 'String'
   required: 'REQUIRED'
 - apiParameterName: 'token2'
   location: 'Query'
   parameterType: 'String'
   required: 'REQUIRED'
```

常量参数、系统参数

您可以根据API文档及映射规则完成相应的参数配置。

- 常量参数的配置如下：

```
x-aliyun-apigateway-request-parameters:
 - apiParameterName: 'token'
   location: 'Query'
   parameterType: 'String'
   required: 'REQUIRED'
   defaultValue: 'e'
x-aliyun-apigateway-service-parameters:
 - serviceParameterName: 'token'
   location: 'Query'
   parameterCatalog: 'CONSTANT'
```

- 系统参数的配置如下：

```
x-aliyun-apigateway-request-parameters:
 - apiParameterName: 'CaClientIp'
   location: 'Query'
   parameterType: 'String'
   required: 'REQUIRED'
x-aliyun-apigateway-service-parameters:
 - serviceParameterName: 'CaClientIp'
   location: 'Query'
   parameterCatalog: 'SYSTEM'
```

改造已有的API

如果一个API已经存在一个API网关，您可以通过以下操作快速地用Funcraft描述出来。

1. 登录API网关控制台。
2. 在左侧导航栏，选择**开放API > API列表**，选择目标API，单击**编辑**，然后不做修改直接保存。
3. 找到**ModifyApi.json**请求，您可以根据映射规则修改RequestParamter等参数的配置。

2.fcli

2.1. 初次使用fcli

fcli是阿里云函数计算的命令行工具，帮助您便捷地管理函数计算中的资源。若您是初次使用fcli，您需要下载并配置fcli。

下载fcli

下载fcli安装包至本地，然后解压安装包。

配置fcli

如果您是第一次使用，需要配置fcli。函数计算提供了以下几种配置方式：

使用 `fcli shell` 命令配置。

1. 在fcli可执行文件所在的文件夹下，执行 `fcli shell` 。
2. 根据界面提示，依次输入阿里云主账号的账号ID、AccessKey ID和AccessKey Secret，然后选择地域。
   您可以在账号管理中获取账号的Account ID，在用户信息管理中获取AccessKeyID和AccessKeySecret。

   ```
   ? Alibaba Cloud Account ID <your account ID>
   ? Alibaba Cloud Access Key ID <your AccessKey ID>
   ? Alibaba Cloud Access Key Secret <your Accesskey Secret>
   ? Default region name cn-shanghai
   Store the configuration in: .fcli
   Welcome to the function compute world. Have fun!
   >>>
   ```

   配置完成后，在fcli可执行文件所在的文件夹下生成*config.yaml*文件。

使用 `fcli config` 命令配置。

1. 在fcli可执行文件所在文件夹下，执行 `fcli config` 。
2. 根据界面提示，依次输入阿里云主账号的账号ID、AccessKey ID和AccessKey Secret，然后选择地域。
   您可以在账号管理中获取账号的Account ID，在用户信息管理中获取AccessKeyID和AccessKeySecret。

   ```
   ? Alibaba Cloud Account ID <your account ID>
   ? Alibaba Cloud Access Key ID <your AccessKey ID>
   ? Alibaba Cloud Access Key Secret <your Accesskey Secret>
   ? Default region name cn-shanghai
   Store the configuration in: .fcli
   ```

   配置完成后，在fcli可执行文件所在的文件夹下生成*config.yaml*文件。

直接配置yaml文件。

1. 打开*~/.fcli/config.yaml*文件。
2. 将对应的参数配置为您自己的值，并保存配置。

```
endpoint: https://<account ID>.<region ID>.fc.aliyuncs.com:8080
api_version: 2016-08-15
access_key_id: <your AccessKeyID>
access_key_secret: <your AccessKeySecret>
security_token: ""
user_agent: fcli-0.1
debug: false
timeout: 60
sls_endpoint: <region ID>.log.aliyuncs.com
```

> **说明** account ID、AccessKeyID、AccessKeySecret的所属账号必须为主账号。您可以在账号管理中获取账号的Account ID，在用户信息管理中获取AccessKeyID和AccessKeySecret。

方法一

方法二

方法三

fcli命令集

函数计算为您提供了fcli的操作命令集以便开发者需要时查阅。开发者也可以通过在可执行文件所在文件夹下执行 `fcli --help` 查看命令的详细信息。

- 通用操作
  - 目录跳转（cd）
  - 列出当前目录下的文件（ls）
  - 查看当前所在目录（pwd）
  - 查看资源的详细信息（info）
  - 配置fcli（config）
  - 删除资源（rm）
  - 沙盒环境（sbox）
  - 帮助（help）
  - 清屏（clear）
  - 退出fcli（exit）
- 服务相关命令
  - 创建服务（mks）
  - 更新服务（ups）
- 函数相关命令
  - 创建函数（mkf）
  - 更新函数（upf）
  - 执行函数（invk）
- 触发器相关命令
  - 创建触发器（mkt）
  - 更新触发器（upt）

2.2. 通用操作

本文介绍fcli的通用操作命令。

前提条件

在可执行文件所在文件夹下执行 `fcli shell` ，进入交互模式。

目录跳转（cd）

cd命令用于切换函数计算的实体的层次关系，而不是切换本地目录。

```
>>> cd serviceName //进入相应serviceName服务的目录。
>>> ls //在service目录下ls，列出当前serviceName服务下所有函数的名称。
>>> cd functionName //进入相应functionName函数的目录。
>>> ls //在function目录下ls，列出当前functionName函数下所有触发器的名称。
```

列出当前目录下的文件（ls）

- `-l int32` 或 `--limit int32` ：指定列出资源的最大数目（默认最大数目为100）。
- `-t string` 或 `--next-token string` ：列出从NextToken开始的资源。
- `-p string` 或 `--prefix string` ：列出指定前缀的资源。
- `-k string` 或 `--start-key string` ：列出从此资源开始的资源。

```
>>> cd myService
>>> ls
aFunction
bFunction
cFunction
cFuncion2
dFunction
eFunction
>>> ls -l 4 //指定列出文件的最大数目为4。
aFunction
bFunction
cFunction
cFuncion2
NextToken: dFunction
>>> ls -t dFunction //列出从NextToken（dFunction）开始的文件。
dFunction
eFunction
>>> ls -p c //列出前缀为c的文件。
cFunction
cFunction2
>>> ls -k dFunction //列出名字从dFunction开始的文件。
dFunction
eFunction
```

查看当前所在目录（pwd）

pwd命令用于查看当前所在目录。

查看资源的详细信息（info）

info命令用于查看函数计算资源的详细信息，其参数可以是服务名称、函数名称以及触发器名称等。

```
>>> info <your serviceName> //查看此服务的详细信息。
>>> info <your functionName> //查看此函数的详细信息。
>>> info <your triggerName> //查看此触发器的详细信息。
```

配置fcli（config）

config命令用于更改*config.yaml*中的配置信息。

```
>>> config --access-key-id 12345678 //修改access-key-id为12345678。
```

删除资源（rm）

- `rm`：删除资源 。

  > **说明** 使用该命令前需要保证目标函数下没有触发器，目标服务下没有函数。

- `-f` 或 `--forced`：删除资源，且删除前无需确认。

```
>>> rm myFunction
Do you want to remove the resource /fc/myService/myFunction [y/n]: //删除资源前需要确认。
>>> rm -f myFunction //删除资源前无需确认，直接删除。
```

沙盒环境（sbox）

fcli为您提供了一个本地的沙盒环境，和函数计算服务中的函数运行环境保持一致。在沙盒环境中，您可以方便地安装第三方依赖库，进行本地调试等操作。

- `-d string` 或 `--code-dir string` ：指定代码所在目录，他将被挂载到沙盒环境的*/code*位置。
- `-t string` 或 `--runtime string` ：指定运行环境。

