函数工作流

最佳实践

文档版本	01
发布日期	2024-03-26

华为云计算技术有限公司

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 使用函数压缩图片 1
1.1 案例概述 1
1.2 准备 2
1.3 构建程序 3
1.4 添加事件源 5
1.5 图片处理 6
2 使用函数为图片打水印 7
2.1 案例概述 7
2.2 准备 7
2.3 构建程序 9
2.4 添加事件源 10
2.5 处理图片 11
3 使用函数处理 DIS 数据 13
3.1 案例概述 13
3.2 准备 13
3.3 构建程序 15
3.4 添加事件源 21
3.5 处理数据 22
4 函数+LTS：日志实时分析实战 24
4.1 案例概述 24
4.2 准备 25
4.3 构建程序 26
4.4 添加事件源 27
4.5 处理结果 28
4.6 应用扩展 28
5 函数+CTS：登录/登出安全分析实战 30
5.1 案例概述 30
5.2 准备 31
5.3 构建程序 32
5.4 添加事件源 32
5.5 处理结果 33

6 定时开关华为公有云虚拟机.....35
7 使用 SpringBoot 构建 FunctionGraph HTTP 函数.....39
8 创建使用自定义认证且后端为 FunctionGraph 的 API.....43
8.1 方案概述.....43
8.2 资源规划.....43
8.3 构建程序.....44
8.4 添加事件源.....49
8.5 调试并调用 API.....50
9 函数+APIG：处理文件上传.....52
9.1 方案概述.....52
9.2 资源规划.....52
9.3 操作流程.....52
9.3.1 NodeJS 语言方案.....53
9.3.2 Python 语言方案.....55
10 使用函数处理 IOT 数据.....57
10.1 案例概述.....57
10.2 准备.....58
10.3 构建函数程序.....60
11 函数+DEW：加解密文件.....63
11.1 案例描述.....63
11.2 准备.....64
11.3 构建程序.....65
11.4 添加事件源.....71
11.5 处理文件.....71
12 工作流+函数：自动化处理 OBS 中数据.....73
12.1 案例概述.....73
12.2 准备.....74
12.3 构建程序.....75
12.4 处理图片.....80
13 函数+LTS：日志实时过滤.....82
13.1 案例概述.....82
13.2 准备.....83
13.3 构建程序.....85
13.4 添加事件源.....87
13.5 处理结果.....87
13.6 应用扩展.....89
14 AS&FunctionGraph 支持优雅关机.....90
14.1 案例概述.....90
14.2 准备.....90

14.3 构建程序.....91
14.4 配置消息通知.....93
14.5 处理展示.....95
15 使用 Go 构建 FunctionGraph HTTP 函数.....97

1 使用函数压缩图片

1.1 案例概述

本手册基于函数工作流服务实践所编写，用于指导您使用函数工作流服务实现图片压缩的功能。

场景介绍

- 将图片上传到特定的OBS桶中
- 将用户上传的每个图像的尺寸进行压缩
- 将处理完后的图像上传到另一个指定的OBS桶中

说明

1. 本教程必须使用两个不同的OBS桶。
2. 保证函数和OBS桶在一个区域（区域都选择默认即可）。

实现流程

- 在OBS服务中，创建两个桶。
- 创建函数，设置OBS触发器。
- 用户向其中一个桶上传图片。
- 触发函数执行，对图片进行压缩处理。
- 函数将处理后的图片上传到指定桶中。

说明

完成本教程后，您的公有云账户将存在以下资源：

1. 2个OBS桶（上传需要处理的图像和存储处理后的图像）
2. 一个创建缩略图的函数（fss_examples_image_thumbnail）
3. 一个OBS触发器，用来关联函数和OBS桶

1.2 准备

创建函数及添加事件源之前，需要创建两个OBS桶，分别用来保存用户上传的图片和压缩处理后的图片。

OBS桶创建以后，需要创建“委托”，给FunctionGraph函数赋权，确保FunctionGraph函数能够访问到OBS资源。

创建 OBS 桶

注意事项

- 上传图片的源桶、输出图片的目标桶和函数必须处于同一个区域下。
- 必须使用两个不同的桶。如果使用一个桶，会无限执行函数。（源桶上传图片会触发函数执行，从而无限循环）。

操作步骤

步骤1 登录对象存储服务控制台，单击“创建桶”，进入“创建桶”界面。

步骤2 在“创建桶”界面，填写存储桶信息。

- 区域：根据实际情况设置
- 桶名称输入：“your-bucket-input”
- 数据冗余存储策略：“单AZ存储”
- 默认存储类别：“标准存储”
- 桶策略："私有"
- 服务端加密：“不开启加密”
- 归档数据直读：“关闭”

其余参数保持默认，单击“立即创建”，完成源桶创建。

步骤3 重复步骤2，创建目标桶。

区域及存储类别与源桶保持一致，桶名称命名为“your-bucket-output”。

步骤4 完成桶创建以后，OBS桶列表有your-bucket-input、your-bucket-output两个桶。

----结束

创建委托

步骤1 在服务控制台左侧导航栏，选择“管理与监管 > 统一身份认证服务”进入统一身份认证服务控制台，在左侧导航栏单击“委托”，进入“委托”界面。

步骤2 单击“创建委托”，进入“创建委托”界面。

步骤3 填写委托信息。

- 委托名称：输入“serverless_trust”。
- 委托类型：选择“云服务”。
- 云服务：选择“函数工作流 FunctionGraph”。

- 持续时间：选择“永久”。
- 描述：填写描述信息。

步骤4 单击“下一步”，进入委托选择页面，在“配置权限”界面勾选“Tenant Administrator”，单击“下一步”。

说明

Tenant Administrator：拥有该权限的用户可以对企业拥有的所有云资源执行任意操作。

步骤5 根据实际业务需求选择资源授权范围，单击“确定”，完成权限委托设置。

----结束

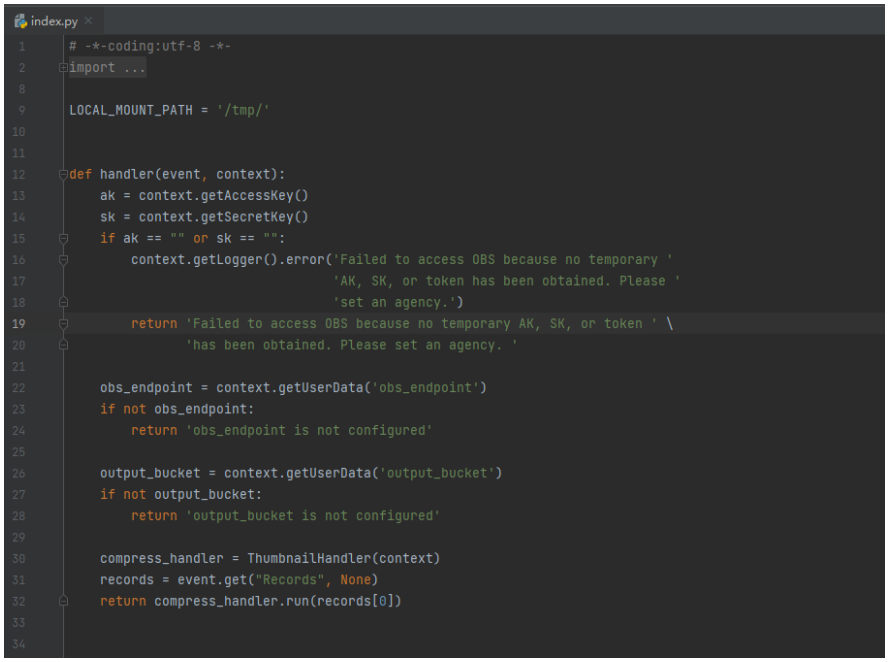
1.3 构建程序

本例提供了实现图片压缩功能的程序包，使用空白模板创建函数，用户可以**下载（fss_examples_image_thumbnail.zip）**（**sha256校验包**）学习使用。

创建程序包

本例使用Python语言实现图片压缩的功能，有关函数开发的过程请参考**Python函数开发**。本例不再介绍业务功能实现的代码，样例代码目录如**图1-1**所示。

图 1-1 样例代码目录



其中index.py为函数执行的入口文件，index.py中入口函数的代码片段如下，参数“output_bucket”为压缩后的图片存储地址，需要在创建函数时配置自定义参数。

```
def handler(event, context):
    ak = context.getAccessKey()
    sk = context.getSecretKey()
    if ak == "" or sk == "":
        context.getLogger().error('Failed to access OBS because no temporary '
                                  'AK, SK, or token has been obtained. Please '
                                  'set an agency.')
```

```
    return 'Failed to access OBS because no temporary AK, SK, or token ' \
           'has been obtained. Please set an agency. '

obs_endpoint = context.getUserData('obs_endpoint')
if not obs_endpoint:
    return 'obs_endpoint is not configured'

output_bucket = context.getUserData('output_bucket')
if not output_bucket:
    return 'output_bucket is not configured'

compress_handler = ThumbnailHandler(context)
records = event.get("Records", None)
return compress_handler.run(records[0])
```

创建函数

创建函数的时候，必须选择委托包含OBS访问权限的委托，否则不能使用OBS服务。

步骤1 登录函数工作流控制台，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。

步骤2 单击“创建函数”，进入创建函数流程。

步骤3 填写函数配置信息。

输入基础配置信息，完成后单击“创建函数”。

- 函数名称：输入“fss_examples_image_thumbnail”
- 委托名称：选择创建委托中创建的“serverless_trust”
- 运行时语言：选择“Python3.6”

步骤4 进入fss_examples_image_thumbnail函数详情页，配置如下信息。

1. 在“代码”页签，代码选择“上传自ZIP文件”，上传样例代码中的“fss_examples_image_thumbnail.zip”，完成后单击“部署”。
2. 在“设置 > 常规设置”页签，设置如下信息，完成后单击“保存”。
   - 内存：选择“256”
   - 执行超时时间：输入“40”
   - 函数执行入口：默认“index.handler”，无需修改
   - 所属应用：默认“default”
   - 描述：输入“压缩图片”
3. 在“设置 > 环境变量”页签，输入环境信息，完成后单击“保存”。

   键output_bucket：index.py文件中定义的存放输出图片的OBS桶参数，值your-bucket-output： 创建OBS桶中创建的存放输出图片OBS桶;

   键obs_endpoint：index.py文件中定义的存放输出图片的OBS桶的地址参数，值obs.region.myhuaweicloud.com。

表 1-1 环境变量说明

环境变量	说明
obs_endpoint	OBS服务终端节点，获取地址请参考地区和终端节点。
output_bucket	存放输出图片的OBS桶。

----结束

添加依赖包

示例代码依赖pillow包，需要通过依赖包的形式进行引入，步骤如下。

步骤1 用户进入fss_examples_image_thumbnail函数详情页，在“代码”页签，单击页面最底部的“添加依赖包”。

步骤2 添加公共依赖包“pillow-7.1.2”。

图 1-2 添加依赖包

----结束

说明

demo包的函数代码中已预置了依赖包的引用配置，因此添加依赖包后无需再进行依赖包的引用配置。

1.4 添加事件源

OBS桶及函数创建以后，可以为函数添加事件源，添加OBS事件源是通过创建OBS触发器实现的，步骤如下。

步骤1 用户进入fss_examples_image_thumbnail函数详情页，在“设置 > 触发器”页签，单击“创建触发器”，弹出创建触发器界面。

步骤2 触发器类型选择“对象存储服务 (OBS)”，填写触发器配置信息，如图1-3所示。

- 桶：选择创建OBS桶中创建的“your-bucket-input”桶。
- 事件：选择“Post”、“Put”。

图 1-3 创建触发器

步骤3 单击"确定"，完成触发器创建。

说明

OBS触发器创建以后，当有图片上传或更新至your-bucket-input桶时，生成事件，触发函数执行。

----结束

1.5 图片处理

当图片上传或更新至your-bucket-input桶时，会生成事件，触发函数运行，将上传图片压缩，保存在your-bucket-output中。

上传图片生成事件

登录**对象存储服务控制台**，进入your-bucket-input桶对象界面，上传image.jpg图片，如**图1-4**所示。

图 1-4 上传图片

名称	存储类别	大小	加密状态
image.jpg	标准存储	28.31 KB	未加密

说明

原始图片image.jpg大小超过28KB。

触发函数自动运行

上传图片至your-bucket-input桶，OBS生成事件触发函数运行，将图片压缩，压缩后的图片存放在your-bucket-output桶中。可以在函数详情页日志页签查看函数运行日志。

进入your-bucket-output桶对象界面，查看压缩后的图片大小。

图 1-5 压缩图片

名称	存储类别	大小	加密状态
image-thumbnail.jpg	标准存储	7.82 KB	未加密

2 使用函数为图片打水印

2.1 案例概述

本手册基于函数工作流服务实践所编写，用于指导您使用函数工作流服务实现为图片打水印的功能。

场景介绍

- 将图片上传到特定的OBS桶中。
- 将用户上传的每个图片打水印。
- 将处理完后的图像上传到另一个指定的OBS桶中。

说明

1. 本教程必须使用两个不同的OBS桶。
2. 保证函数和OBS桶在一个区域（区域都选择默认即可）。

实现流程

- 在OBS服务中，创建两个桶。
- 创建函数，设置OBS触发器。
- 用户向其中一个桶上传图片。
- 触发函数执行，对图片打水印。
- 函数将处理后的图片上传到指定桶中。

说明

完成本教程后，您的公有云账户将存在以下资源：

1. 2个OBS桶（上传需要处理的图像和存储处理后的图像）
2. 一个为图片打水印的函数
3. 一个OBS触发器，用来关联函数和OBS桶

2.2 准备

创建函数及添加事件源之前，需要创建两个OBS桶，分别用来保存用户上传的图片和打水印后输出的图片。

OBS桶创建以后，需要创建委托，给FunctionGraph函数赋权，确保FunctionGraph函数能够访问到OBS资源。

创建 OBS 桶

注意事项

- 上传图片的源桶、输出图片的目标桶和函数必须处于同一个区域下。
- 必须使用两个不同的桶。如果使用一个桶，会无限执行函数。（源桶上传图片会触发函数执行，从而无限循环）。

操作步骤

步骤1 登录对象存储服务控制台，单击“创建桶”，进入“创建桶”界面。

步骤2 在“创建桶”界面，填写存储桶信息。

- 区域：根据实际情况设置
- 数据冗余存储策略：“单AZ存储”
- 桶名称输入：“hugb-bucket-input”
- 默认存储类别：“标准存储”
- 桶策略："私有"
- 服务端加密：“关闭”
- 归档数据直读：“关闭”

单击“立即创建”，完成源桶创建。

步骤3 重复步骤2，创建目标桶。

区域及存储类别与源桶保持一致，桶名称命名为“hugb-bucket-output”。

步骤4 完成桶创建以后，OBS桶列表有hugb-bucket-input、hugb-bucket-output两个桶。

----结束

创建委托

步骤1 在服务控制台左侧导航栏，选择“管理与监管 > 统一身份认证服务”进入统一身份认证服务控制台，在左侧导航栏单击“委托”，进入“委托”界面。

步骤2 单击“创建委托”，进入“创建委托”界面。

步骤3 填写委托信息。

- 委托名称：输入“serverless_trust”。
- 委托类型：选择“云服务”。
- 云服务：选择“函数工作流 FunctionGraph”。
- 持续时间：选择“永久”。
- 描述：填写描述信息。

步骤4 单击“下一步”，进入委托选择页面，在“配置权限”界面勾选“Tenant Administrator”，单击“下一步”。

说明

Tenant Administrator：拥有该权限的用户可以对企业拥有的所有云资源执行任意操作。

步骤5 根据实际业务需求选择资源授权范围，单击“确定”，完成权限委托设置。

----结束

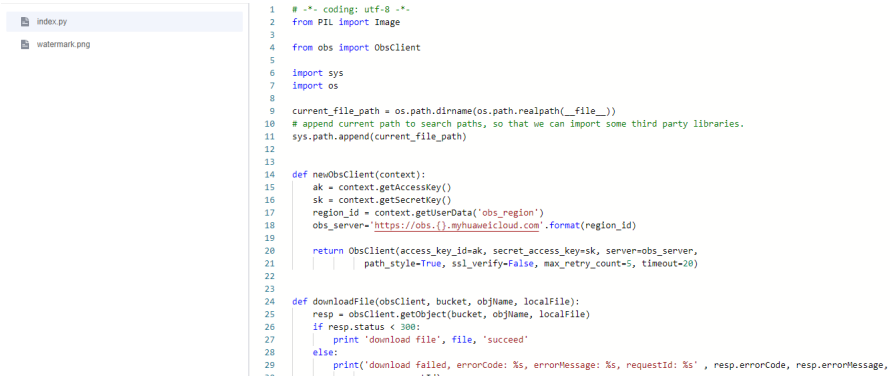
2.3 构建程序

本例提供了为图片打水印功能的程序包，使用空白模板创建函数，用户可以**下载（watermark.zip）**学习使用。

创建程序包

本例使用Python语言实现为图片打水印的功能，有关函数开发的过程请参考**Python函数开发**。本例不再介绍业务功能实现的代码，样例代码目录如**图2-1**所示。

图 2-1 样例代码目录



其中index.py为函数执行的入口文件，index.py中入口函数的代码片段如下，参数“obs_output_bucket”为打水印后的图片存储地址，需要在创建函数时配置自定义参数。

```
def handler(event, context):
    srcBucket, srcObjName = getObjInfoFromObsEvent(event)
    outputBucket = context.getUserData('obs_output_bucket')

    client = newObsClient(context)
    # download file uploaded by user from obs
    localFile = "/tmp/" + srcObjName
    downloadFile(client, srcBucket, srcObjName, localFile)

    outFileName, outFile = watermark_image(localFile, srcObjName)
    # 将转换后的文件上传到新的obs桶中
    uploadFileToObs(client, outputBucket, outFileName, outFile)

    return 'OK'
```

创建函数

创建函数的时候，必须选择委托包含OBS访问权限的委托，否则不能使用OBS服务。

步骤1 登录**函数工作流控制台**，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。

步骤2 单击“创建函数”，进入创建函数流程。

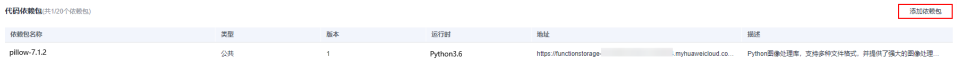
步骤3 填写函数配置信息。

输入基础配置信息，完成后单击“创建函数”。

- 函数名称：输入“fss_examples_image_watermark”
- 委托名称：选择**创建委托**中创建的“serverless_trust”
- 运行时语言：选择“Python3.6”

步骤4 进入fss_examples_image_watermark函数详情页，在“代码”页签，单击页面最底部的“添加依赖包”，添加公共依赖包“pillow-7.1.2”。

图 2-2 添加依赖包



步骤5 进入fss_examples_image_watermark函数详情页，配置如下信息。

1. 在“代码”页签，代码选择“上传自ZIP文件”，上传样例代码watermark.zip，完成后单击“部署”。
2. 在“设置 > 常规设置”页签，设置如下信息，完成后单击“保存”。
   - 内存：选择“128”
   - 执行超时时间：输入“3”
   - 函数执行入口：默认“index.handler”，无需修改
   - 所属应用：默认“default”
   - 描述：输入“图片打水印”
3. 在“设置 > 环境变量”页签，输入环境信息，完成后单击“保存”。以下截图仅供参考，在实际使用中，请根据实际情况替换。

   键obs_output_bucket：为index.py文件中定义的存放输出水印图片的OBS桶参数，值hugb-bucket-output：为**创建OBS桶**中创建的存放输出水印图片的OBS桶。

   键obs_region： OBS桶obs_output_bucket所在的Region。

表 2-1 环境变量说明

环境变量	说明
obs_region	OBS桶所属区域，此处应与函数所属区域保持一致。
obs_output_bucket	存放输出水印图片的OBS桶。

----结束

2.4 添加事件源

OBS桶及函数创建以后，可以为函数添加事件源，添加OBS事件源是通过创建OBS触发器实现的，步骤如下。

步骤1 用户进入fss_examples_image_watermark函数详情页，在“触发器”页签，单击“创建触发器”，弹出“创建触发器”界面。

步骤2 触发器类型选择“存储(OBS)”，填写触发器配置信息，如图2-3所示。

桶选择创建OBS桶中创建的“hugb-bucket-input”桶。

事件选择“Post”、“Put”。

图 2-3 创建 OBS 触发器

步骤3 单击“确定”，完成触发器创建。

说明

OBS触发器创建以后，当有图片上传或更新至hugb-bucket-input桶时，生成事件，触发函数执行。

----结束

2.5 处理图片

当图片上传后更新至hugb-bucket-input桶时，会生成事件，触发函数运行，将上传图片打水印，保存在hugb-bucket-output中。

上传图片生成事件

登录对象存储服务控制台，进入hugb-bucket-input桶对象界面，上传image.jpg图片，如图2-4所示。

图 2-4 上传图片

触发函数自动运行

上传图片至hugb-bucket-input桶，OBS生成事件触发函数运行，为图片打水印，输出图片存放在hugb-bucket-output桶中。可以在fss_examples_image_watermark函数详情页“日志”页签查看函数运行日志。

进入hugb-bucket-output桶对象界面，可以看到输出的图片image.jpg，如图2-5所示。单击操作列的“下载”可将图片下载至本地查看图片处理效果，效果如图2-6所示。

图 2-5 输出图片

图 2-6 效果图

3 使用函数处理 DIS 数据

3.1 案例概述

本手册基于函数工作流服务实践所编写，用于指导您使用函数工作流服务实现处理DIS数据的功能。

场景介绍

使用数据接入服务（DIS）采集IOT实时数据流，需要将采集到的数据进行处理（比如格式转换），然后存储到表格存储服务（CloudTable Service）中，使用FunctionGraph函数可以实现此功能。

实现流程

- 创建虚拟私有云和集群。
- 构建实现数据处理功能的程序，将程序打包。
- 在函数工作流服务中，创建函数。
- 配置DIS事件，测试函数，处理数据。

3.2 准备

案例实现的功能是将DIS数据格式转换，存储到表格存储服务中，所以需要先在表格存储服务创建集群，在创建集群时需要使用虚拟私有云。

创建函数之前，需要创建委托，给FunctionGraph函数赋权，确保FunctionGraph函数能够访问到DIS和CloudTable资源。

创建虚拟私有云

步骤1 登录虚拟私有云控制台，单击“创建虚拟私有云”，进入“创建虚拟私有云”界面。

步骤2 填写私有云配置信息。

基本信息中名称输入“vpc-cloudtable”，其他使用系统默认。

子网配置使用系统默认。

步骤3 确认配置信息无误，单击“立即创建”，创建虚拟私有云。

----结束

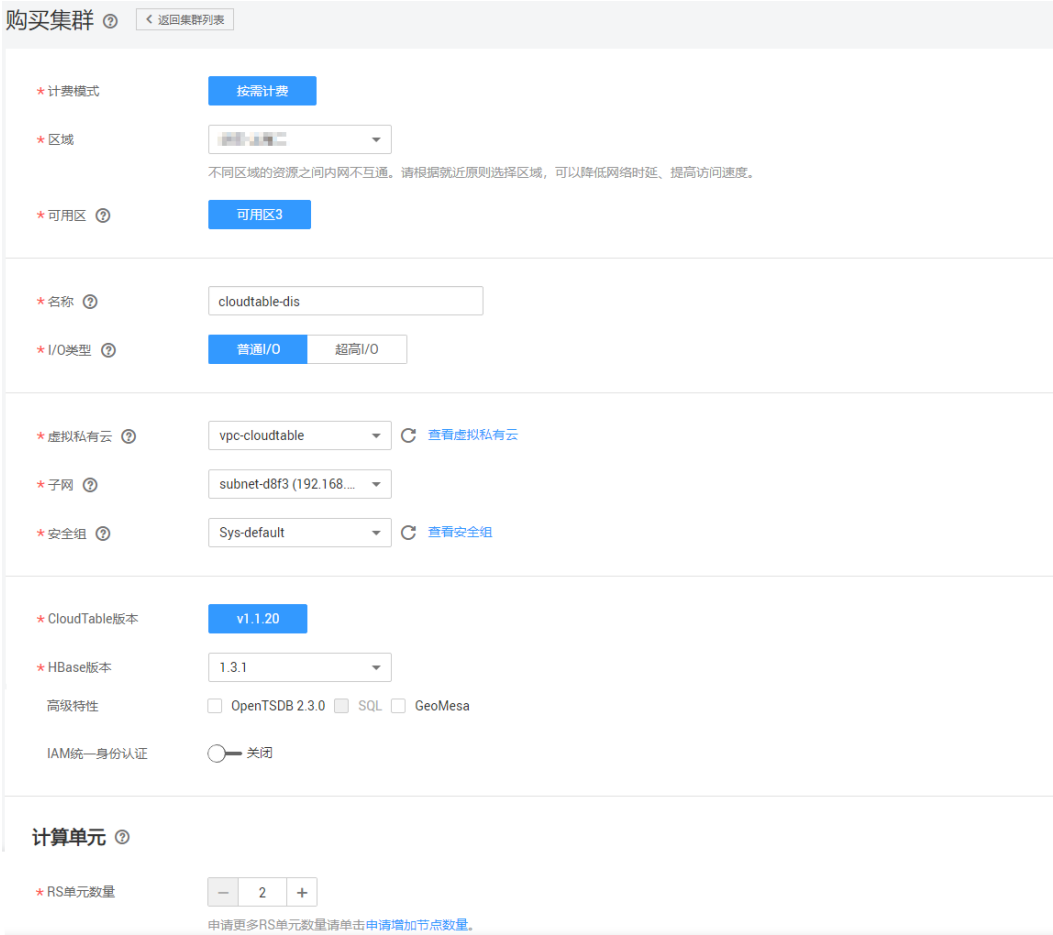
创建集群

步骤1 在服务控制台左侧导航栏，选择“大数据 > 表格存储服务”，进入表格存储服务控制台后，在“集群模式”界面，单击“购买集群”，进入“购买集群”界面。

步骤2 填写集群配置信息。

- 区域使用系统默认。
- 名称输入“cloudtable-dis”。
- 虚拟私有云选择**创建虚拟私有云**中创建的“vpc-cloudtable”。
- 其他配置保持默认，无需修改。

图 3-1 购买集群



步骤3 确认配置信息无误，单击“立即购买”，创建集群。

图 3-2 创建集群

说明

创建集群需要较长时间，可以从图3-2中查看进度，请耐心等待。

----结束

创建委托

步骤1 在服务控制台左侧导航栏，选择“管理与监管 > 统一身份认证服务”，进入统一身份认证服务控制台后，在左侧导航栏单击“委托”，进入“委托”界面。

步骤2 单击“创建委托”，弹出“创建委托”界面。

步骤3 填写委托信息。

- 委托名称：输入“DISDemo”。
- 委托类型：选择“云服务”。
- 云服务：选择“函数工作流 FunctionGraph”。
- 持续时间：选择“永久”。

步骤4 单击“下一步”，进入委托选择页面，在“配置权限”界面勾选“Tenant Administrator”，单击“确定”

图 3-3 创建委托

说明

Tenant Administrator：拥有该权限的用户可以对企业拥有的所有云资源执行任意操作。

步骤5 单击“确定”，完成权限委托设置。

----结束

3.3 构建程序

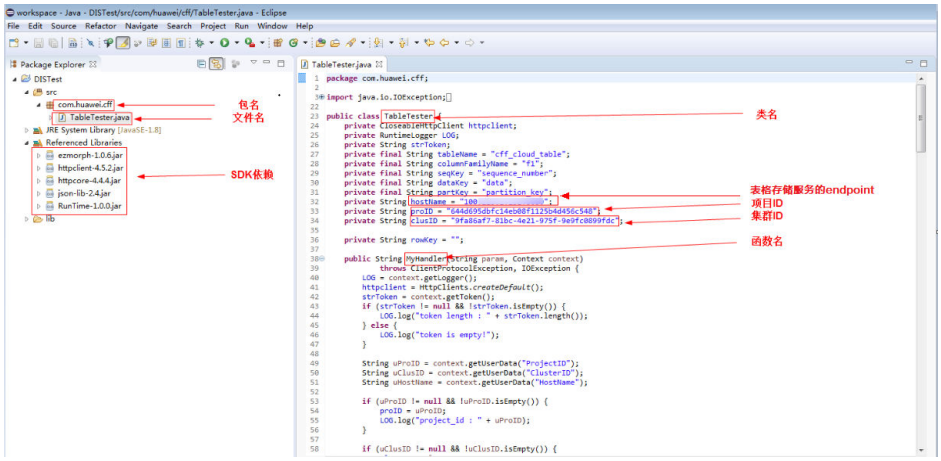
本例提供了DIS数据流格式转换的源码和程序包（包含函数依赖），使用空白模板创建函数，用户可以下载、学习使用。

创建工程

本例使用Java语言实现DIS数据流格式转换功能，有关函数开发的过程请参考**Java函数开发指南**，本例不再介绍业务功能实现的代码。

下载样例源码（fss_examples_dis_cloudtable_src.zip），解压缩，在Eclipse中导入工程，如**图3-4**所示。

图 3-4 样例代码说明

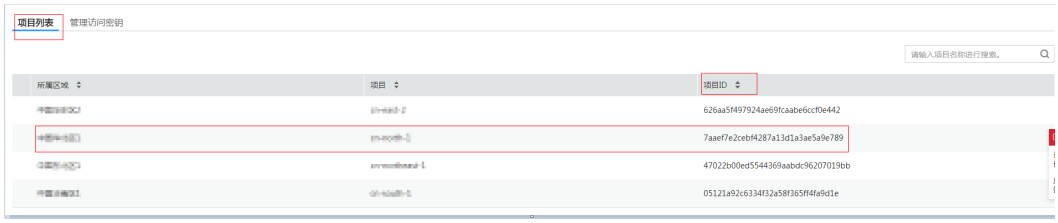


在样例代码中，需要修改proID（项目ID）、clusID（集群ID）、hostName（表格存储服务的endpoint）并保存。

项目ID获取方法：进入“个人中心 > 我的凭证”，如**图3-5**所示，在“项目列表”获得项目ID，如**图3-6**所示。

图 3-5 我的凭证

图 3-6 项目 ID



集群ID获取方法：登录**表格存储服务**，进入集群管理，选择**创建集群**中创建的cloudtable-dis集群，进入集群详情页，可以查看集群ID，如**图3-7**所示。

图 3-7 集群 ID

创建FunctionGraph函数时，需要设置函数执行入口，Java函数执行入口格式为：[包名].[文件名].[函数名]，上述源码对应的函数执行入口为：com.huawei.cff.TableTester.MyHandler。