```
>>> sbox -d code -t nodejs6 //在code目录下安装依赖，语言为Node.js6。
Entering the container. Your code is in the /code direcotry.
root@df9fc****:/code# npm init -f //Node.js6需要生成package.json。
npm info it worked if it ends with ok
npm info using npm@3.10.10
npm info using node@v6.10.3
npm WARN using --force I sure hope you know what you are doing.
Wrote to /code/package.json:
{
"name": "code",
"version": "1.0.0",
"description": "",
"main": "index.js",
"dependencies": {
 "jimp": "^0.2.28"
},
"devDependencies": {},
"scripts": {
 "test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC"
}
npm info init written successfully
npm info ok
root@df9fc****:/code# npm install jimp //安装jimp依赖库。
npm info it worked if it ends with ok
...
npm info lifecycle jimp@0.2.28~postinstall: jimp@0.2.28
code@1.0.0 /code
-- jimp@0.2.28
npm WARN code@1.0.0 No description
npm WARN code@1.0.0 No repository field.
npm info ok
root@df9fc****:/code# exit //退出sbox环境。
```

帮助（help）

help命令用于列出帮助信息。

- `help` ：可以列出所有的命令。
- `[command] --help` ：可以列出此操作下的所有参数的介绍。
  例如 `ls --help` 。

清屏（clear）

clear命令用于清屏。

退出fcli（exit）
exit 命令用于退出fcli工具。

2.3. 服务相关命令
本文介绍fcli中服务相关的命令。

前提条件
在可执行文件所在文件夹下执行 `fcli shell` ，进入交互模式。

创建服务（mks）
- `-d string` 或 `--description string` ：服务中的描述信息。
- `-p string` 或 `--log-project string` ：指定服务对应的日志项目Project。
- `-l string` 或 `--log-store string` ：指定服务对应的日志库Logstore。
- `-r string` 或 `--role string` ：指定服务对应的角色。

```
>>> mks myService //新建一个服务，不带任何高级配置内容。
>>> mks myService -d 'my description' -p my-log-project -l my-log-store -r acs:ram::myID:role/myRoleName //新建服务，其中描述信息为my description，日志项目为my-log-project，日志存储my-log-store，角色为myRoleName。
```

更新服务（ups）
该命令的参数与mks命令的参数相同。
- `-d string` 或 `--description string` ：服务中的描述信息。
- `-p string` 或 `--log-project string` ：指定服务对应的日志项目Project。
- `-l string` 或 `--log-store string` ：指定服务对应的日志库Logstore。
- `-r string` 或 `--role string` ：指定服务对应的角色。
-

```
>>> ups myService -d 'my description' -p my-log-project -l my-log-store -r acs:ram::myID:role/myRoleName //更新服务，其中描述信息为my description，日志项目为my-log-project，日志存储my-log-store，角色为myRoleName。
```

更多信息
- 列出此账户下的所有服务
- 查看服务具体信息
- 删除服务

2.4. 函数相关命令
本文介绍fcli中函数相关的命令。

前提条件
在可执行文件所在文件夹下执行 `fcli shell` ，进入交互模式。

创建函数（mkf）
- `-b string` 或 `--code-bucket string` ：指定代码所在的OSS Bucket。
- `-o string` 或 `--code-object string` ：指定代码所在的Bucket中的Object Key。

- -d string 或 --code-dir string ：指定代码所在的目录。
- -f string 或 --code-file string ：指定压缩的代码文件。
- -h string 或 --handler string ：设置函数handler，handler的格式为“文件名.函数名”。例如hello_world.handler指定了函数的调用入口为*hello_world.js*文件中的handler函数。
- -e int32 或 --initializationTimeout int32 ：设置初始化函数超时时间（默认30s）。
- -i string 或 --initializer string ：设置初始化函数。
- -m int32 或 --memory int32 ：设置函数执行的内存大小。
- -t string 或 --runtime string ：指定运行环境。
- --timeout int32 ：设置函数超时时间（默认30s）。

```
// 在相应service目录下
>>> mkf myFunction -t nodejs6 -h myFunction.handler -b ossBucketName -o objectKey //代码存储在OSS上，-t指定Runtime为Node.js6，-h指定函数入口，-b指定代码所在的OSS Bucket，-o指定了代码在Bucket中的Object Key。
>>> mkf myFunction -t nodejs6 -h myFunction.handler -d codeDir/myFunction -m 512 //代码存储在本地，-d指定了代码所在目录，-m设置函数执行的内存大小。
>>> mkf myFunction -h myFunction.handler -f code.zip -t nodejs6 //代码在本地的code.zip中
```

更新函数（upf）

- -b string 或 --code-bucket string ：指定代码所在的OSS Bucket。
- -o string 或 --code-object string ：指定代码所在的Bucket中的Object Key。
- -d string 或 --code-dir string ：指定代码所在的目录。
- -f string 或 --code-file string ：指定压缩的代码文件。
- -h string 或 --handler string ：设置函数handler，handler的格式为“文件名.函数名”。例如hello_world.handler指定了函数的调用入口为*hello_world.js*文件中的handler函数。
- -m int32 或 --memory int32 ：设置函数执行的内存大小。
- -t string 或 --runtime string ：指定运行环境。

```
// 在相应service目录下
>>> upf myFunction -t nodejs6 -h myFunction.handler -b ossBucketName -o objectKey //代码存储在OSS上，-t指定Runtime为Node.js6，-b指定代码所在的OSS Bucket，-o指定了代码在Bucket中的Object Key。
>>> upf myFunction -t nodejs6 -h myFunction.handler -d codeDir/myFunction -m 512 //代码存储在本地，-d指定了代码所在目录，-m设置函数执行的内存大小。
>>> upf myFunction -h myFunction.handler -f code.zip -t nodejs6 //代码在本地的code.zip中。
>>> upf myFunction -t nodejs6 -i myFunction.newInitializer -e 30  -b ossBucketName -o objectKey // 将initializer从myFunction.initializer更新至myFunction.newInitializer。代码存储在OSS上且包含initializer函数，-t指定Runtime，-i指定initializer入口，-e指定initializer超时时间，-b指定代码所在的OSS Bucket，-o指定了代码在Bucket中的Object Key。
>>> upf myFunction -t nodejs6  -i "" -b ossBucketName -o objectKey // 将initializer从myFunction.newInitializer更新至空，即关闭函数initializer功能。
>>> upf myFunction -t nodejs6 -i  myFunction.newInitializer -e 30 -b ossBucketName -o objectKey // 将initializer从空更新至myFunction.newInitializer。
```

执行函数（invk）

- -e 或 --encode ：对函数的返回值进行base64编码。
- -f string 或 --event-file string ：从文件中读取触发事件内容。
- -s string 或 --event-str string ：从字符串中读取触发事件内容。
- -o string 或 --output-file string ：将返回结果写入文件的文件名称。

- `-t string` 或 `--type string` ：触发类型，取值：
  - Sync：同步触发（默认值）
  - Async：异步触发

```
>>> invk myFunction //如果不需要输入参数，不需要触发事件的话，则直接调用。
>>> invk myFunction -e //对函数的返回值进行base64编码。
>>> invk myFunction -s 'hello,world'//从字符串中读取触发事件内容。
>>> invk myFunction -f event.json //从文件中读取触发事件内容。
>>> invk myFunction -o code/result.txt //将返回结果写入result.txt文件中。
>>> invk myFunction -t Async //设置触发类型为异步触发。
```

更多信息

- 列出指定服务下的所有函数
- 查看函数具体信息
- 删除函数

2.5. 触发器相关命令

本文介绍fcli中触发器相关的命令。

前提条件

在可执行文件所在文件夹下执行 `fcli shell` ，进入交互模式。

创建触发器（mkt）

- `-r string` 或 `--invocation-role string` ：设置触发角色。
- `-s string` 或 `--source-arn string` ：事件源的资源符号，例如acs:oss:cn-shanghai:12345678:myBucketName。
- `-c string` 或 `--trigger-config string` ：设置触发器配置文件。
- `-t string` 或 `--type string` ：触发器类型，默认值为oss。
- 创建OSS触发器

```
>>> mkt myFunction/myFunctionTrigger -t oss -r acs:ram::12345678:role/AliyunOSSEventNotificationRole -s acs:oss:cn-shanghai:12345678:myOssBucket -c code/ossTrigger.yaml
//其中yaml的文件内容如下triggerConfig。
  events:
    - oss:ObjectCreated:PutObject
    - oss:ObjectRemoved:DeleteObject
  filter:
    key:
      prefix: myPrefix
      suffix: mySuffix
```

- 创建HTTP触发器