程序打包

使用Eclipse生成Jar包，步骤如下图所示，得到Table Tester.jar文件。

图 3-8 Export

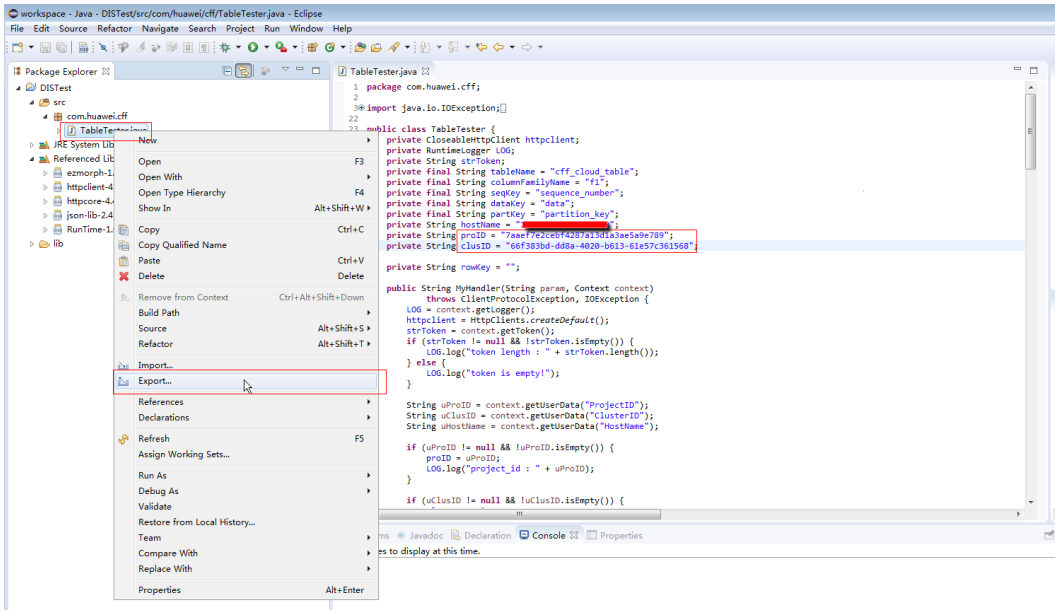


图 3-9 选择类型

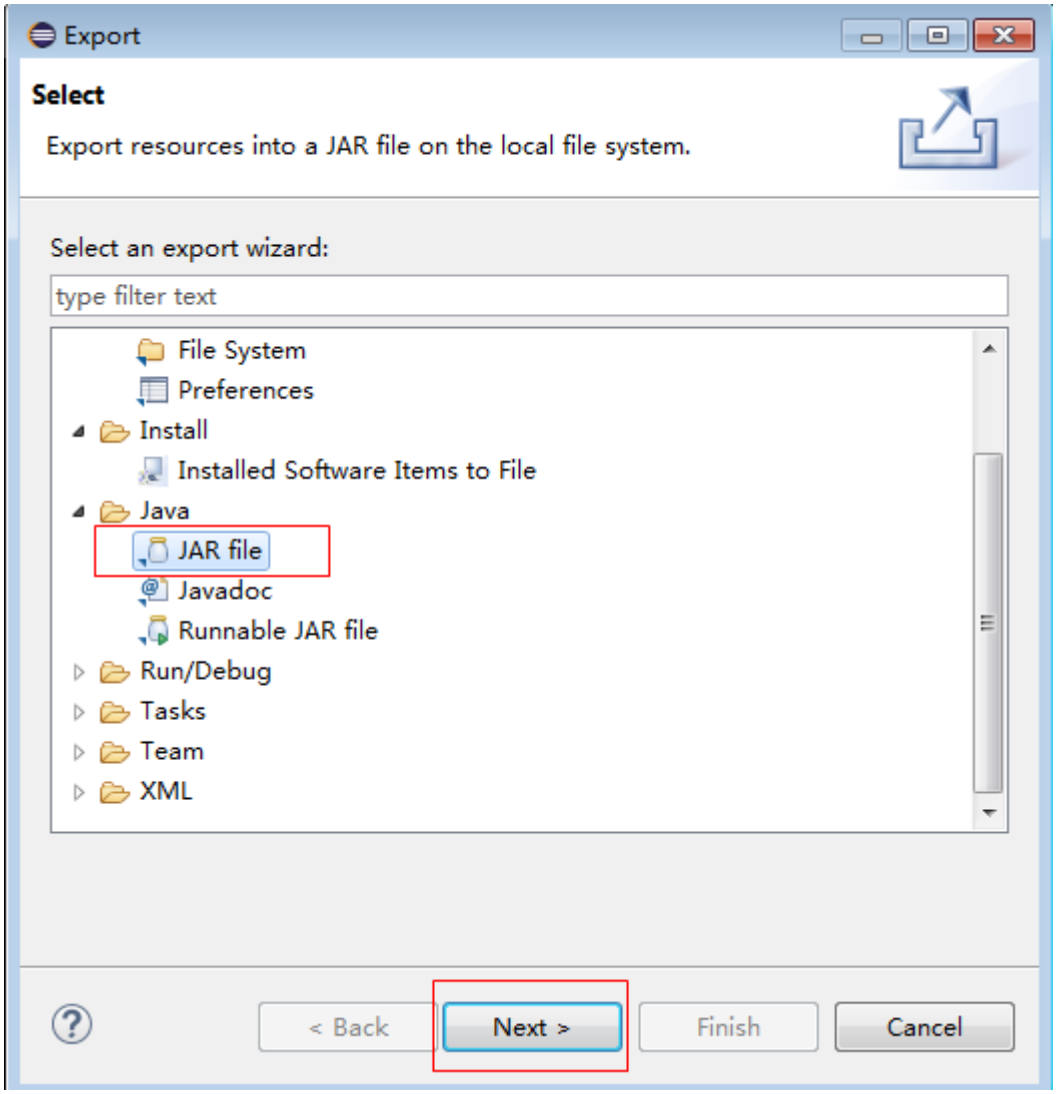
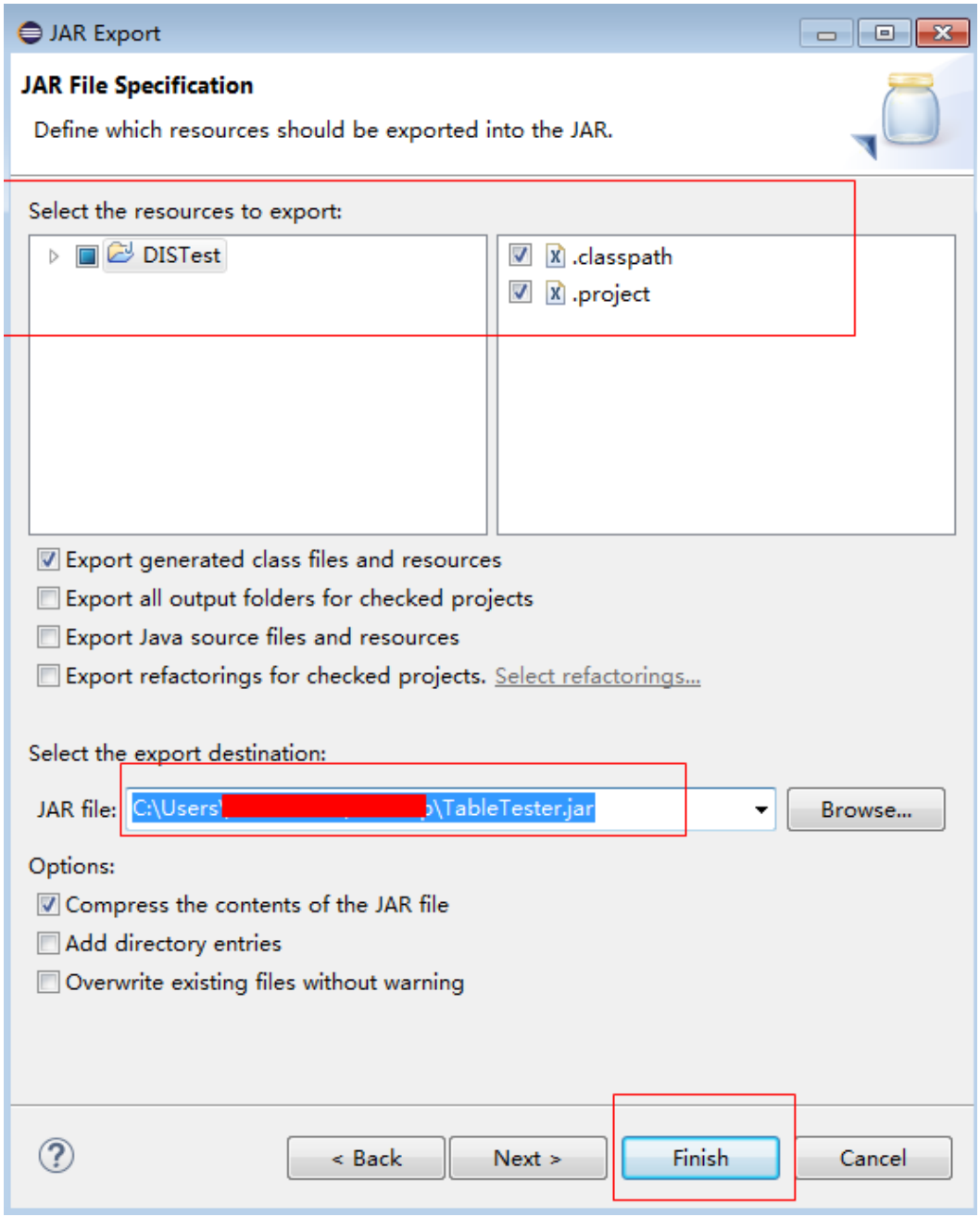


图 3-10 发布



将函数依赖打包，方法如下。

下载程序包（fss_examples_dis_cloudtable.zip）文件，解压缩目录如图3-11所示。使用Table Tester.jar替换DIS Test.jar，替换文件目录后如图3-12所示。打ZIP包，如图3-13所示，得到disdemo.zip文件。

图 3-11 文件目录

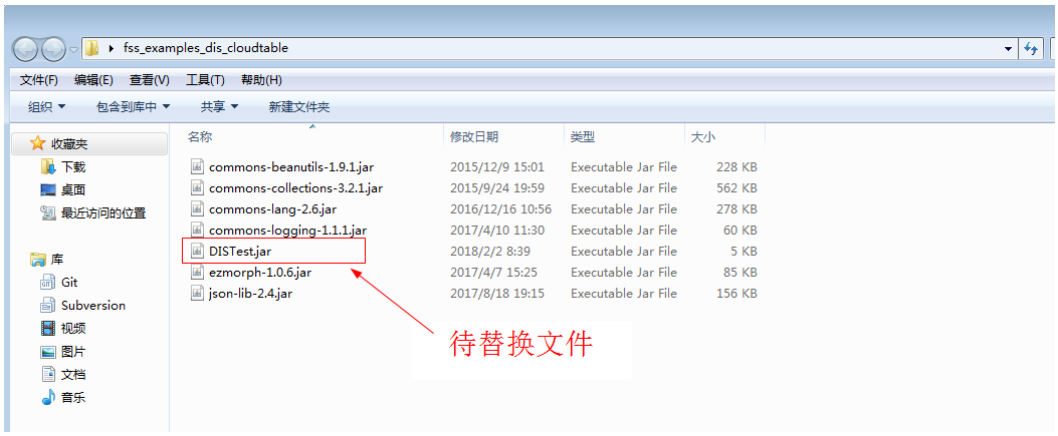


图 3-12 替换后文件目录

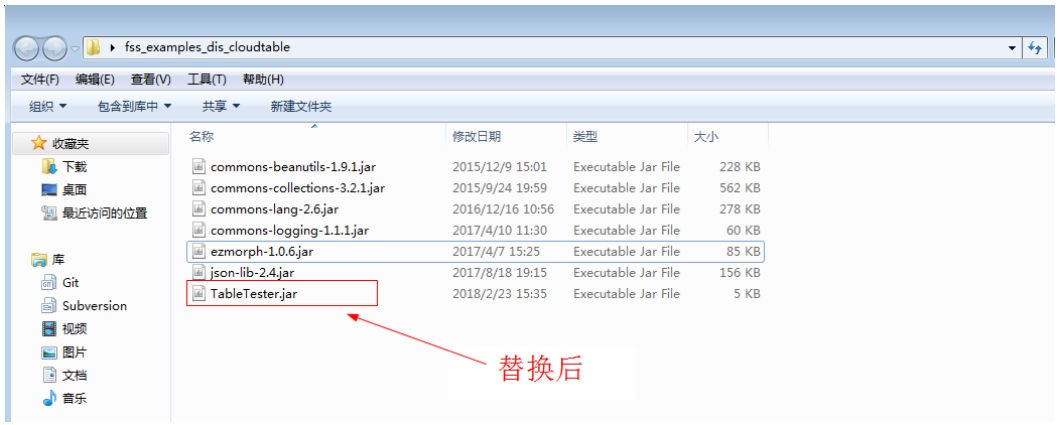
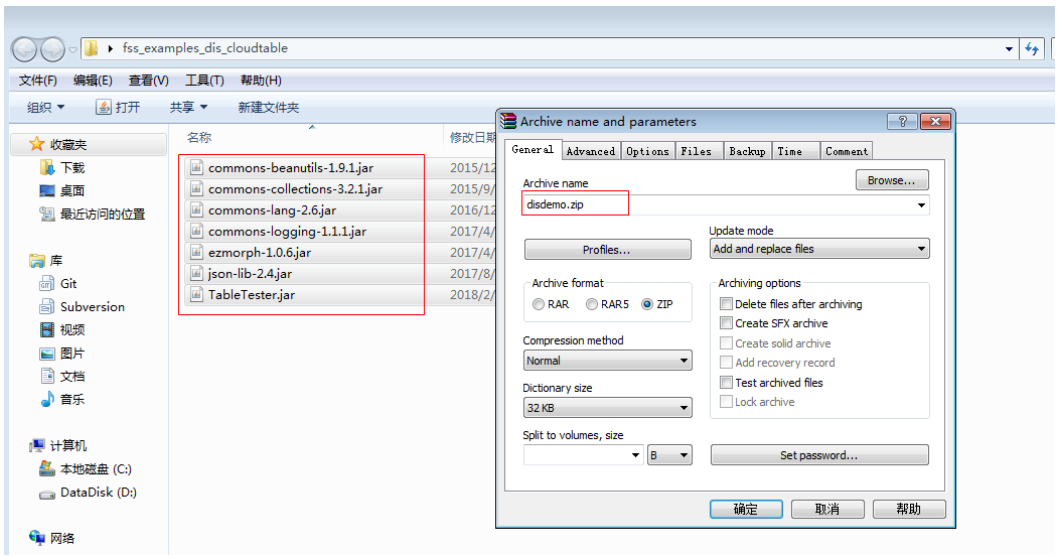


图 3-13 打 ZIP 包



创建函数

创建函数的时候，必须选择能够访问到DIS和CloudTable资源的委托。

步骤1 登录函数工作流控制台，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。

步骤2 单击“创建函数”，进入创建函数流程。

步骤3 填写函数基本信息，完成后单击“创建函数”。

- 函数名称：输入“DISDemo”
- 委托名称：选择准备中创建的“DISDemo”
- 运行时语言选择：“Java 8”

步骤4 进入函数详情页，配置如下信息。

- 在“设置 > 常规设置”页签，修改函数执行入口为“com.huawei.cff.TableTester.MyHandler”，单击“保存”。
- 在“代码”页签，选择“上传自Zip文件”，选择上传程序打包中的代码包“disdemo.zip”，单击“部署”。

----结束

修改函数配置

函数创建完成后，函数默认内存为128MB，超时时间默认为3s，资源太少，需要修改。

步骤1 进入DISDemo函数详情页，在“设置 > 基本设置”页签，修改配置信息。

- 内存：选择“512”
- 执行超时时间：输入“15”
- 其他配置项不修改。

步骤2 单击“保存”，保存配置信息。

----结束

3.4 添加事件源

函数创建以后，可以为函数添加事件源，本例通过配置DIS测试事件，模拟DIS输入数据，步骤如下。

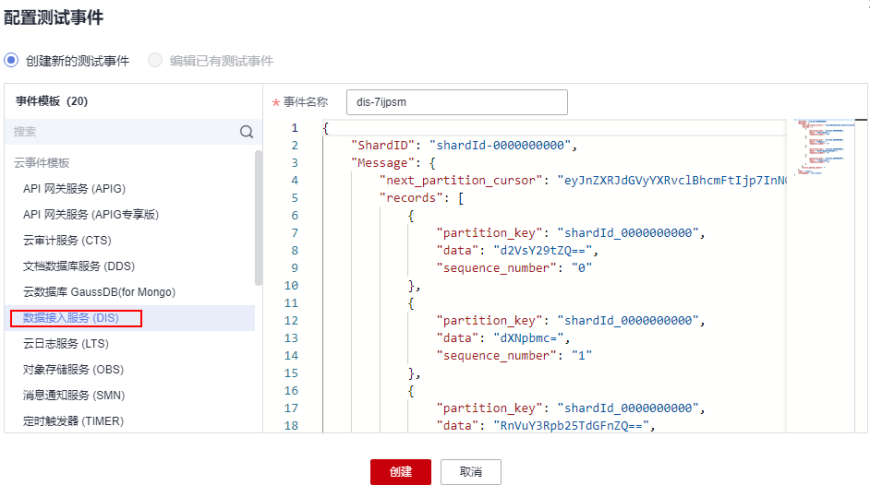
步骤1 用户进入DISDemo函数详情页，在“代码”页签下，选择配置测试事件，如图3-14所示，弹出“配置测试事件页”。

图 3-14 配置测试事件

步骤2 在“配置测试事件页”，输入配置信息，如图3-15所示。

- 配置测试事件：选择“创建新的测试事件”。
- 事件模板：选择“数据接入服务（DIS）”。
- 事件名称：输入“dis-test”。

图 3-15 测试事件



步骤3 单击“创建”，完成测试事件配置。

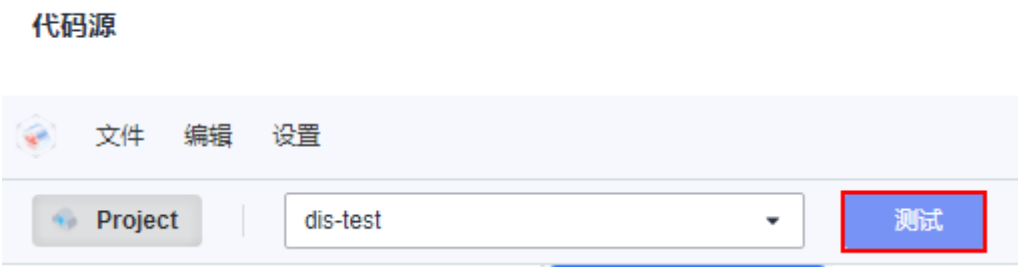
----结束

3.5 处理数据

处理模拟数据步骤如下。

步骤1 用户进入DISDemo函数详情页，选择“dis-test”测试事件，单击“测试”，测试函数，如图3-16所示。

图 3-16 配置测试事件



步骤2 函数执行成功后，部分函数日志如图3-17所示，全部的日志信息，可以到“日志”页签查询。

图 3-17 函数执行结果

----结束

4 函数+LTS：日志实时分析实战

4.1 案例概述

场景介绍

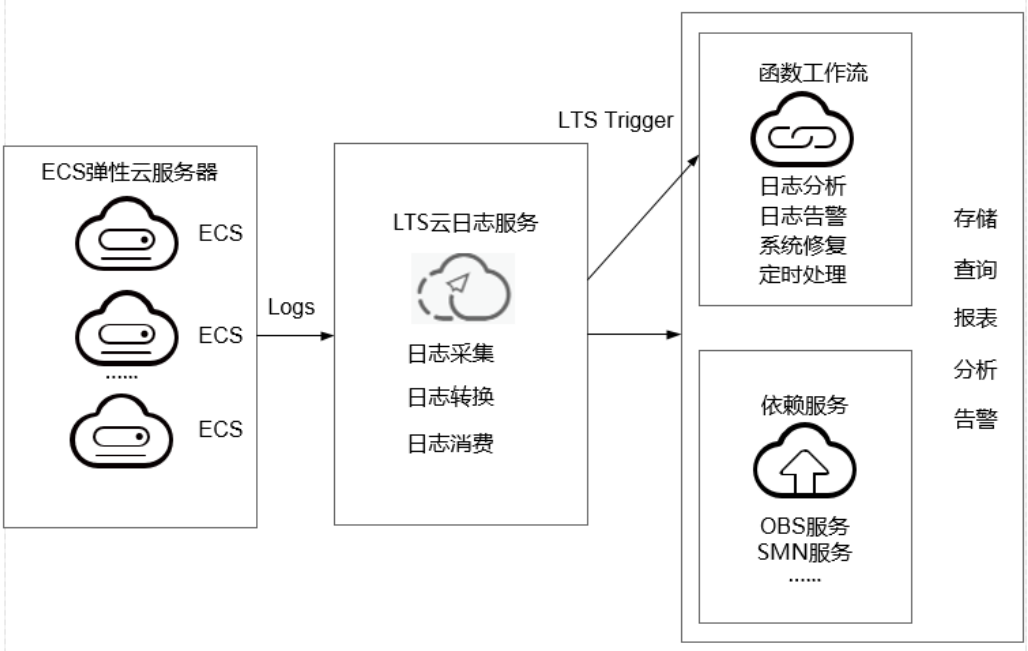
通过LTS云日志服务，快速完成ECS等服务器的任务运行日志采集、加工和转换。

通过函数工作流服务中的函数创建LTS触发器获取日志数据，经由自定义函数对日志中的关键信息进行分析和处理，过滤出告警日志。

SMN消息通知服务通过短信和邮件推送告警信息，通知业务人员进行处理。

将函数处理后的日志数据投递至OBS桶中集中存储，便于后续处理。处理流程如图4-1。

图 4-1 处理流程



案例价值点

- 通过LTS日志服务，快速完成日志采集和转换。
- 基于serverless无服务架构的函数计算提供数据加工、分析，事件触发，弹性伸缩，无需运维，按需付费。
- 结合SMN消息通知服务提供日志、告警功能。

4.2 准备

日志采集和存储

- 在云日志服务创建日志组，此处以polo.guoying为例，创建过程请参考**创建日志组**。
- 在云日志服务创建日志流，此处以lts-topic-gfz3为例，创建过程请参考**创建日志流**。
- 在云日志服务配置Agent，快速将ECS等服务器上日志采集到指定的日志组，配置过程请参考**安装ICAgent**。

告警消息推送

- 在SMN消息通知服务创建主题，此处以主题名称fss_test为例，创建过程请参考**创建SMN日志主题**。
- 在SMN消息通知服务订阅主题，用于将告警消息推送至该主题下的订阅终端，此处以添加邮件订阅终端为例，订阅fss_test主题，订阅过程请参考**订阅主题**。
- SMN主题名称需添加在函数的环境变量中，以便将告警消息推送至该主题下的订阅终端。环境变量名称为“SMN_Topic”，环境变量值为SMN主题名称。以主题名称fss_test为例，在函数的环境变量配置中添加：“SMN_Topic”：“fss_test”。

说明

订阅主题可选择通过邮件、短信、HTTP/HTTPS等形式推送告警消息

本案例中推送告警消息的事件是：当日志事件通过LTS触发器触发函数执行时，函数中过滤告警日志，产生的告警消息推送至SMN主题的订阅终端。

云端数据加工处理

在OBS对象存储服务创建OBS桶和OBS对象，并配置事件通知。

1. 在OBS对象存储服务创建OBS桶和OBS对象，如**图2 OBS桶**所示，创建过程请参考**创建OBS桶**。

图 4-2 OBS 桶

说明

创建的OBS桶名为“logstore”，OBS对象为“log.txt”用于存储日志数据。

创建委托

1. 登录统一身份认证服务控制台。
2. 在统一身份认证服务的左侧导航窗格中，选择“委托”页签，单击右上方的“+创建委托”。

图 4-3 创建委托

3. 开始配置委托。
   - 委托名称：LtsOperation。
   - 委托类型：选择“云服务”。
   - 云服务：选择“函数工作流 FunctionGraph”。
   - 持续时间：选择“永久”。
   - 描述：填写描述信息。
4. 单击“下一步”，进入委托选择页面，在右方搜索框中搜索“LTS Administrator”权限和“Tenant Administrator”并勾选。

图 4-4 选择权限

说明

选择“LTS Administrator”，由于该策略有依赖，在勾选LTS Administrator时，还会自动勾选依赖的策略：Tenant Guest。

5. 单击“下一步”，请根据业务需要选择权限的作用范围。

4.3 构建程序

本案例提供了实现提取告警日志功能的程序包，使用空白模板创建函数，用户可以**下载（fss_examples_logstore_warning.zip）**学习使用。

创建功能函数

创建实现日志提取功能的函数，将示例代码包上传。创建过程请参考创建函数，运行时语言选择“Python2.7”，委托名称选择创建委托中的“LtsOperation”。

函数实现的功能是：将收到的日志事件数据进行base64解码，然后提取出包含“WRN”、“WARN”、“ERR”或“ERROR”关键字的告警日志，将此级别的日志投递至OBS桶中集中存储。可根据您的业务日志的具体内容配置相应的日志提取条件。

设置环境变量

在函数配置页签需配置环境变量，分别表示OBS桶地址、OBS桶名称以及OBS对象名称，说明如表1 环境变量说明表所示。

表 4-1 环境变量说明表

环境变量	说明
obs_address	OBS服务终端节点，获取地址请参考地区和终端节点。
obs_store_bucket	日志存储的目标桶名称。
obs_store_objName	日志存储的目标文件。
SMN_Topic	SMN主题名称。
region	您所在区域的region值，获取请参考地区和终端节点。

环境变量的设置过程请参考使用环境变量。

4.4 添加事件源

选择准备中创建的日志组和日志主题，创建LTS触发器，LTS触发器配置如图4-5所示。

图 4-5 创建 LTS 触发器

LTS日志服务的消费端在日志累积大小或日志累积时间满足条件时消费LTS日志数据，并根据订阅该组LTS日志数据的函数URN触发函数执行。

4.5 处理结果

若日志包含“WRN”、“WARN”、“ERR”或“ERROR”关键字的告警日志，可收到SMN发送的通知消息邮件，如图4-6所示。同时可以查看OBS桶中的log.txt文件，可查看到具体的告警日志内容，如图4-7所示。

图 4-6 告警消息邮件通知

Get warning message.The content of message is:["{\"ip\": \"192.168.1.98\", \"line_no\": 616, \"host_name\": \"ecs-testagent.novalocal\", \"time\": 1530009653059, \"path\": \"/usr/local/telescope/log/common.log\", \"message\": \"2018-06-26/18:40:53 [WRN] [config.go:82] The projectId or instanceId of config.json is not consistent with metadata, use metadata.\\\\n\", \"log_uid\": \"663d6930-792d-11e8-8b09-286ed488ce70\"}"]

图 4-7 告警日志详情

"\"{\\\"message\\\":\\\"2018-06-26/18:40:53 [WRN] [config.go:82] The projectId or instanceId of config.json is not consistent with metadata, use
metadata.\\\\n\\\",\\\"time\\\":1530009653059,\\\"host_name\\\":\\\"ecs-testagent.novalocal\\\",\\\"ip\\\":\\\"192.168.1.98\\\",\\\"path\\\":
\\\"/usr/local/telescope/log/common.log\\\",\\\"log_uid\\\":\\\"663d6930-792d-11e8-8b09-286ed488ce70\\\",\\\"line_no\\\":616}\""

可以通过函数指标查看函数的调用情况，如图4-8所示。

图 4-8 函数指标

4.6 应用扩展

本案例展示了函数工作流服务+LTS云日志服务实现日志云端处理并推送告警消息的功能，并将告警日志投递至OBS桶中集中存储。函数工作流服务+LTS云日志服务的应用

广泛，如以下应用场景：利用函数的TIMER触发器，定时对存储在OBS桶中的日志数据进行个性化分析和处理。

5 函数+CTS：登录/登出安全分析实战

5.1 案例概述

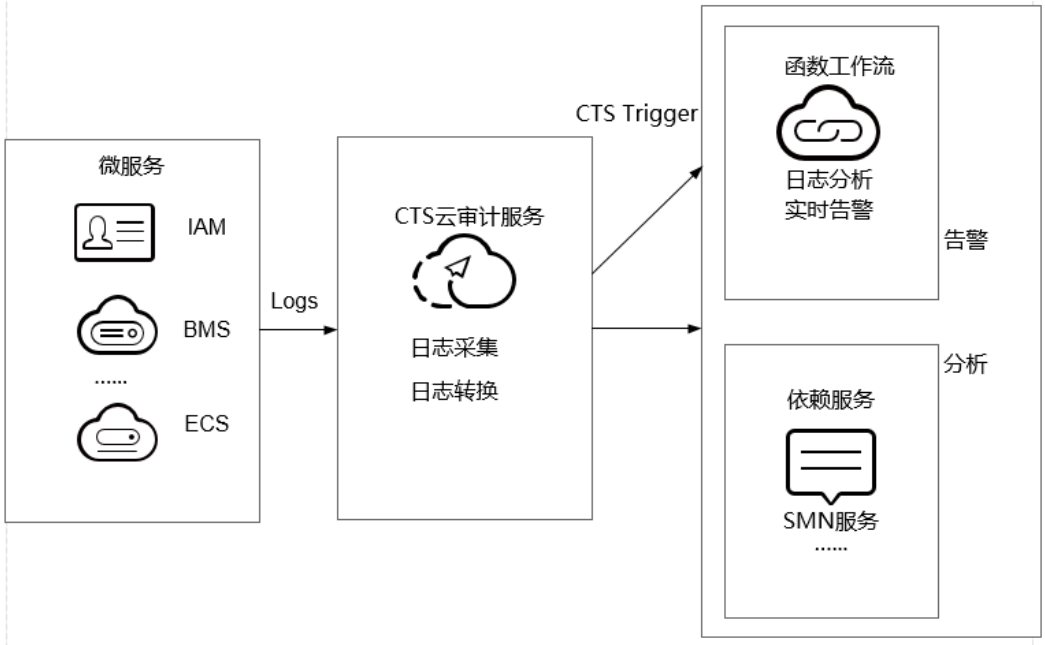
场景介绍

通过CTS云审计服务，完成对公有云账户对各个云服务资源操作动作和结果的实时记录。

通过在函数工作流服务中创建CTS触发器获取订阅的资源操作信息，经由自定义函数对资源操作的信息进行分析和处理，产生告警日志。

SMN消息通知服务通过短信和邮件推送告警信息，通知业务人员进行处理。处理流程如图5-1所示。

图 5-1 处理流程



案例价值点

- 通过CTS云审计服务，快速完成日志分析，对指定IP进行过滤。
- 基于serverless无服务架构的函数计算提供数据加工、分析，事件触发，弹性伸缩，无需运维，按需付费。
- 结合SMN消息通知服务提供日志、告警功能。

5.2 准备

开通 CTS 云审计服务

在云审计服务中开通配置追踪器，如图5-2所示。开通案例参考追踪器配置。

图 5-2 配置追踪器

创建委托

步骤1 登录统一身份认证服务控制台，在左侧导航栏单击“委托”，进入“委托”界面。

步骤2 单击“创建委托”，进入“创建委托”界面。

步骤3 填写委托信息。

- 委托名称：输入“serverless_trust”。
- 委托类型：选择“云服务”。
- 云服务：选择“函数工作流 FunctionGraph”。
- 持续时间：选择“永久”。
- 描述：填写描述信息。

步骤4 单击“下一步”，进入委托选择页面，在“配置权限”界面勾选“Tenant Administrator”，单击“确定”。

说明

Tenant Administrator：拥有该权限的用户可以对企业拥有的所有云资源执行任意操作。

步骤5 单击“确定”，完成权限委托设置。

----结束

告警消息推送

- 在SMN消息通知服务创建主题，此处以主题名称cts_test为例，创建过程请参考创建主题。
- 在SMN消息通知服务订阅主题，用于将告警消息推送至该主题下的订阅终端，此处以添加邮件订阅终端为例，订阅cts_test主题，订阅过程请参考订阅主题。

说明

订阅主题可选择通过邮件、短信、HTTP/HTTPS等形式推送告警消息

本案例中推送告警消息的事件是：当日志事件通过CTS触发器触发函数执行时，函数中过滤白名单告警日志，产生的告警消息推送至SMN主题的订阅终端。

5.3 构建程序

本案例提供了实现告警日志功能的程序包，使用空白模板创建函数，用户可以**下载（index.zip）**学习使用。

创建功能函数

创建实现日志提取功能的函数，将**示例代码**包上传。创建过程请参考**创建函数**，运行时语言选择“Python2.7”，委托名称选择**创建委托**中的“serverless_trust”。