```
>>> mkt myFunction/myFunctionTrigger -t http  -c code/httpTrigger.yaml
//其中yaml的文件内容如下triggerConfig:
 authType: anonymous
 methods:
  - GET
```

命令

示例

更新触发器（upt）

- `-r string` 或 `--invocation-role string`：设置触发角色。
- `-s string` 或 `--source-arn string`：事件源的资源符号，例如acs:oss:cn-shanghai:12345678:myBucketName。
- `-c string` 或 `--trigger-config string`：设置触发器配置文件。
- `-t string` 或 `--type string`：触发器类型，默认值为oss。

```
>>> upt myFunction/myFunctionTrigger -t oss -r acs:ram::account_id:role/AliyunOSSEventNotificationRole -s acs:oss:cn-region:account_id:bucketName -c code/trigger.yaml
```

更多信息

- 列出当前目录下的文件（ls）
- 查看资源的详细信息（info）
- 删除资源（rm）

2.6. 日志相关命令

本文介绍fcli中的日志相关命令。

前提条件

在可执行文件所在文件夹下执行 `fcli shell`，进入交互模式。

创建日志项目和日志库（mkl）

mkl命令用于创建函数服务对应的日志项目Project和日志库Logstore。

- `-p string`：创建日志项目（Log Project）。
- `-s string`：创建日志库（Log Store）。

```
>>> mkl -p my-log-project -s my-log-store myService
// 为服务新建日志项目和日志库。
// Log Project的名称全局唯一，如果名称已被占用，那么会创建失败。
```

查看日志（logs）

logs命令用于查看日志相关信息。

- `-c int` 或 `--count int`：设置返回日志数目的最大行数（默认是1000行）。
- `-d int` 或 `--duration int`：返回从这段时间之前一直到现在的函数日志，单位秒，默认86400秒（24小时）。
- `-e string` 或 `--end string`：设置查看日志的截止时间，格式为UTC RFC3339，例如2017-01-01T01:02:03Z。
- `-s string` 或 `--start string`：设置查看日志的起始时间，格式为UTC RFC3339，例如2017-01-01T01:02:03Z
- `-t` 或 `--tail`：设置从倒数第i行开始打印日志。

```
// 在相应service目录下。
logs myFunction //默认打印一天内的前1000行日志。
logs -d 60 -c 5000 myFunction //打印一分钟内执行的日志，最多打印5000条。
logs -t -c 100 myFunction // 打印倒数100行日志。
logs -s 2018-01-22T18:00:00Z -e 2018-01-22T19:00:00Z myFunction //打印从2018-01-22T18:00:00Z到2018-01-22T19:00:00Z的函数日志信息。
```

2.7. 角色授权相关命令

本文介绍fcli中的角色授权相关命令。

前提条件

在可执行文件所在文件夹下执行 `fcli shell` ，进入交互模式。

创建RAM权限策略（mkrp）

mkrp命令用于创建RAM权限策略。

- `-a string` 或 `--action string` ： 设置策略的操作名称。
- `-r string` 或 `--resource string` ： 设置策略的操作对象。

> **说明** 关于权限策略的详细内容，请参见Policy语法结构。

创建角色（mksr）

mksr命令用于创建角色，函数计算可以使用此角色访问云资源。

```
mksr roleName
```

赋予角色RAM策略（attach）

attach命令用于将RAM策略赋予指定角色。

- `-p string` 或 `--policy string` ： 指定RAM策略。
- `-r string` 或 `--role string` ： 指定RAM角色。

```
attach -p /ram/policies/myPolicy -r /ram/roles/myRole //将myPolicy策略赋予myRole角色。
```

为角色解除权限策略（detach）

detach命令用于为指定角色解除指定策略。

- `-a string` 或 `--action string` ： 设置策略的操作名称。
- `-r string` 或 `--resource string` ： 设置策略的操作对象。
-

```
detach -p /ram/policies/myPolicy -r /ram/roles/myRole //为myRole角色解除myPolicy授权。
```

为函数计算服务授权（grant）

grant 命令用于为函数计算服务授予指定权限。

```
grant myService
Please input the role name: myRole
Please input the policy name: myPolicy
Permission grant scenarios:
 1. Allow FC write function logs to your log store.
 2. Allow FC copy code from your OSS location.
Please input your choice [1-2]: 1
Please input the log project: my-log-project
Please input the log store: my-log-store
```

3.Serverless Devs

通过Serverless Devs您可以像使用手机一样使用Serverless。

中文文档 | English

> **说明**
> 官方网址：https://www.serverless-devs.com
> 文档地址：https://www.serverless-devs.com/docs/

```
_________                .__              ________
/  _____/ ______________ __ ____________| |  ____  ______ ______\_____ \  _______ ________
\_____ \_/ __ \_  __ \ \// __ \_  __ \ |_/ __ \/ ___// ___/ |  | \_/ __ \ \/ /  ___/
/        \  ___/|  | \/\  /\  ___/|  | \/ |_\  ___/\___ \\___ \  |  `   \  ___/\  /\___ \
/_______  /\___  >__|    \_/  \___  >__|  |____/\___  >____  >____  >/_______  /\___  >\_//____  >
        \/     \/                 \/                 \/     \/     \/         \/     \/        \/
```

- 项目介绍
- 快速入门
- 命令行工具
  - 入门文档
    - 快速入门
    - 安装文档
    - 账号配置
    - Yaml配置
  - 指令文档
    - Config指令
    - Init指令
    - Search指令
    - Set指令
    - Platform指令
    - GUI指令
    - 泛指令
- 应用中心

    - 入门相关
        - 快速入门
        - 应用中心简介
        - Package汇总
    - 应用中心版 awesome
        - 组件列表
        - 应用列表
        - 插件列表
- 其他文档
    - Package概念
    - Package开发指南
    - Package开发文档
        - Component开发
        - Application开发
        - Plugin开发
    - 各云厂商密钥（凭证）获取方法
        - 阿里云
        - 百度云
        - AWS
        - Azure
        - Google Cloud
        - 华为云
        - 腾讯云
- 常见问答：
    - 常见问答

3.1. 什么是Serverless Devs Tool

Serverless Devs Tool是一款让Serverless开发者开发及提高运维效率的工具。通过该工具，您可以更简单、更快速的进行项目的开发、创建、测试及部署，实现项目全生命周期的管理。本文介绍Serverless Devs Tool的基本功能。

应用场景

Serverless Devs Tool是一个包含云产品、资源、全链路及生命周期管理平台的工具。该工具在组件化和插件化的共同作用下，参与项目的创建、开发、调试、部署与运维的全过程。本文以阿里云函数计算组件为例：

1. 项目的创建：通过命令行工具或应用中心进行项目的最初创建。
2. 项目的开发：通过本地调试等能力，来验证本地开发的正确性等。
3. 项目的调试：通过本地调试与远程调用、日志查询等能力，来进行项目的最终调试。
4. 项目的部署：先通过项目部署、依赖安装、项目构建等流程构建出完整的部署包，再进行项目的部署。
5. 项目的运维：
   - 通过查询指标检查项目的健康度。
   - 通过查询日志定位问题。
   - 通过项目发布能力发布版本、别名及灰度等。

基本功能

Serverless Devs App Store的应用中心拥有大量的项目模板、案例模板，您可以使用相关指令通过这些模板将项目一键部署到指定的云平台上，具体信息，请参见Serverless Devs Tool。

产品优势

- 支持Serverless服务和框架
  Serverless Devs App Store是一个组件化与插件化的Serverless开发者平台，在该平台中，您可以进行可插拔式的使用不同Serverless服务和框架，同时您也可以参与开发组件和插件。在Serverless Devs Tool中无论是工业级的Serverless服务，还是各类开源的Serverless框架，都可以得到支持。您无需对每一款Serverless工具进行研究和学习，只需通过Serverless Devs Tool，就可以简单、快捷的使用主流Serverless服务和框架。
- 支持可视化编辑和部署
  Serverless Devs Tool拥有完善的可视化编辑和部署流程。在Serverless Devs App Store中，您可以通过关键词快速检索到自己所需的应用案例或组件，并且可以通过可视化编辑完成项目配置，然后进行部署。

在应用中心及可视化编辑和部署的情况下，Serverless项目的整体部署时间大约降低了50%。同时Serverless Devs App Store也是一个开发者开源共建的平台，您可以在Serverless Devs App Store发布您自己的组件和应用供更多人学习、参考以及使用。

- 灵活与开放的使用方法
  Serverless Devs Tool在进行项目描述时不仅可以对函数计算、API网关、对象存储等资源进行描述，而且可以通过Serverless Devs Tool提供的插件以及钩子进行安装、创建及推送等行为描述。
  Serverless Devs Tool不会对组件的命令进行限制，而是鼓励开发者针对不同的组件，开发不同的能力来应对更多、更复杂的场景，例如阿里云函数计算组件，不仅支持函数的部署、删除等基础能力，还支持日志查询、指标查询、本地构建、依赖安装及调试等更多定制化的能力。

安装方式

安装方式

3.2. 安装方式

Serverless Devs Tool是一个让Serverless开发者开发和提高运维效率的工具。通过该工具，您可以更简单、更快速的进行项目的开发、创建、测试及部署，实现项目的全生命周期管理。本文介绍安装Serverless Devs Tool的两种方式。

macOS或Linux快速安装

如果您是macOS或Linux用户，您可以执行以下命令安装：

```
curl -o- -L http://cli.so/install.sh | bash
```

npm安装

通过npm包管理安装：适用于已经成功安装npm的Windows、macOS、Linux平台。

> **注意**
> 您需要下载及安装Node.js10及以上版本，详细信息，请参见Node.js。

1. 在Windows、macOS、Linux平台执行以下命令安装Serverless Devs Tool。

   ```
   npm install @serverless-devs/s -g
   ```

   > **说明**
   > - 如果在Linux或macOS下执行该命令报错且报错信息为 `Error: EACCES: permission denied` ，请执行命令 `sudo npm install @serverless-devs/s -g` 。
   > - 如果安装过程较慢，您可以使用淘宝npm源，安装命令为 `npm --registry=https://registry.npm.taobao.org install @serverless-devs/s -g` 。

2. 安装完成之后，在控制终端执行以下命令查看版本信息。

   ```
   s -v
   ```

   预期输出：

```
Serverless Tool Version:
  Release: 1.1.2
  Local  : 1.1.0

  You can upgrade through : npm install @serverless-devs/s -g
  Update information:
    1. Add the .signore file of platform publish
    2. Fix the component template problem in platform init.
    3. Fix exit code problem during execution (normal exit 0, abnormal exit 1)
```

4.Aliyun Serverless VSCode Extension插件

Aliyun Serverless VSCode Extension是一款VSCode图形化开发调试函数计算以及操作函数计算资源的插件。本文介绍了如何通过该插件创建函数以及该插件的常见功能。

前提条件

如果您期望使用Aliyun Serverless VSCode Extension的所有功能，那么您需要确保系统中有以下组件：

- VSCode：可以在Visual Studio Code官网中下载安装。
- Docker：可以在aliyun/fun中根据教程安装配置Docker。

背景信息

Aliyun Serverless VSCode Extension是函数计算提供的VSCode插件，该插件结合了函数计算命令行工具Fun和函数计算SDK的功能，是基于VSCode的开发、调试、部署工具。通过该插件，您可以：

- 快速地在本地初始化项目、创建函数。
- 运行、调试本地函数，以及部署服务函数至云端。