函数实现的功能是：将收到的日志事件数据进行分析，过滤白名单功能，对非法IP登录/登出，进行SMN消息主题邮件告警。形成良好的账户安全监听服务。

设置环境变量

在函数配置页签需配置环境变量，设置SMN主题名称，说明如**表5-1**所示。

表 5-1 环境变量说明表

环境变量	说明
SMN_Topic	SMN主题名称。
RegionName	Region域
IP	白名单

环境变量的设置过程请参考**使用环境变量**。

5.4 添加事件源

选择**准备**中开通的CTS云审计服务，创建CTS触发器，CTS触发器配置如**图5-3**所示。

图 5-3 创建 CTS 触发器

CTS云审计服务监听IAM服务中user资源类型，监听login、logout操作。

5.5 处理结果

若用户触发账号的登录/登出操作，订阅服务类型日志被触发，日志会直接调用用户函数，通过函数代码对当前登录/出的账号进行IP过滤，若不在白名单内，可收到SMN发送的通知消息邮件，如图5-4所示。

图 5-4 告警消息邮件通知

```
Illegal operation[  IP:10.65.56.139,  Action:login]
-------
```

邮件信息中包含非法请求ip地址和用户执行的动作（login/logout）。

可以通过函数指标查看函数的调用情况，如图5-5所示。

图 5-5 函数指标

6 定时开关华为公有云虚拟机

应用场景

当您需要在特定时间打开或者关闭华为公有云虚拟机时，可以考虑通过函数服务调用华为云ECS接口，定时开关虚拟机。

- 开机节点：需要定时打开的虚拟机。
- 关机节点：需要定时关闭的虚拟机。

前提条件

1. 根据实际业务，获取**定时开启华为公有云虚拟机的程序包**或者**定时关闭华为公有云虚拟机的程序包**。
2. 创建委托EcsOperation，添加“ECS FullAccess”权限，请参考**创建委托**。

创建委托

1. 登录**统一身份认证服务控制台**。
2. 在统一身份认证服务的左侧导航窗格中，选择“委托”页签，单击右上方的“+创建委托”。

图 6-1 创建委托

3. 开始配置委托。
   - 委托名称：EcsOperation。
   - 委托类型：选择“云服务”。
   - 云服务：选择“函数工作流 FunctionGraph”。
   - 持续时间：选择“永久”。
   - 描述：填写描述信息。

4. 单击“下一步”，进入委托选择页面，在右方搜索框中搜索“ECS FullAccess”权限并勾选。

图 6-2 选择权限

5. 单击“下一步”，请根据业务需要选择权限的作用范围。

构建程序

步骤1 创建功能函数。

创建定时开启或者关闭华为公有云虚拟机的函数，上传**定时开启华为公有云虚拟机的程序包**或者**定时关闭华为公有云虚拟机的程序包**，并选择创建的委托EcsOperation。创建过程请参考**创建函数**。

运行时语言选择“Python3.6”，委托名称选择上一步创建的委托“EcsOperation”。

步骤2 设置环境变量。

在“配置”页签配置环境变量，说明如**表6-1**所示。

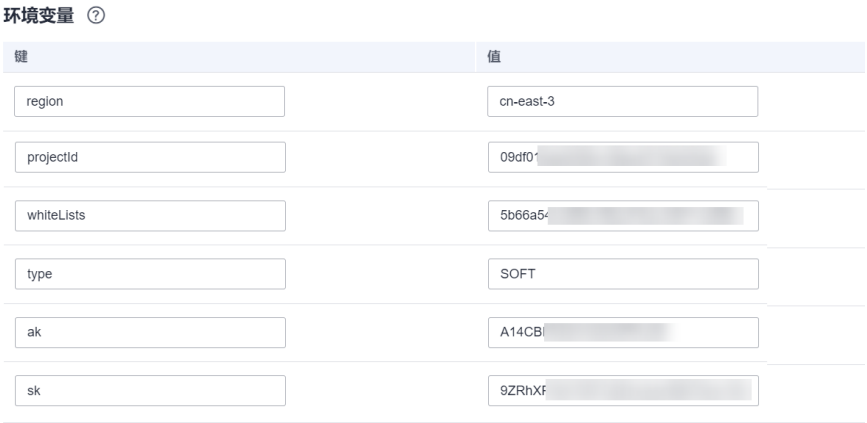
表 6-1 环境变量说明

环境变量	说明
region	ECS所在的区域，如ap-southeast-1
projectId	ECS所在的Project ID，获取方法请参见**获取项目ID**。
whiteLists	• 当定时开启华为公有云虚拟机时，填写**需开启**的虚拟机ID，**以英文逗号分隔** • 当定时关闭华为公有云虚拟机时，填写**需关机**的虚拟机ID，**以英文逗号分隔**
type	仅需在定时关机时确认是否需要配置。 关机类型： SOFT：普通关机（默认） HARD：强制关机

环境变量的设置过程请参考**使用环境变量**。

说明

- 本案例对函数执行的区域没有要求，若函数和待开关机节点在同一region，按照上述操作即可。若函数和待开关机节点不在同一region，如函数运行在中国-香港，想要开启或者关闭亚太-曼谷的弹性云服务的虚拟机，只需要将projectId、region更改为亚太-曼谷区域的信息，并在环境变量中添加ak、sk（**获取AK/SK**），再去掉配置的委托即可。
  - AK/SK认证就是使用AK/SK对请求进行签名，在请求时将签名信息添加到消息头，从而通过身份认证。
  - AK（Access Key ID）：访问密钥ID。与私有访问密钥关联的唯一标识符；访问密钥ID和私有访问密钥一起使用，对请求进行加密签名。
  - SK（Secret Access Key）：与访问密钥ID结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。



- 如果开启或者关闭的虚拟机数量过多，则需要增大超时时间。
- **表6-1**中除whiteLists以外的环境变量必须添加，whiteLists根据实际情况选择添加或者不添加，whiteLists为需开机/关机的ecs服务器ID，以逗号分隔。
- {region}.{domain}组成ECS的终端节点Endpoint，如：ap-southeast-1.myhuaweicloud.com，具体Endpoint信息，请参考**地区和终端节点**。

步骤3 选择依赖包。

在“代码”页签，添加“huaweicloudsdk_ecs_core_py3.6”依赖包。

添加依赖包详细操作请参见**配置函数依赖**。

说明

如果您所在区域无法添加“huaweicloudsdk_ecs_core_py3.6”依赖包，请联系客服具体咨询。

----结束

添加事件源

创建TIMER触发器，TIMER触发器配置如**图6-3**所示。

图 6-3 创建 TIMER 触发器

7 使用 SpringBoot 构建 FunctionGraph HTTP 函数

方案概述

本章节主要指导使用Springboot开发应用的用户，部署业务到FunctionGraph。

用户通常可以使用**SpringInitializr**或者IntelliJ IDEA新建等多种方式构建SpringBoot，本章节以Spring.io 的**https://spring.io/guides/gs/rest-service/** 项目为例，使用HTTP函数的方式部署到FunctionGraph上。

操作流程

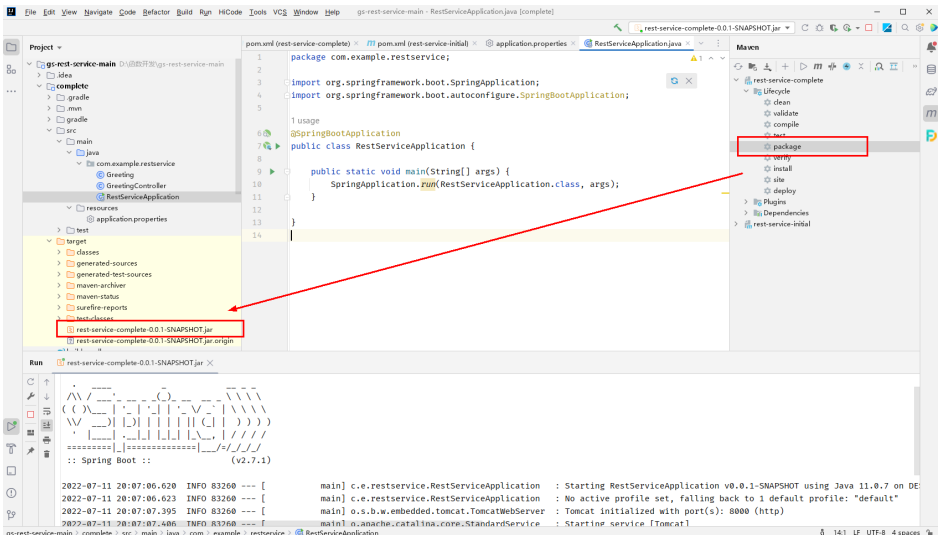
将既有项目部署到FunctionGraph通常只需要：修改项目监听端口号为8000，然后在jar包同目录创建bootstrap文件写入执行jar包的命令。

本案例使用IntelliJ IDEA，Maven项目。

构建代码包

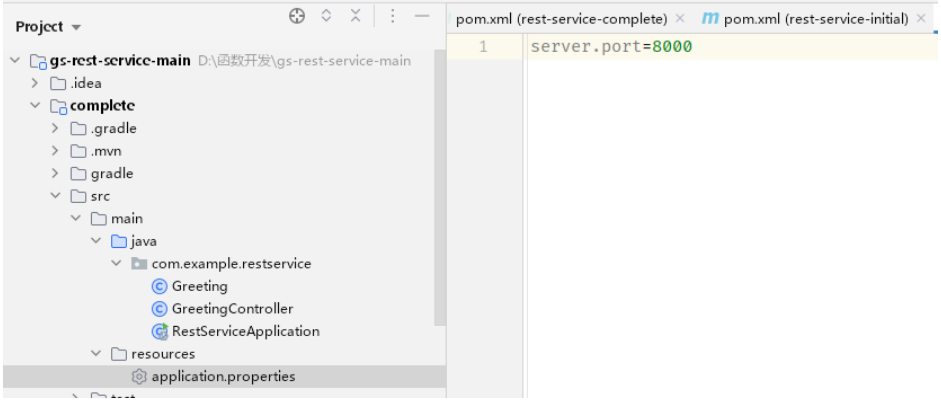
1. 打开Springboot项目，在maven插件处单击package，生成jar包。

图 7-1 生成 jar 包



2. 配置工程web端口。HTTP函数当前支持8000端口，需配置工程web端口为 8000（**此端口请勿修改**），可以使用application.properties文件来配置，也可以在启动时指定端口号。

图 7-2 配置 8000 端口



3. 在jar包同目录创建bootstrap文件，输入启动参数。

```
/opt/function/runtime/java11/rtsp/jre/bin/java -jar -Dfile.encoding=utf-8 /opt/function/code/rest-service-complete-0.0.1-SNAPSHOT.jar
```

4. 选中jar包和bootstrap文件，打包成zip包。

创建HTTP函数并上传代码

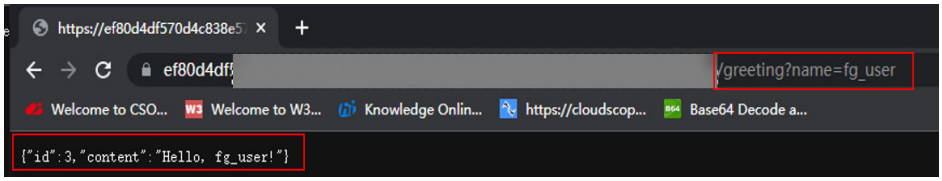
创建1个HTTP函数，并上传已打包的zip包。请参见**创建HTTP函数**。

验证结果

- 使用函数测试事件验证
  a. 在函数详情页，选择函数版本，单击“配置测试事件”，弹出“配置测试事件”页。
  b. 选择事件模板“apig-event-template”，修改测试事件中的path、pathParameters参数，构建一个简单的Get请求。
  c. 单击“创建”，完成测试事件创建。
  d. 单击“测试”，获取响应。

  建议在测试时函数内存规格、超时时间调大，如512MB、5s。
- 配置APIG触发器测试
  a. 请参见**使用APIG触发器**，创建APIG触发器，“安全认证”建议选择“None”，方便调试。
  b. 复制生成的调用URL在浏览器进行访问。如**图 调用函数**所示，在URL后添加请求参数greeting?name=fg_user，响应如下。

图 7-3 调用函数



默认生成的APIG触发器的调用URL为“域名/函数名”，在本案例中即：https://your_host.com/springboot_demo，URL中包含了函数名

springboot_demo作为path的第一部分。如果直接Get https://your_host.com/springboot_demo/greeting，springboot接收到的请求地址将包含springboot_demo/greeting两部分。此处需注意： 如果用户直接把已有的工程上传，会因为path里多了函数名而无法直接访问自己的服务。因此，请参考以下两种方法注解或去除函数名。

- 方法一：修改代码中的Mapping地址，例如在GetMapping注解或者类注解上添加默认的path第一部分。

  图 7-4 修改 Mapping 地址

```
@RestController
public class GreetingController {

    1 usage
    private static final String template = "Hello, %s!";
    1 usage
    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/springboot_demo/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template, name));
    }
}
```

- 方法二：单击触发器名称，跳转至API网关服务，直接修改path去除函数名。

常见问题

1. 我的代码可以访问哪些目录?

   根据上文中的bootstrap文件里的命令，可以看出上传的代码包最终被存在函数实例（指函数运行的环境/计算资源，可以理解为容器）/opt/function/code/ 路径。但是该目录只可以读，不可以写入。 如果您希望在代码运行期间写入一些数据到实例里，打印日志到本地，或者您使用的依赖默认写入jar所在的目录，请对/tmp目录进行写入操作。

2. 我的日志如何被收集，应该怎么输出日志?

   函数实例在一段时间内没有请求会被销毁，写入到本地日志会同时被销毁，当前用户也无法在函数运行中查看函数本地日志，所以建议不要仅将日志写入到本地。产生的日志建议输出到控制台，如配置log4j输出target为System.out，或直接用print函数打印日志等。

   输出到控制台的日志，会被函数系统收集，如果用户开通LTS服务，日志会被放入LTS 可以进行较为实时的日志分析。

   调测建议：建议在调测时候**开通LTS日志**，单击“到LTS进行日志分析”，在实时日志中进行观察分析。

3. 我的代码具有什么用户的执行权限?

   和普通事件函数一样，代码执行时并没有root权限，因此需要root权限的代码或者命令都无法在HTTP函数里执行。

4. 如何为多个模块的springboot项目进行打包配置?