- 拉取云端的服务函数列表、查看服务函数配置信息、调用云端函数。
- 获得模版文件的语法提示：自动补全、Schema校验、悬浮提示。

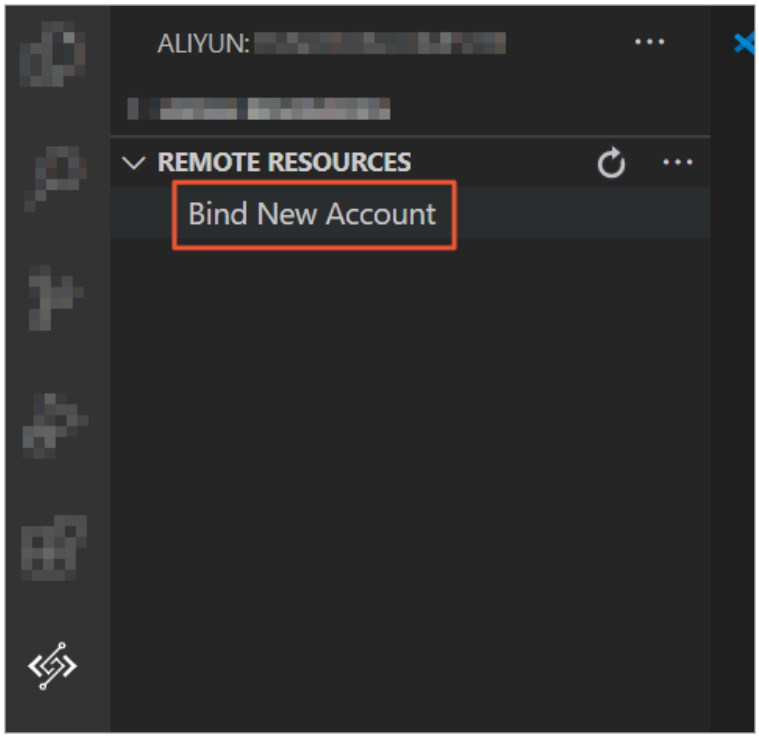
安装插件

1. 打开VSCode并进入插件市场。
2. 在插件市场中搜索Aliyun Serverless，查看详情并安装。
3. 重启VSCode，左侧导航栏中会展示已安装的Aliyun Serverless VSCode Extension插件图标。

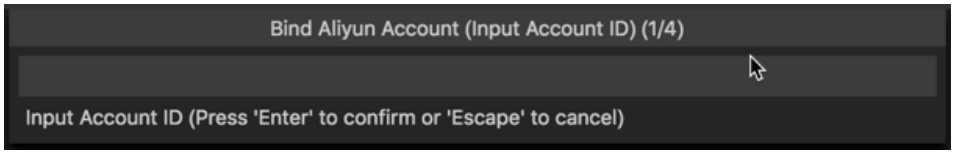
快速入门

1. 绑定阿里云账户。

i. 在左侧导航栏，单击Aliyun Serverless VSCode Extension图标，然后单击**Bind New Account**。

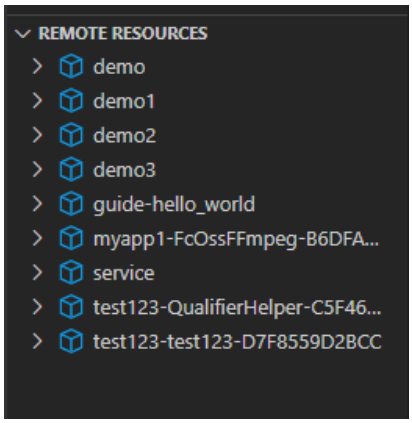


ii. 依次输入阿里云Account ID、阿里云AccessKey ID、阿里云AccessKey Secret、自定义的账户别名（即账户本地名称）。
您可以在账号管理中获取账号的Account ID，用户信息管理中获取AccessKey ID和AccessKey Secret。

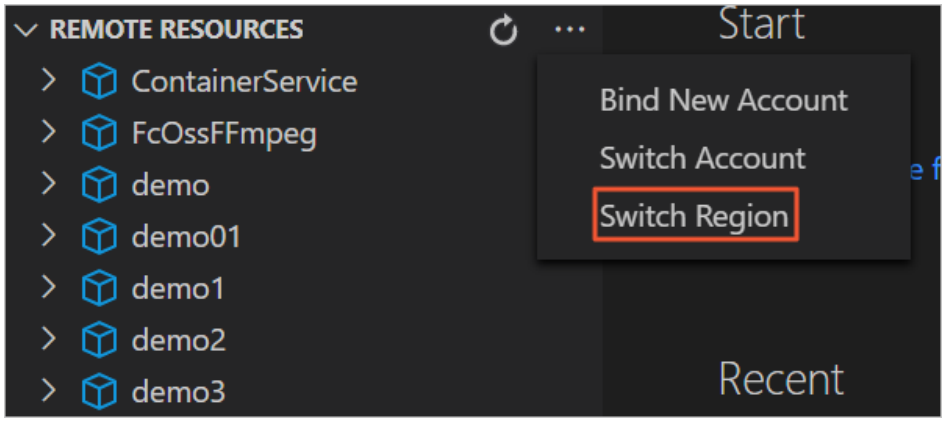


> **说明** VSCode支持子账号登录。您在使用子账号登录VSCode时，阿里云Account ID需要使用您的主账号的信息，阿里云AccessKey ID及阿里云AccessKey Secret需要使用您的子账号信息。

绑定完成后，可以看到所绑定的阿里云账户的云端服务与函数列表。

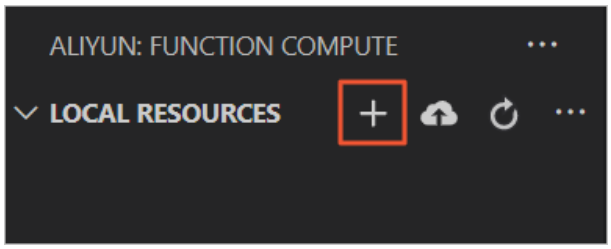


您还可以在**REMOTE RESOURCES**面板中，单击右上角的更多信息图标，在下拉菜单中，选择**Switch Region**来查看不同地域的服务与函数。

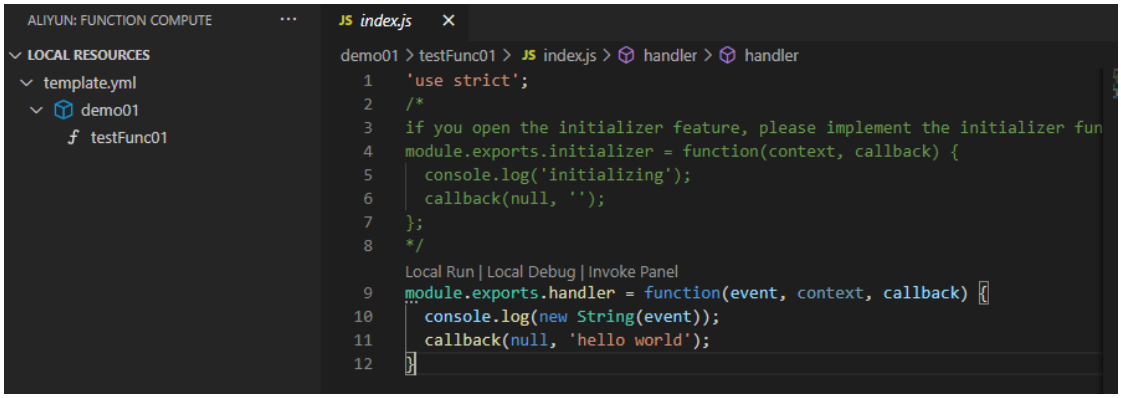


2. 创建函数。

i. 通过VSCode，打开一个空的目录文件。单击LOCAL RESOURCES中的创建函数图标，可以在本地初始化一个函数计算项目。



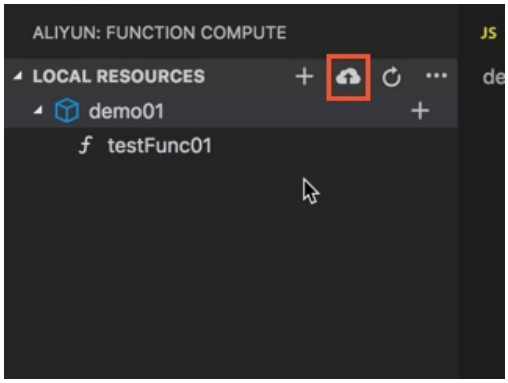
ii. 按照导航依次输入或选择服务名称、函数名称、函数运行环境、函数类型。填写完毕后，插件会自动创建函数并在LOCAL RESOURCES面板中会展示新建的本地服务与函数。



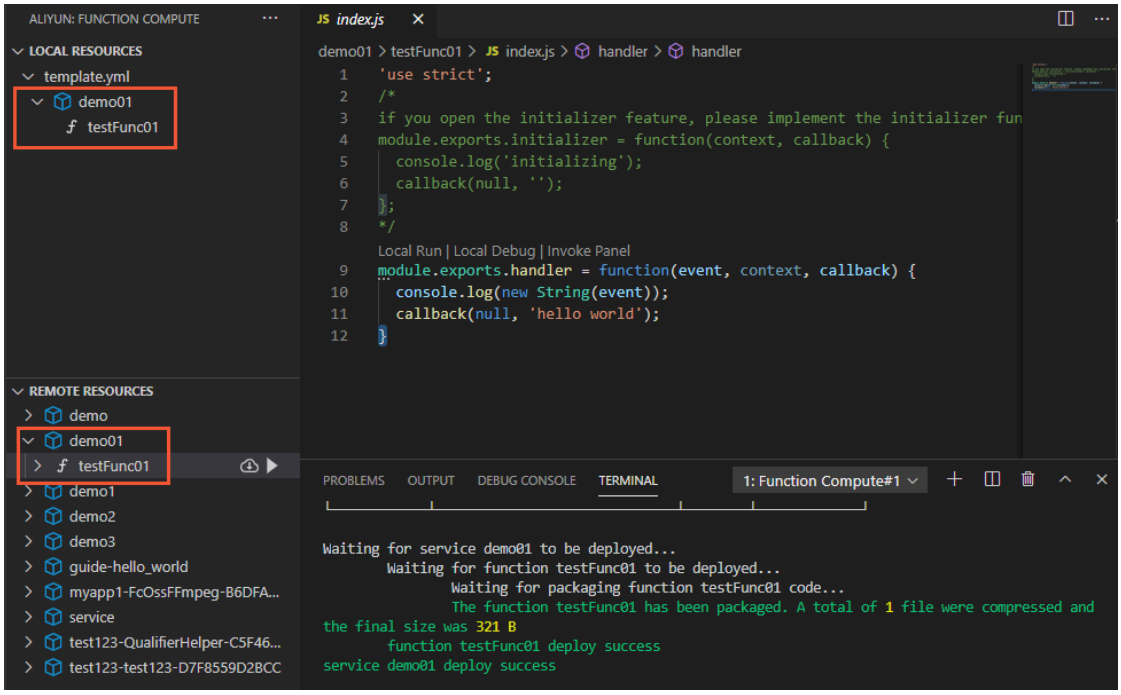
您也可以直接单击LOCAL RESOURCES中服务名右侧的创建函数图标，来为该服务创建函数。按照导航依次输入或选择函数名称、函数运行时、函数类型即可。

3. 部署服务以及函数。

i. 单击LOCAL RESOURCES面板中的部署图标，可以将本地的服务与函数部署到云端。

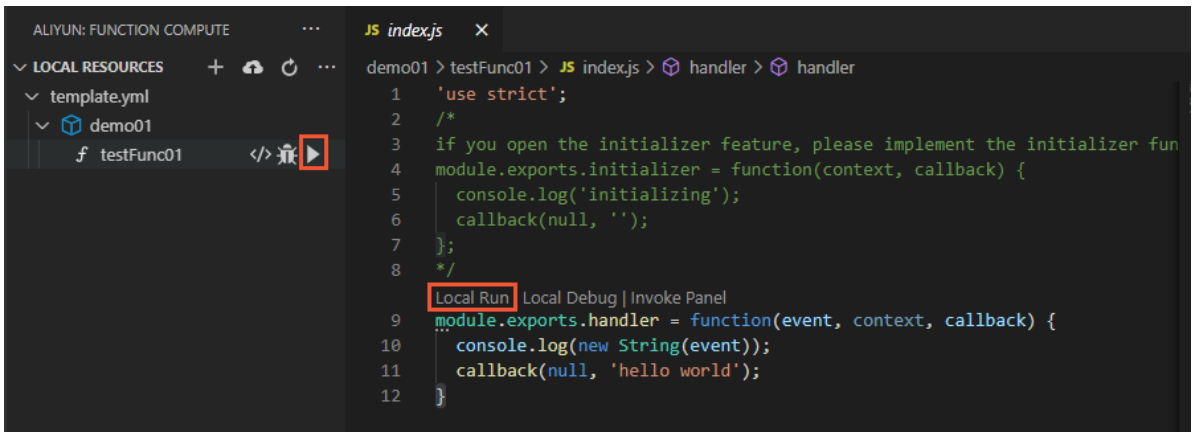


部署完成后，单击REMOTE RESOURCES面板中的刷新图标，可以查看部署到云端的服务与函数。

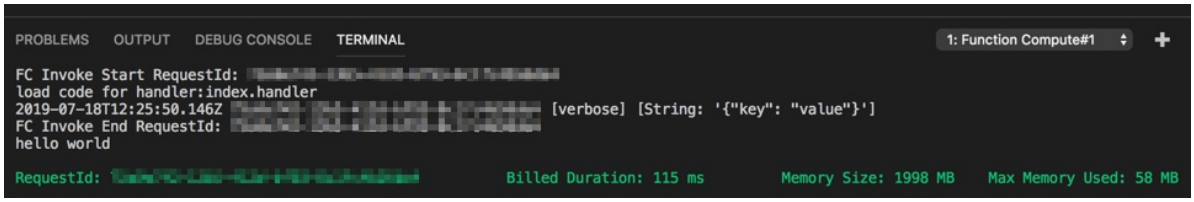


其余功能介绍

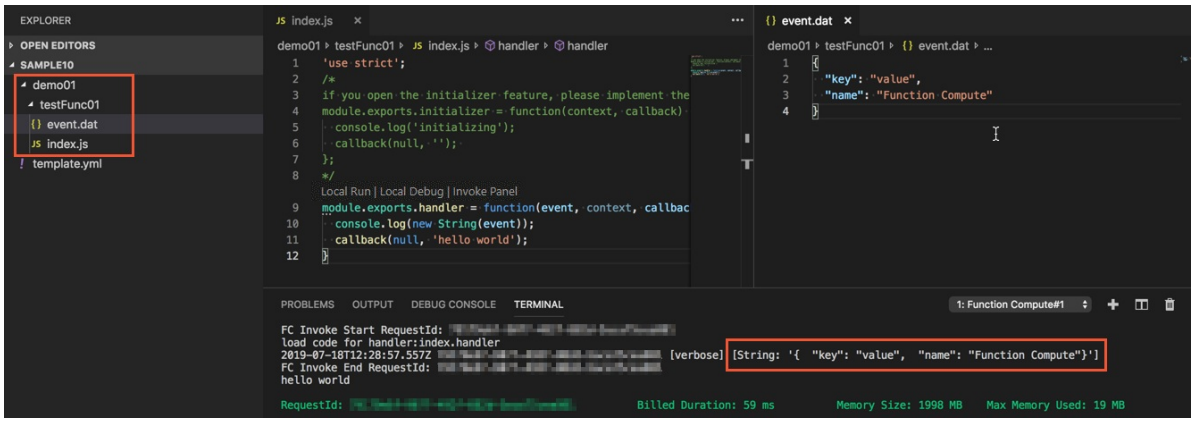
- 本地调用函数
  在LOCAL RESOURCES面板中，单击函数名称右侧的执行图标或*Handler*文件中的执行链接，可以在本地调用该函数。



函数的日志以及结果会输出在TERMINAL中。



插件会为您在函数入口文件同目录下创建*event.dat*文件，您可以通过修改该文件设置每次调用函数时触发的事件信息。

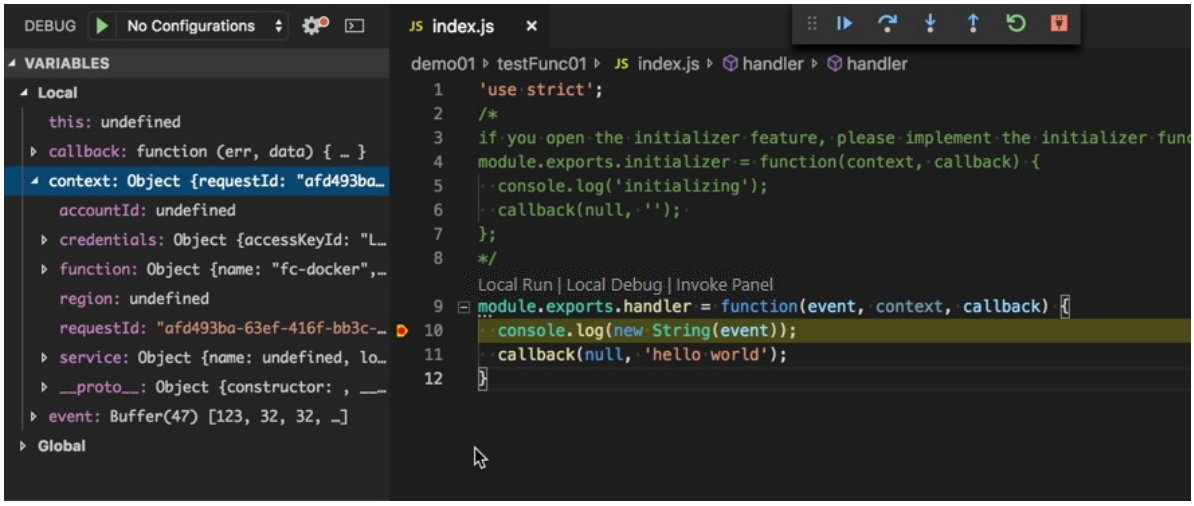


- 本地调试函数

> **注意**
> - 若您想要调试Python 2.7或Python 3 runtime的函数，需要事先在插件安装Python插件。
> - 若您想调试PHP runtime的函数，需要事先在插件安装PHP Debug插件。

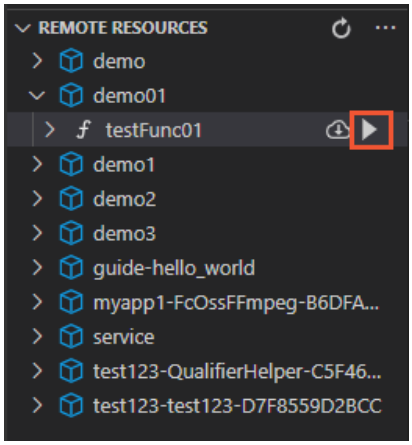
在LOCAL RESOURCES面板中，单击函数名称右侧的调试图标或*Handler*文件中的调试链接，可以在本地调试该函数。

在代码文件中插入断点，启动调试后即可看到调试信息。

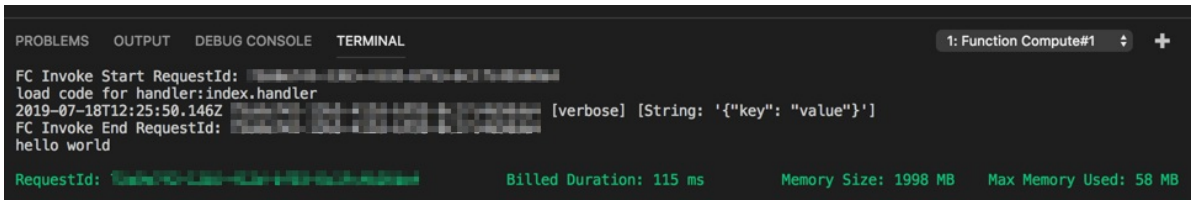


插件会为您在函数入口文件同目录下创建*event.dat*文件，您可以通过修改该文件设置每次调试函数时触发的事件信息。

- 执行云端函数
  单击**REMOTE RESOURCES**面板中函数右侧的执行图标，可以执行云端函数。

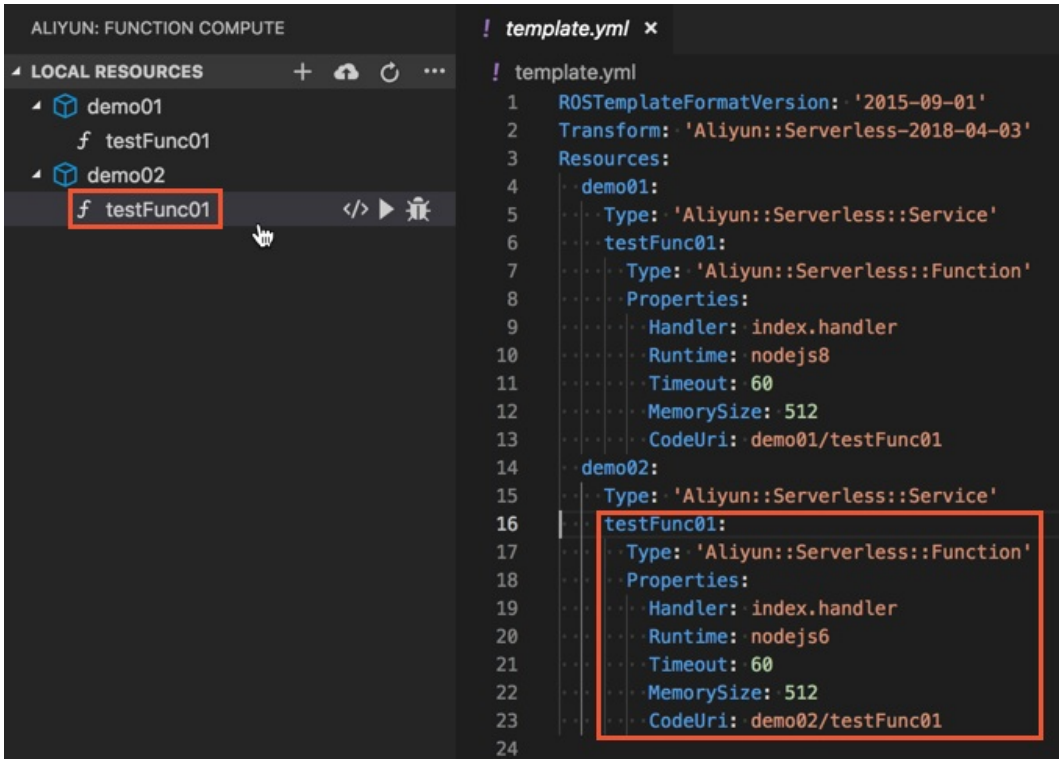


函数的日志以及结果会输出在**TERMINAL**中。



插件会为您在项目根目录下创建*event.dat*文件，您可以通过修改该文件设置每次调用云端函数时触发的事件信息。

- 跳转到模版文件定义
  函数计算Fun工具通过YAML格式的模板文件来描述Serverless应用。通过Aliyun Serverless VSCode Extension创建函数时，会使用默认值自动填充模版文件。若您想修改本地服务或函数的配置，可以通过单击**LOCAL RESOURCES**面板中的服务或函数名，跳转到模版文件中的相关描述，所选择资源在模板文件中的相关描述块会高亮并逐渐褪去。



- 模版文件提示
  - 自动补全
    支持模版文件*template.yml*内所有资源配置属性的自动补全。自动补全会依据缩进层级给出精准的提示选项。
  - Schema校验
    支持模版文件*template.yml*内所有资源配置信息的校验。在*template.yml*中会检测资源的配置信息是否符合规格说明。
  - 悬浮提示
    提供模版文件*template.yml*内所有资源配置的上下文帮助。在*template.yml*中，将鼠标悬浮在相关资源的键名上，会出现关于该键下可配置字段的悬浮信息展示（字段名、字段类型）。

反馈

如果您在使用中遇到问题，欢迎扫描以下二维码加入函数计算官方客户群或在github中反馈。