   您需要为多个模块的springboot项目设置以下打包配置：

```
<build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <mainClass>com.example.YourServiceMainClass</mainClass>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

8 创建使用自定义认证且后端为FunctionGraph 的 API

8.1 方案概述

在API的安全认证方面，API网关提供IAM认证、APP认证等方式，帮助用户快速开放API，同时API网关也支持用户使用自己的认证方式（以下简称自定义认证），以便更好地兼容已有业务能力。

本手册基于函数工作流服务实践编写，指导您快速创建后端服务为FunctionGraph的API，并通过APIG安全认证中的“自定义认证”鉴权方式进行调用。

解决方案

- 登录FunctionGraph控制台，创建函数，并将其定义为自定义认证函数。
- 登录FunctionGraph控制台，创建一个业务函数。
- 在APIG中创建一个API分组，用来存放API。
- 创建一个鉴权方式为自定义认证且后端为FunctionGraph的API。
- 调试API。

说明

完成本教程后，您的公有云账户将存在以下资源：

1. 一个API分组（存放API）。
2. 一个自定义认证函数。
3. 一个业务函数。
4. 一个鉴权方式为自定义认证且后端为FunctionGraph的API。

8.2 资源规划

请保证以下资源在同一区域。

表 8-1 资源规划

资源	数量（个）
API分组	1
自定义认证函数	1
业务函数	1
API	1

8.3 构建程序

创建 API 分组

创建函数及添加事件源之前，需要先创建一个API分组，API分组是API的管理单元，用来存放API。

说明

您需要拥有一个APIG实例后才能开启API网关服务相关功能，具体请参见购买实例。

步骤1 登录APIG控制台，在左侧导航栏选择“API管理 > API分组”，单击“创建API分组”。

步骤2 选择直接创建，设置以下分组信息，完成后单击“确定”创建分组。

- 分组名称：输入您自定义的分组名称，例如APIGroup_test。
- 描述：输入对分组的描述。

----结束

创建自定义认证函数

前端自定义认证指APIG利用校验函数对收到的API请求进行安全认证，如果您想要使用自己的认证系统对API的访问进行认证鉴权，您可以在API管理中创建一个前端自定义认证来实现此功能。您需要先在FunctionGraph创建一个函数，通过函数定义您所需的认证信息，函数创建成功后，即可对API网关中的API进行认证鉴权。

本示例以Header中的请求参数：event["headers"]，为例进行演示。请求参数详细说明请参见请求参数代码定义示例。

步骤1 在服务控制台左侧导航栏，选择“计算 > 函数工作流”，进入函数工作流控制台后，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。

步骤2 单击“创建函数”，进入创建函数流程。

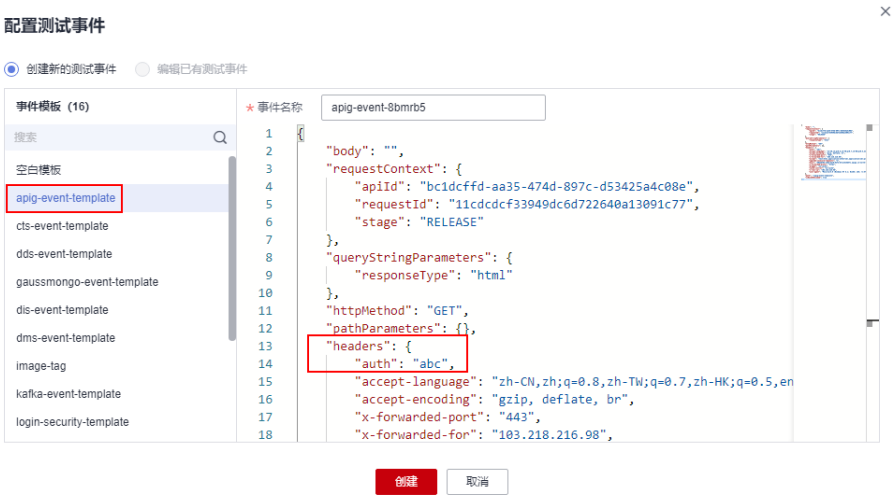
步骤3 填写函数配置信息，完成后单击“创建函数”。

- 模板：选择“使用空白模板”。
- 函数名称：输入您自定义的函数名称，例如：apig-test。
- 委托名称：选择“未使用任何委托”。
- 运行时语言：选择“Python 2.7”。

步骤4 进入函数详情页，在“代码”页签，进行代码在线编辑，复制**Header中的请求参数定义代码示例**中的代码并单击“部署”，更新函数。

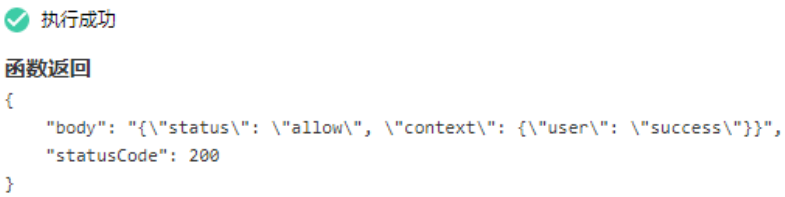
步骤5 配置测试事件，测试用于前端自定义认证的函数。单击“配置测试事件”，选择“apig-event-template”。根据实际情况修改后保存测试模板(本示例在"headers"中添加"auth":"abc"），完成后单击“创建”。

图 8-1 配置测试事件



步骤6 单击“测试”，执行结果为“成功”时，表示自定义认证函数创建成功。

图 8-2 查看执行结果



----结束

创建自定义认证

在APIG中创建自定义认证，对接前端自定义认证的函数。

步骤1 在服务控制台左侧导航栏，选择“应用中间件 > API网关”登录APIG控制台，在左侧导航栏选择“API管理 > API策略”，在“自定义认证”页签下，单击“创建自定义认证”，弹出“创建自定义认证”对话框。

步骤2 配置自定义认证基础信息，如下图所示。

- 认证名称：输入您自定义的名称，例如Authorizer_test。
- 类型：选择“前端”。
- 函数地址：请选择用于前端自定义认证的函数apig-test。

图 8-3 创建自定义认证

步骤3 完成后单击“确定”，完成自定义认证的创建。

----结束

创建后端业务函数

API网关（APIG）支持选择FunctionGraph作为后端服务类型，当请求设置函数工作流为后端服务的API时，API网关会触发相应的函数，函数工作流会将执行结果返回给API网关（APIG）。

步骤1 创建函数方法与上述**创建自定义认证函数**相同，只需修改函数名称，避免名称重复。

步骤2 在函数详情页的“代码”页签，进行代码在线编辑，并传入如下所示的代码，完成后单击“部署”，更新函数。

```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    body = "<html><title>Functiongraph Demo</title><body><p>Hello, FunctionGraph!</p></body></html>"
    print(body)
    return {
        "statusCode":200,
        "body":body,
        "headers": {
            "Content-Type": "text/html",
        },
```

```
    "isBase64Encoded": False
}
```

----结束

请求参数代码定义示例

在FunctionGraph中开发函数，以python2.7语言为例，函数代码需要满足如下条件。

函数有明确的接口定义，如下所示：

def handler (event, context)

- 入口函数名（handler）：入口函数名称，需和函数执行入口处用户自定义的入口函数名称一致。
- 执行事件（event）： 函数执行界面由用户输入的执行事件参数， 格式为JSON对象。
- 上下文环境（Context）：Runtime提供的函数执行上下文，其接口定义在**SDK接口**说明。

执行事件（event）支持三种请求参数定义，格式为：

- Header中的请求参数：event["headers"]["参数名"]
- Query中的请求参数：event["queryStringParameters"]["参数名"]
- 您自定义的用户数据：event["user_data"]

函数代码获取的三种请求参数与API网关自定义认证中的参数关系如下所示：

- Header中的请求参数：对应自定义认证中参数位置为Header的身份来源，其参数值在您调用使用该前端自定义认证的API时传入
- Query中的请求参数：对应自定义认证中参数位置为Query的身份来源，其参数值在您调用使用该前端自定义认证的API时传入
- 您自定义的用户数据：对应自定义认证中的用户数据，其参数值在您创建自定义认证时输入
- 函数的返回值不能大于1M，必须满足如下格式：

```
{    "statusCode":200,
     "body": "{\"status\": \"allow\", \"context\": {\"user\": \"abc\"}}"
}
```

其中，body字段的内容为字符串格式，json解码之后为：

```
{
   "status": "allow/deny",
   "context": {
      "user": "abc"
   }
}
```

“status”字段为必选，用于标识认证结果。只支持“allow”或“deny”，“allow”表示认证成功，“deny”表示认证失败。

“context”字段为可选，只支持字符串类型键值对，键值不支持JSON对象或数组。

context中的数据为您自定义的字段，认证通过后作为认证参数映射到API网关后端参数中，其中context中的参数名称与系统参数名称必须完全一致，且区分大小写，context中的参数名称必须以英文字母开头，支持英文大小写字母、数字、下划线和中划线，且长度为1 ~ 32个字符。

Header中的请求参数定义代码示例：

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event["headers"].get("auth")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"success"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

Query中的请求参数定义代码示例：

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event["queryStringParameters"].get("test")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"abcd"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

用户数据定义代码示例：

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event.get("user_data")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"abcd"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

8.4 添加事件源

创建 API

API分组、自定义认证函数、后端函数均创建成功以后，可以创建API，设置安全认证为自定义认证，并定义后端服务类型为FunctionGraph，步骤如下。

步骤1 登录APIG控制台，在左侧导航栏选择“API管理 > API列表”，单击右上方的“创建API”。

步骤2 配置API基本信息，详细如图8-4、图8-5所示。

- API名称：输入您自定义的名称，例如API_test。
- 所属分组：请选择上述操作中创建的API分组“APIGroup_test”。
- URL：请求方法选择“ANY”，请求协议选择“HTTPS”，请求路径填写“/testAPI”。
- 网关响应：选择“default”。
- 安全认证：选择“自定义认证”。
- 自定义认证：选择上述操作中创建的自定义认证“Authorizer_test”。

图 8-4 前端定义配置

图 8-5 安全配置

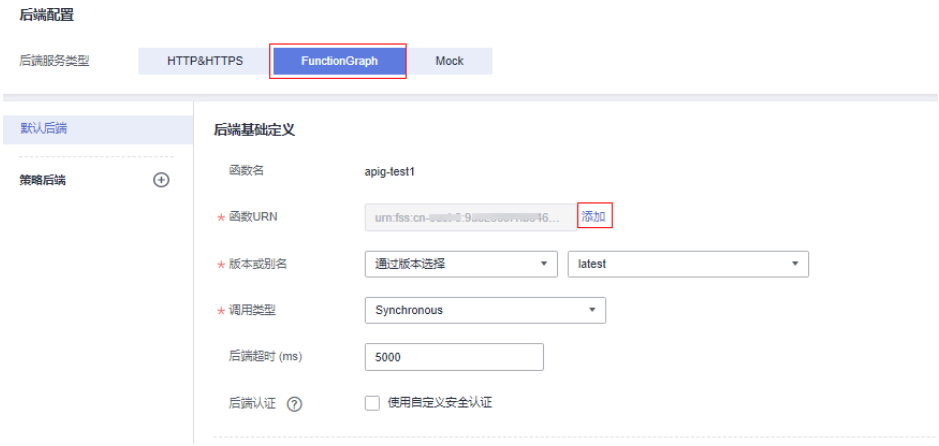
说明

更多API配置项详细描述，请参见**创建API**。

步骤3 单击“下一步”，进行后端配置，详细如**图8-6**所示。

- 后端服务类型：选择“FunctionGraph”
- 函数URN：添加创建的业务函数
- 版本或别名：选择“latest”版本
- 调用类型：选择“Synchronous”

图 8-6 后端服务配置



步骤4 单击下一步，完成API创建。

步骤5 继续在当前页面，单击“发布”，将已创建的API发布至RELEASE环境。

图 8-7 发布 API

----结束

8.5 调试并调用 API

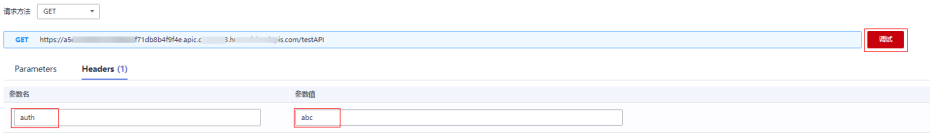
API网关提供了在线调试的功能，因此一般建议在API网关上完成API配置之后，可以先通过此功能确认API是否配置成功。

步骤1 登录APIG控制台，左侧导航栏选择“API管理 > API列表”，单击进入已创建的API“API_test”，单击“调试”。

步骤2 在本案例中，需要添加Headers参数，完成后单击“调试”。

- 参数名：输入“auth”
- 参数值：输入“abc”

图 8-8 添加 Headers 参数



步骤3 API返回内容即为前面步骤中创建的业务函数返回内容。如图8-9。

图 8-9 API 返回内容

```
HTTP/1.1 200 OK
Content-Length: 87
Connection: keep-alive
Content-Type: text/html; charset=UTF-8
Date: Tue, 07 Feb 2023 06:39:18 GMT
Server: api-gateway
Strict-Transport-Security: max-age=31536000; includeSubdomains;
X-Apig-Latency: 2140
X-Apig-Ratelimit-Api: remain:99,limit:100,time:1 minute
X-Apig-Ratelimit-Api: remain:15601,limit:16000,time:1 second
X-Apig-Ratelimit-Api-Allenv: remain:5999,limit:6000,time:1 second
X-Apig-Ratelimit-Api-Allenv: remain:15601,limit:16000,time:1 second
X-Apig-Ratelimit-User: remain:9870,limit:10000,time:1 second
X-Apig-Upstream-Latency: 1539
X-Cff-Billing-Duration: 5
X-Cff-Invoke-Summary: {"funcDigest":"64999f78efbc98714f57b3f190573be","duration":4.952,"billingDuration":5,"memo
rySize":128,"memoryUsed":25.906,"podName":"pool22-300-128-fusion-67fc9b8d95-s6rsv"}
X-Cff-Request-Id: 495bcdf5-d474-4aa5-ba04-c79f84d4367c
X-Content-Type-Options: nosniff
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Request-Id: dfa7d5925751f31f12221f45459a1312
X-Xss-Protection: 1; mode=block;


<html><title>Functiongraph Demo</title><body><p>Hello, FunctionGraph!</p></body></html>
```

----结束

9 函数+APIG：处理文件上传

9.1 方案概述

应用场景

端侧文件上传云服务器是Web和App应用的一类场景，例如服务运行日志的上报，Web应用图片上传等，函数可作为后端，结合APIG提供通用的API处理这类场景。本章节以NodeJS和Python语言为例，指导用户如何开发后端解析函数，获取上传的文件。

约束与限制

- 单次请求上传文件大小不超过6MB。
- 函数逻辑处理时间不超过15分钟。

9.2 资源规划

表 9-1 资源规划

产品	配置示例
API网关APIG	● 区域：新加坡 ● 规格：可使用共享版APIG或者创建专项版APIG实例
函数工作流FunctionGraph	● 区域：新加坡 ● 计费模式：按需计费

9.3 操作流程

本方案包含以下操作步骤

1. 创建文件接收函数：接收上传的文件并解析内容。
2. 端到端测试：绑定APIG触发器，测试文件上传及处理流程。

9.3.1 NodeJS 语言方案

前提条件

- 已拥有华为云账号且已实名认证。
- 华为云账号未欠费，且有足够金额购买本案例所涉及的资源。

操作步骤

步骤1 创建函数

1. 登录函数工作流控制台，在左侧导航栏选择“函数 > 函数列表”，单击“创建函数”。
2. 选择创建“空白函数”，填写函数信息，完成后单击“创建函数”。
   - 函数类型：事件函数
   - 区域：亚太-新加坡
   - 函数名称：upload-file-1
   - 委托名称：未使用任何委托
   - 运行时：Node.js 14.18
3. 在“代码”页签，复制如下代码替换默认的函数代码，并单击“部署”更新函数。

```
const stream = require("stream");
const Busboy = require("busboy");

exports.handler = async (event, context) => {
    const logger = context.getLogger()
    logger.info("Function start run.");
    if (!("content-type" in event.headers) ||
        !event.headers["content-type"].includes("multipart/form-data")) {
        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json'
            },
            'body': 'The request is not in multipart/form-data format.',
        };
    }

    const busboy = Busboy({ headers: event.headers });
    let buf = Buffer.alloc(0);
    busboy.on('file', function (fieldname, file, filename, encoding, mimetype) {
        logger.info('filename:' + JSON.stringify(filename))
        file.on('data', function (data) {
            logger.info('Obtains ' + data.length + ' bytes of data.')
            buf = Buffer.concat([buf, data]);
        });
        file.on('end', function () {
            logger.info('End data reception');
        });
    });

    busboy.on('finish', function () {
        //这里处理数据
        logger.info(buf.toString());
        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json'
            },
            'body': 'ok',
```

```
        };
    });