5.其他

如果您需要使用阿里云的函数计算服务，您不仅可以通过函数计算控制台实现，也可以通过其他工具例如IntellIj IDEA、Eclipse及PyCharm工具实现。通过其他工具可以运行、下载云端函数，创建、上传本地函数。本文以IntellIj IDEA工具为例介绍如何管理函数。

前提条件

- 已安装Funcraft工具，请参见Funcraft。
- 已开通函数计算，请参见函数计算。

背景信息

函数计算是一个事件驱动的全托管Serverless计算服务，能帮您快速调动和使用大量计算资源，提供弹性可靠的运行环境。使用函数计算，您无需采购与管理服务器等基础设施，只需编写代码并上传即可在云端运行任务。

安装Funcraft工具可以实现在其他工具中创建、运行、调试和部署函数，还可以运行、下载云端的函数。

安装Cloud Toolkit插件是您使用不同的工具例如IntellIj IDEA、Eclipse及PyCharm管理函数的必要前提，Cloud Toolkit插件与函数计算可以实现数据共享，安装并配置Cloud Toolkit插件后可以在工具中查看该账号下的函数信息。

操作步骤

1. 安装和配置Cloud Toolkit插件
   以下分别介绍在三种工具中安装和配置Cloud Toolkit插件的方法。

在IntelliJ IDEA中安装和配置Cloud Toolkit

i. 下载并安装以下工具：

- JDK 1.8及以上版本
- IntellIj IDEA（2018.02及以上版本）

ii. 安装及配置Cloud Toolkit插件。
在插件市场中下载安装。

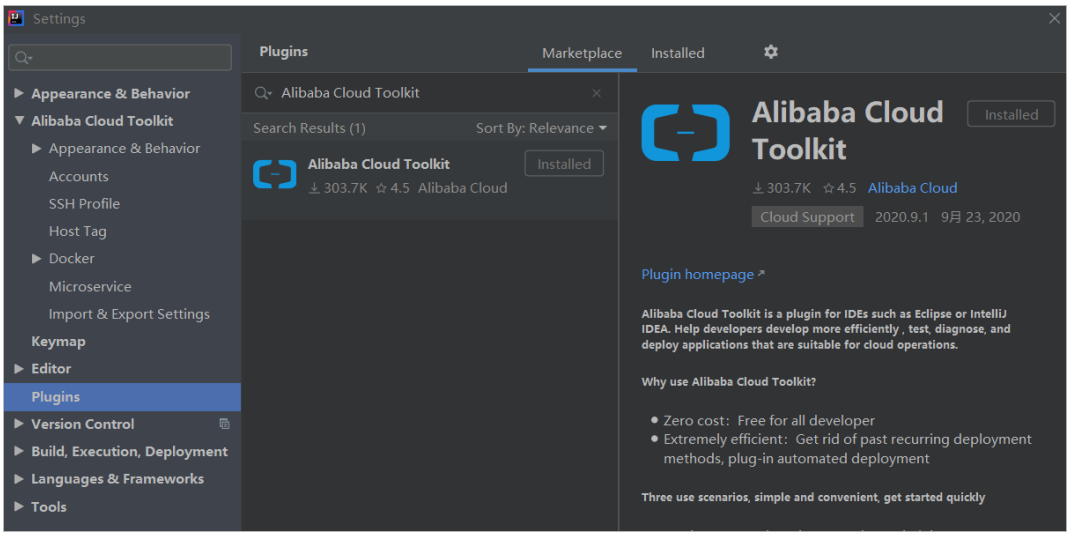
a. 在IntelliJ IDEA顶部菜单栏中选择**File > Settings**。

b. 在**Settings**对话框的左侧导航栏中单击**Plugins**。

c. 在**Plugins**区域单击**Marketplace**。

d. 在搜索栏中输入*Alibaba Cloud Toolkit*。

e. **Search Results**区域会出现Alibaba Cloud Toolkit，单击**Installed**。



f. 等待下载、安装完成后，单击**Restart IDE**。

在Eclipse中安装和配置Cloud Toolkit

i. 下载并安装以下工具：

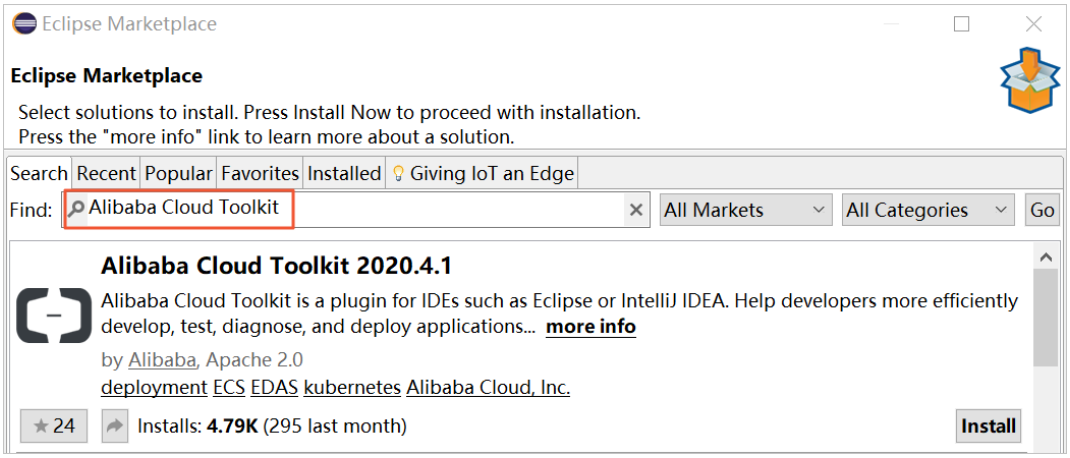
- JDK 1.8及以上版本
- Eclipse IDE 4.5.0（代号：Mars）及以上版本

> **说明** 若需使用内置Terminal功能，确保您的TM Terminal为v4.1.0及以上版本。

ii. 安装及配置Cloud Toolkit插件。
在Eclipse Marketplace中下载安装。

a. 在Eclipse顶部菜单栏中选择**Help > Eclipse Marketplace**。

b. 在**Eclipse Marketplace**对话框中**Find**右侧的文本框中输入*Alibaba Cloud Toolkit*，然后回车。