    //APIG触发器默认对数据进行Base64编码，这里解码
    const body = Buffer.from(event.body, "base64");
    var bodyStream = new stream.PassThrough();
    bodyStream.end(body);
    bodyStream.pipe(busboy);
}
```

步骤2 配置函数依赖

1. 制作依赖包。代码中选择busboy库解析上传的文件，需要生成Node.js14.18版本对应的依赖包busboy.zip。如果您使用Node.js语言其他版本，请制作对应版本的依赖包，具体请参考**制作依赖包**。
2. 创建依赖包。在左侧导航栏“函数 > 依赖包”管理页面，单击“创建依赖包”，配置完成后单击“确定”。
   - 依赖包名称：busboy
   - 代码上传方式：上传ZIP文件
   - 运行时：Node.js 14.18
   - 文件上传：添加制作完成的依赖包
3. 添加依赖包。进入upload-file-1函数详情页面，在“代码”页签最底部，单击“添加依赖包”。在“私有依赖包”的包源中，选择上一步创建的busboy依赖包，单击“确定”，完成依赖包的添加。

步骤3 配置APIG触发器

1. 在upload-file-1函数详情页面，单击“设置 > 触发器”，开始创建触发器。
2. 单击“创建触发器”，触发器类型可以选择“API 网关服务(APIG)”或“API 网关服务(APIG 专享版本)”，此处以共享版APIG为例。
   - API名称：默认即可，无需修改。
   - 分组：选择在APIG创建的API分组，若无分组，可单击“创建分组”跳转至APIG创建。
   - 发布环境：RELEASE。
   - 安全认证：此处为方便测试，配置“None”，实际业务请选择更安全的认证方式，例如IAM认证等。
   - 请求协议：选择“HTTPS”。
   - 后端超时（毫秒）：默认5000毫秒。

步骤4 端到端测试

以curl工具为例（curl -F的方式主要用的是linux环境），您也可以选择postman等其他工具，在本地创建app.log文件，内容自定义，此处简单举例：

```
start something
run
stop all
```

执行如下命令测试：

```
curl -iv {APIG触发器URL} -F upload=@/{本地文件路径}/app.log
```

图 9-1 示例

在upload-file-1函数详情页面的“监控”页签下，查看日志，可看到文件内容的打印。实际业务中，用户可根据需要修改代码保存数据到对象存储OBS、日志服务LTS等云服务或直接处理。

----结束

9.3.2 Python 语言方案

前提条件

- 已拥有华为云账号且已实名认证。
- 华为云账号未欠费，且有足够金额购买本案例所涉及的资源。

操作步骤

步骤1 创建函数

1. 登录函数工作流控制台，在左侧导航栏选择“函数 > 函数列表”，单击“创建函数”。
2. 选择创建“空白函数”，填写函数信息，完成后单击“创建函数”。
   - 函数类型：事件函数
   - 区域：亚太-新加坡
   - 函数名称：upload-file-2
   - 委托名称：未使用任何委托
   - 运行时：Python 3.6
3. 在“代码”页签，复制如下代码替换默认的函数代码，并单击“部署”更新函数。

```
# -*- coding: utf-8 -*-

from requests_toolbelt.multipart import decoder
import base64

def handler(event, context):
    context.getLogger().info("Function start run.")

    content_type = ''
    if "content-type" in event['headers']:
        content_type = event['headers']['content-type']

    if "multipart/form-data" not in content_type:
        return {
            "statusCode": 200,
            "body": "The request is not in multipart/form-data format.",
            "headers": {
                "Content-Type": "application/json"
            }
        }

    body = event['body']
    #APIG触发器默认对数据进行Base64编码，这里解码
    raw_data = base64.b64decode(body)
    for part in decoder.MultipartDecoder(raw_data, content_type).parts:
        #这里处理数据
        context.getLogger().info(part.content)

    return {
        "statusCode": 200,
        "body": "ok",
        "headers": {
```

```
        "Content-Type": "application/json"
    }
}
```

步骤2 配置APIG触发器

1. 在upload-file-1函数详情页面，单击“设置 > 触发器”，开始创建触发器。
2. 单击“创建触发器”，触发器类型可以选择“API 网关服务(APIG)”或“API 网关服务(APIG 专享版本)”，此处以共享版APIG为例。
   - API名称：默认即可，无需修改。
   - 分组：选择在APIG创建的API分组，若无分组，可单击“创建分组”跳转至APIG创建。
   - 发布环境：RELEASE。
   - 安全认证：此处为方便测试，配置“None”，实际业务请选择更安全的认证方式，例如IAM认证等。
   - 请求协议：选择“HTTPS”。
   - 后端超时（毫秒）：默认5000毫秒。

步骤3 端到端测试

以curl工具为例（curl -F的方式主要用的是linux环境），您也可以选择postman等其他工具。在本地创建app.log文件，内容自定义，此处简单举例：

```
start something
run
stop all
```

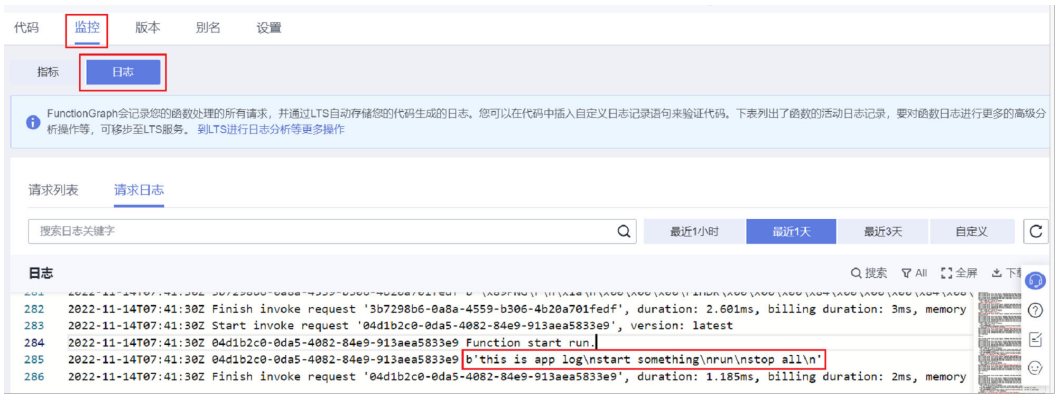
执行如下命令测试：

```
curl -iv {APIG触发器URL} -F upload=@/{本地文件路径}/app.log
```

图 9-2 示例

在upload-file-2函数详情页面的“监控”页签下，查看日志，可看到文件内容的打印。实际业务中，用户可根据需要修改代码保存数据到对象存储OBS、日志服务LTS等云服务或直接处理。

图 9-3 查看日志



----结束

10 使用函数处理 IOT 数据

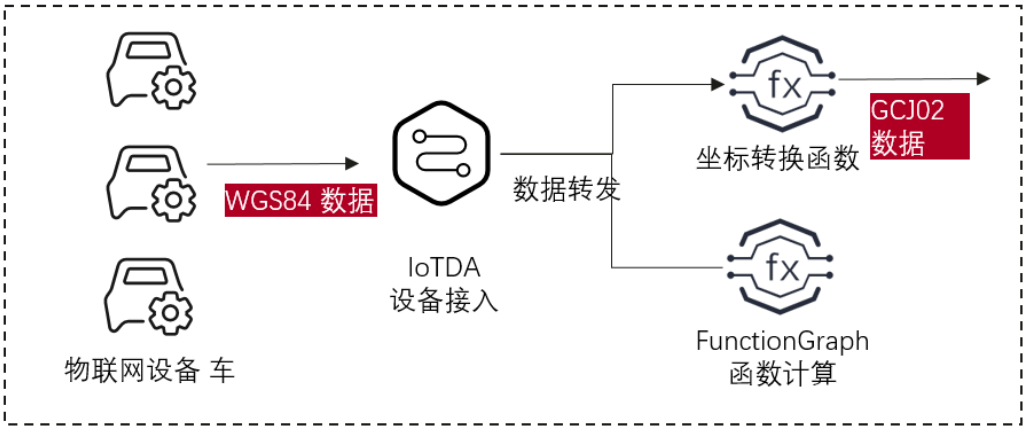
10.1 案例概述

场景介绍

该案例演示客户如何使用FunctionGraph 与IoTDA 服务组合，处理物联网设备上报以及设备状态变动的相关数据。物联网设备在IoTDA 平台进行管理，设备产生的数据可以从IoTDA直接流转触发FunctionGraph 的函数运行。用户可以根据需要编写函数处理这些数据。

通常该组合，可以适用于以下场景，如将设备上报的数据在处理后进行存储到如OBS；对上报的数据进行结构化，清洗然后存储到数据库；根据设备状态变化进行事件通知等。

该案例重点在如何组合IoTDA 与 FunctionGraph，关于如何在IoTDA 以及设备上进行设备管理和数据上报，需要用户进一步参考IoTDA的文档。在该案例中，我们使用IoTDA + FunctionGraph 做一个坐标转换的示例（WGS84 坐标转 GCJ02坐标）。



实现流程

- 在IoTDA创建IoTDA实例（测试时可以创建标准版免费体验）。
- 在FunctionGraph创建函数。

- 在IoTDA设置转发规则或者在FunctionGraph创建IoT触发器。
- 在IoTDA转发规则发送测试消息。

10.2 准备

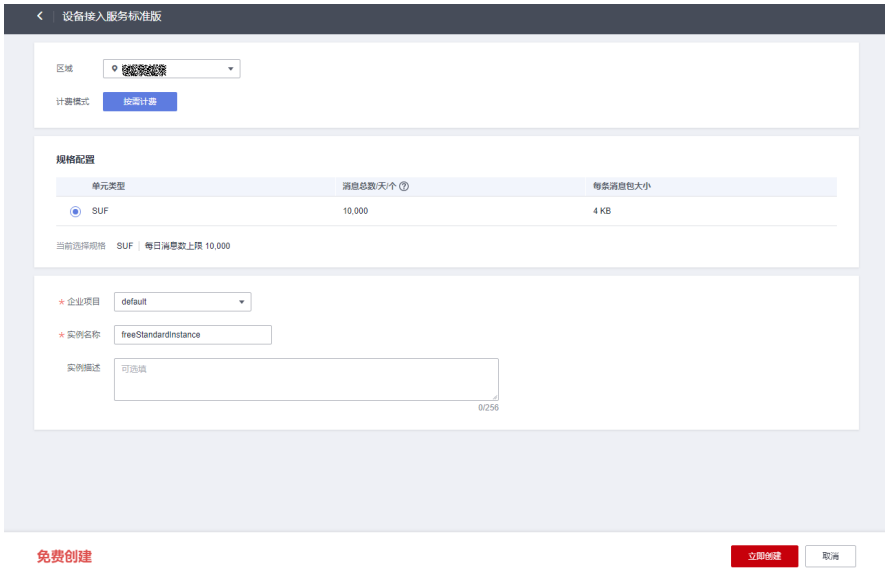
创建IoTDA 转发规则前，需要先创建IoTDA实例，在正常的使用中还需要创建产品，设备。在本案例中我们只测试，只需要先创建IoTDA 实例。

创建 IoTDA IoT 实例

步骤1 登录IoTDA控制台，左侧导航栏选择“IoTDA实例”，进入选择界面。

步骤2 在“IoTDA实例”界面右侧，单击“开通免费单元 ”，进入参数配置界面，请您根据实际业务需求进行配置。

图 10-1 开通免费单元



步骤3 参数配置完成后，单击“立即创建”，完成IoTDA实例创建。

----结束

创建函数

步骤1 在服务控制台左侧导航栏，选择“计算 > 函数工作流”进入函数工作流控制台，单击“创建函数”。

步骤2 输入函数名称 iotdemo，选择熟悉的运行时，案例这里使用Python 3.9 ，然后点击创建。

----结束

创建转发规则

转发规则用于数据从IoTDA流转到指定函数，以触发函数运行，可以在IoTDA 页面创建转发规则，也可以在FunctionGraph 创建 IoT触发器来实现。下面说明在IoTDA 页面创建转发规则。

步骤1 在服务控制台左侧导航栏，选择“IoT物联网 > 设备接入”进入IoTDA控制台，单击左侧边栏 “规则”，然后单击“数据转发”，单击“创建规则”。

步骤2 输入基本信息，然后单击创建规则。

说明

- 规则名称 iotfg 也可自定义
- 数据来源选择 设备消息
- 触发事件选择 设备消息上报

步骤3 设置转发目标，单击“添加” ，转发目标选择 FunctionGraph 。

步骤4 首次使用需要授权IotDA访问FunctionGraph函数，单击“授权”即可。

步骤5 选择刚创建的函数iotdemo 。

步骤6 单击启动规则。

----结束

10.3 构建函数程序

编辑函数程序

打开创建的函数iotdemo，复制以下坐标转换代码，仅供测试不建议用于生产用途，用户也可以根据自己的需要修改。

```
# -*- coding:utf-8 -*-
import json
import math
from math import pi

def handler(event, context):
    data = event["notify_data"]["body"]
    lat = data["lat"]
    lng = data["lng"]
    print(f" WGS84: ({lng},{lat})")
    gcj_lng, gcj_lat = transform(lng, lat)
    print(f" GCJ02: ({gcj_lng},{gcj_lat})")
    body = {
        "gcj_lng": gcj_lng,
        "gcj_lat": gcj_lat
    }
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(body),
        "headers": {
            "Content-Type": "application/json"
        }
    }

def transform(lon, lat):
    a = 6378245.0
    ee = 0.00669342162296594323

    dlat = transform_lat(lon - 105.0, lat - 35.0)
    dlon = transform_lon(lon - 105.0, lat - 35.0)

    rad_lat = lat / 180.0 * pi
    magic = math.sin(rad_lat)
    magic = 1 - ee * magic * magic
    sqrt_magic = math.sqrt(magic)

    dlat = (dlat * 180.0) / ((a * (1 - ee)) / (magic * sqrt_magic) * pi)
    dlon = (dlon * 180.0) / (a / sqrt_magic * math.cos(rad_lat) * pi)

    mg_lon = lon + dlon
    mg_lat = lat + dlat

    return mg_lon, mg_lat

def transform_lon(x, y):
    ret = 300.0 + x + 2.0 * y + 0.1 * x * x + \
        0.1 * x * y + 0.1 * math.sqrt(math.fabs(x))
    ret += (20.0 * math.sin(6.0 * pi * x) +
            20.0 * math.sin(2.0 * pi * x)) * 2.0 / 3.0
    ret += (20.0 * math.sin(pi * x) +
            40.0 * math.sin(pi / 3.0 * x)) * 2.0 / 3.0
    ret += (150.0 * math.sin(pi / 12.0 * x) +
            300.0 * math.sin(pi / 30.0 * x)) * 2.0 / 3.0
return ret