c. 在搜索结果中单击**Alibaba Cloud Toolkit**区域右下角的**Install**。

d. 根据Eclipse安装页面的提示，完成后续安装步骤。

在PyCharm中安装和配置Cloud Toolkit

i. 安装PyCharm（2018.01及以上版本）。

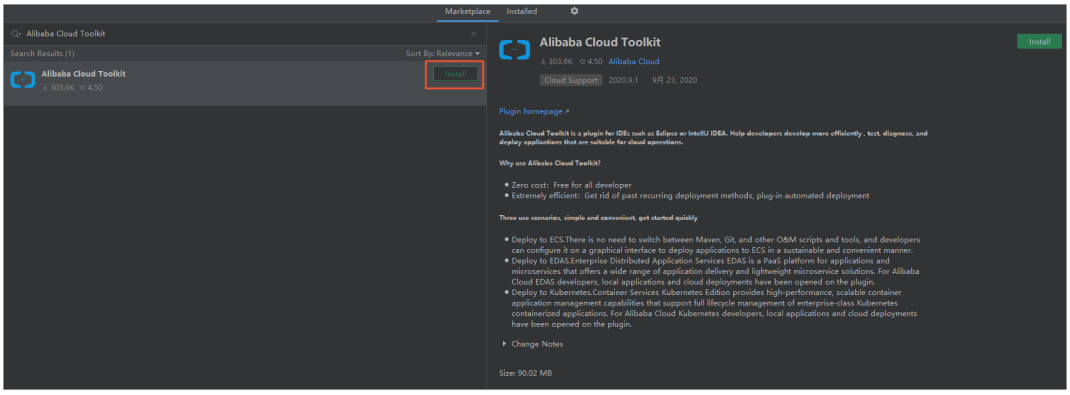
ii. 安装及配置Cloud Toolkit插件。
在PyCharm Marketplace中下载安装

a. 打开PyCharm工具。

b. 在左侧导航栏单击**Plugins**。

c. 在**Marketplace**页签的搜索栏中输入*Alibaba Cloud Toolkit*。

d. 在搜索区域中会出现Alibaba Cloud Toolkit，单击**Install**。



e. 在**Third-party Plugins Privacy Note**对话框中单击**Accept**。

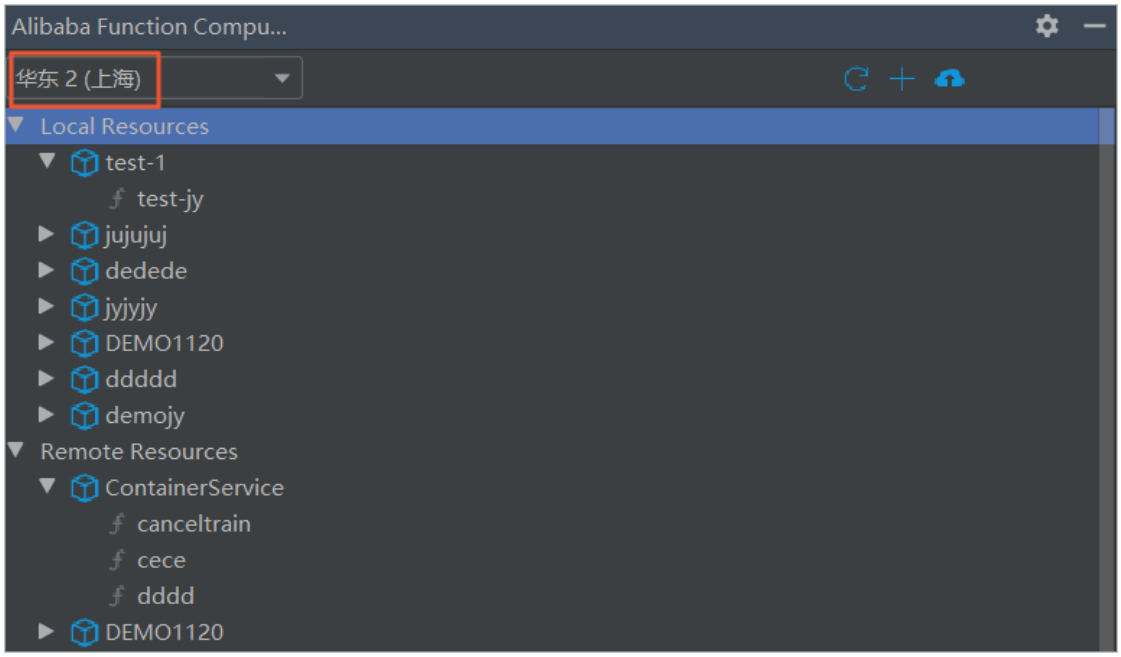
f. 根据PyCharm安装页面的提示，完成后续安装步骤。

2. 管理函数
本文以IntellIj IDEA工具为例介绍如何管理函数。

i. 打开IntellIj IDEA工具。

ii. 在右侧导航栏中单击**Alibaba Function Compute**。

iii. 在Alibaba Function Compute面板中选择地域，然后按需对本地资源或云端资源进行操作。



- 展开Local Resources进行本地资源操作：
  - 创建资源：在Alibaba Function Compute面板的右上角单击加号图标，在Create Function对话框中配置相关信息，然后单击Add。

    > **注意** 如果您初次使用IDEA，您需要先创建本地资源，再在本地进行运行、调试函数等操作。

  - 运行本地函数：右键单击目标函数，单击Local Run。
  - 调试本地函数：右键单击目标函数，单击Local Debug。
  - 查看函数：右键单击目标函数，单击Go To Code。

    > **说明** Java语言的函数不支持此功能。

  - 部署服务及服务内的所有函数：右键单击目标服务，单击Deploy Service。
  - 部署单个函数：右键单击目标函数，单击Deploy Function。
- 展开Remote Resources进行云端资源操作：
  - 运行远端函数：右键单击目标函数，单击Remote Run or Copy URL。
  - 下载服务及服务内的所有函数：右键单击目标服务，单击Import To Local。
  - 下载单个函数：右键单击目标函数，单击Import To Local。
  - 查看服务性能：右键单击目标服务，单击Properties。
  - 查看函数性能：右键单击目标函数，单击Properties。

> **说明** 执行下载、运行和部署等操作时，IntelliJ IDEA的Console区域会打印操作日志，您可以根据日志信息检查部署结果。