def transform_lat(x, y):
    ret = -100.0 + 2.0 * x + 3.0 * y + 0.2 * y * y + \
```

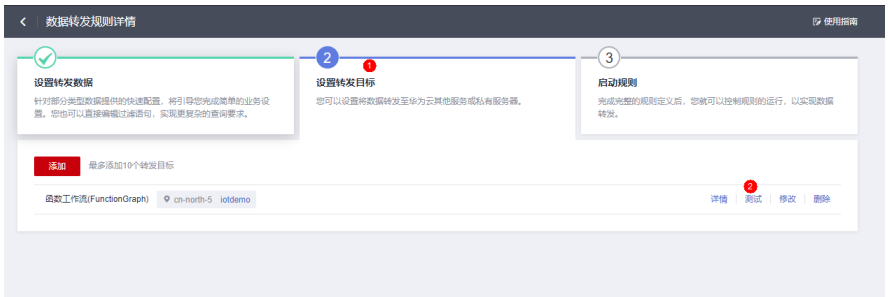
```
    0.1 * x * y + 0.2 * math.sqrt(math.fabs(x))
ret += (20.0 * math.sin(6.0 * pi * x) +
        20.0 * math.sin(2.0 * pi * x)) * 2.0 / 3.0
ret += (20.0 * math.sin(pi * y) +
        40.0 * math.sin(pi / 3.0 * y)) * 2.0 / 3.0
ret += (160.0 * math.sin(pi / 12.0 * y) +
        320 * math.sin(pi / 30.0 * y)) * 2.0 / 3.0
return ret
```

通过 IoTDA 进行线上联调测试

步骤1 登录IoTDA控制台，左侧导航栏选择“规则 > 数据转发”后，并在“规则列表”中单击目标规则名称所在行右侧的“详情”，进入数据转发规则详情页面。

步骤2 选择“设置转发目标”，并单击转发目标所在行右侧的“测试”，开始编辑测试数据。

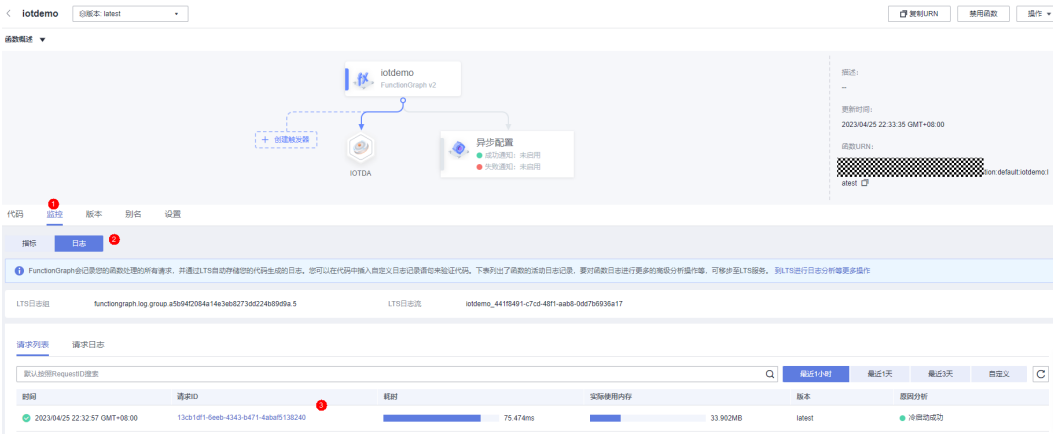
图 10-2 转发规则测试

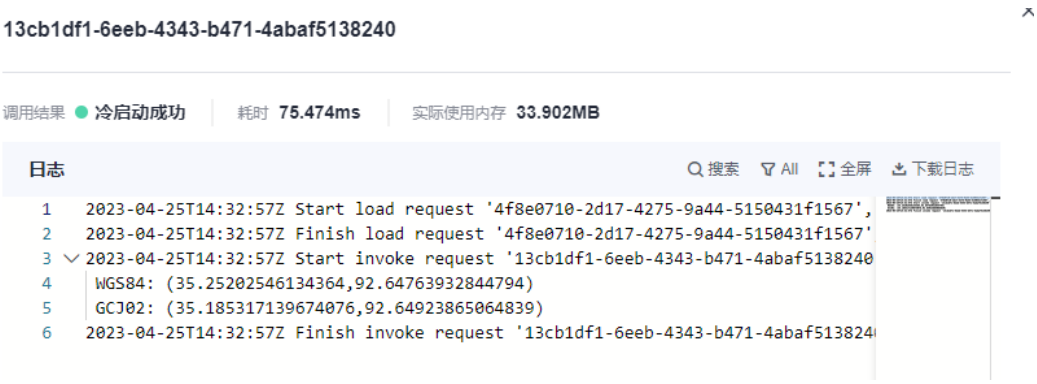


步骤3 输入测试数据单击“连通性测试”。

```
{
    "resource": "device.message",
    "event": "report",
    "event_time": "string",
    "notify_data": {
        "header": {
            "app_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
            "device_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
            "node_id": "ABC123456789",
            "product_id": "ABC123456789",
            "gateway_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
            "tags": [
                {
                    "tag_key": "testTagName",
                    "tag_value": "testTagValue"
                }
            ]
        },
        "body": {
            "lat": 92.64763932844794,
            "lng": 35.25202546134364
        }
    }
}
```

步骤4 到FunctionGraph 页面， 单击 “监控” “日志” 随后点击蓝色的请求id 查看日志。





可以对程序进行修改，使数据可以用于调用其他系统或进行持久化存储，如存储到obs等。

----结束

11 函数+DEW：加解密文件

11.1 案例描述

华为云DEW通过使用硬件安全模块HSM（Hardware Security Module, HSM）保护密钥的安全，所有的用户密钥都由HSM中的根密钥保护，避免密钥泄露。DEW对密钥的所有操作都会进行访问控制及日志跟踪，提供所有密钥的使用记录，满足审计和合规性要求，同时用户可通过购买专属加密实例加密用户业务系统（包含敏感数据加密、金融支付加密以及电子票据加密等），帮助用户加密企业自身的敏感数据（如合同、交易、流水等）以及企业用户的敏感数据（用户身份证号码、手机号码等），以防止黑客攻破网络、拖库导致数据泄露、内部用户非法访问或篡改数据等风险。本手册基于函数工作流服务实践所编写，用于指导您使用函数工作流服务加DEW来加解密特定的文件。

场景介绍

- 将文件上传到特定的OBS桶中。
- 将用户上传的每个文件进行加/解密。
- 将处理完后的文件上传到另一个指定的OBS桶中。

说明

1. 本教程必须使用两个不同的OBS桶。
2. 保证函数和OBS桶在一个区域（区域都选择默认即可）。

实现流程

- 在OBS服务中，创建两个桶。
- 创建函数，设置OBS触发器。
- 用户向其中一个桶上传文件。
- 触发函数执行，对文件加/解密。
- 函数将处理后的文件上传到指定桶中。

说明

完成本教程后，您的公有云账户将存在以下资源：

1. 2个OBS桶（上传需要处理的文件和存储处理后的文件）
2. 一个为文件加/解密的函数
3. 一个OBS触发器，用来关联函数和OBS桶

11.2 准备

创建函数及添加事件源之前，需要创建两个OBS桶，分别用来保存用户上传的文件和加/解密后输出的文件。

OBS桶创建以后，需要创建委托，给FunctionGraph函数赋权，确保FunctionGraph函数能够访问到OBS资源，本指导以加密文件为例：

创建 OBS 桶

注意

- 上传文件的源桶、输出文件的目标桶和函数必须处于同一个区域下。
- 必须使用两个不同的桶。如果使用一个桶，会无限执行函数。（源桶上传文件会触发函数执行，从而无限循环）。

操作步骤

步骤1 登录对象存储服务控制台，单击“创建桶”，进入“创建桶”界面。

步骤2 在“创建桶”界面，填写存储桶信息。

- 区域：根据实际情况设置
- 数据冗余存储策略：“单AZ存储”
- 桶名称输入：“dew-bucket-input”
- 默认存储类别：“标准存储”
- 桶策略选择："私有"
- 归档数据直读：“关闭”

单击“立即创建”，完成源桶创建。

步骤3 重复步骤2，创建目标桶。

区域及存储类别与源桶保持一致，桶名称命名为“dew-bucket-output”。

步骤4 完成桶创建以后，OBS桶列表有dew-bucket-input、dew-bucket-output两个桶。

----结束

创建 dew 密钥文件

注意

- 创建dew密钥和函数必须处于同一个区域下。

操作步骤

步骤1 在服务控制台左侧导航栏，选择“安全与合规 > 数据加密服务”进入DEW服务控制台，单击“创建密钥”，进入“创建密钥”界面。

步骤2 在“创建密钥”界面，单击“确定”，完成密钥创建。

步骤3 完成密钥创建以后，需要记录主密钥id，供后面使用。

----结束

创建委托

步骤1 在服务控制台左侧导航栏，选择“管理与监管 > 统一身份认证服务”进入统一身份认证控制台，在左侧导航栏单击“委托”，进入“委托”界面。

步骤2 单击“创建委托”，进入“创建委托”界面。

步骤3 填写委托信息。

- 委托名称：输入“serverless_trust”。
- 委托类型：选择“云服务”。
- 云服务：选择“函数工作流 FunctionGraph”。
- 持续时间：选择“永久”。
- 描述：填写描述信息。

步骤4 单击“下一步”，进入委托选择页面，在“配置权限”界面勾选“Tenant Administrator”，单击“确定”。

说明

Tenant Administrator：拥有该权限的用户可以对企业拥有的所有云资源执行任意操作。

步骤5 单击“确定”，完成权限委托设置。

----结束

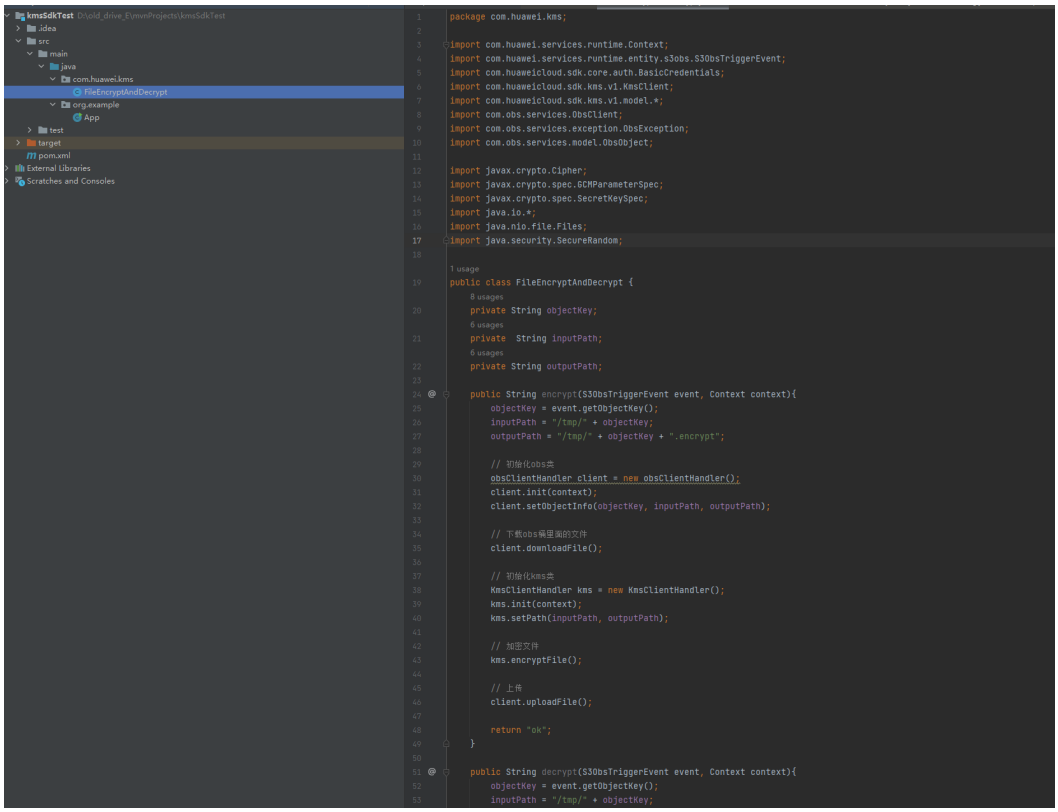
11.3 构建程序

本例提供了为文件加/解密的程序包，使用空白模板创建函数，用户可以使用示例代码学习使用。

创建程序包

本例使用Java8语言实现加/解密的功能，有关函数开发的过程请参考**Java函数开发**。本例不再介绍业务功能实现的代码，样例代码目录如**图11-1**所示。

图 11-1 样例代码目录



其中FileEncryptAndDecrypt为函数执行的入口类，FileEncryptAndDecrypt类中入口函数的代码如下：

```
package com.huawei.kms;
import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.s3obs.S3ObsTriggerEvent;
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.kms.v1.KmsClient;
import com.huaweicloud.sdk.kms.v1.model.*;
import com.obs.services.ObsClient;
import com.obs.services.exception.ObsException;
import com.obs.services.model.ObsObject;
import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.nio.file.Files;
import java.security.SecureRandom;
public class FileEncryptAndDecrypt {
    private String objectKey;
    private  String inputPath;
    private String outputPath;
    public String encrypt(S3ObsTriggerEvent event, Context context){
        objectKey = event.getObjectKey();
        inputPath = "/tmp/" + objectKey;
        outputPath = "/tmp/" + objectKey + ".encrypt";
        // 初始化obs类
        obsClientHandler client = new obsClientHandler();
        client.init(context);
        client.setObjectInfo(objectKey, inputPath, outputPath);
        // 下载obs桶里面的文件
        client.downloadFile();
        // 初始化kms类
        KmsClientHandler kms = new KmsClientHandler();
```

```
        kms.init(context);
        kms.setPath(inputPath, outputPath);
        // 加密文件
        kms.encryptFile();
        // 上传
        client.uploadFile();
        return "ok";
    }
    public String decrypt(S3ObsTriggerEvent event, Context context){
        objectKey = event.getObjectKey();
        inputPath = "/tmp/" + objectKey;
        outputPath = "/tmp/" + objectKey + ".decrypt";
        // 初始化obs类
        obsClientHandler client = new obsClientHandler();
        client.init(context);
        client.setObjectInfo(objectKey, inputPath, outputPath);
        // 下载obs桶里面的文件
        client.downloadFile();
        // 初始化kms类
        KmsClientHandler kms = new KmsClientHandler();
        kms.init(context);
        kms.setPath(inputPath, outputPath);
        // 加密文件
        kms.decryptFile();
        // 上传
        client.uploadFile();
        return "ok";
    }
    static class KmsClientHandler {
        // DEW服务接口版本信息，当前固定为v1.0
        private static final String KMS_INTERFACE_VERSION = "v1.0";
        private static final String AES_KEY_BIT_LENGTH = "256";
        private static final String AES_KEY_BYTE_LENGTH = "32";
        private static final String AES_ALG = "AES/GCM/PKCS5Padding";
        private static final String AES_FLAG = "AES";
        private static final int GCM_TAG_LENGTH = 16;
        private static final int GCM_IV_LENGTH = 12;
        private String ACCESS_KEY;
        private String SECRET_ACCESS_KEY;
        private String PROJECT_ID;
        private String KMS_ENDPOINT;
        private String keyId;
        private String cipherText;
        private String inputPath;
        private String outputPath;
        private Context context;
        private KmsClient kmsClient = null;
        void init(Context context) {
            this.context = context;
        }
        void initKmsClient() {
            if (kmsClient == null) {
                ACCESS_KEY = context.getAccessKey();
                SECRET_ACCESS_KEY = context.getSecretKey();
                PROJECT_ID = context.getProjectID();
                KMS_ENDPOINT = context.getUserData("kms_endpoint");
                keyId = context.getUserData("kms_key_id");
                cipherText = context.getUserData("cipher_text");
                final BasicCredentials auth = new
BasicCredentials().withAk(ACCESS_KEY).withSk(SECRET_ACCESS_KEY).withProjectId(PROJECT_ID);
                kmsClient = kmsClient.newBuilder().withCredential(auth).withEndpoint(KMS_ENDPOINT).build();
            }
        }
        byte[] getEncryptPlainKey() {
            final CreateDatakeyRequest createDatakeyRequest = new
CreateDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
                    .withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));
            final CreateDatakeyResponse createDatakeyResponse =
```

```
kmsClient.createDatakey(createDatakeyRequest);
            final String cipherText = createDatakeyResponse.getCipherText();
            return hexToBytes(createDatakeyResponse.getPlainText());
        }
        byte[] hexToBytes(String hexString) {
            final int stringLength = hexString.length();
            assert stringLength > 0;
            final byte[] result = new byte[stringLength / 2];
            int j = 0;
            for (int i = 0; i < stringLength; i += 2) {
                result[j++] = (byte) Integer.parseInt(hexString.substring(i, i + 2), 16);
            }
            return result;
        }
        public void setPath(String inputPath, String outputPath) {
            this.inputPath = inputPath;
            this.outputPath = outputPath;
        }
        public void  encryptFile() {
            final File outEncryptFile = new File(outputPath);
            final File inFile = new File(inputPath);
            final byte[] iv = new byte[GCM_IV_LENGTH];
            final SecureRandom secureRandom = new SecureRandom();
            secureRandom.nextBytes(iv);
            doFileFinal(Cipher.ENCRYPT_MODE, inFile, outEncryptFile, getEncryptPlainKey(), iv);
        }
        byte[] getDecryptPlainKey() {
final CreateDatakeyRequest createDatakeyRequest = new
CreateDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
        .withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));
// 创建数据密钥
final CreateDatakeyResponse createDatakeyResponse = kmsClient.createDatakey(createDatakeyRequest);
            final DecryptDatakeyRequest decryptDatakeyRequest  = new
DecryptDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
                    .withBody(new
DecryptDatakeyRequestBody().withKeyId(keyId).withCipherText(createDatakeyResponse.getCipherText()
).withDatakeyCipherLength(AES_KEY_BYTE_LENGTH));
            return hexToBytes(kmsClient.decryptDatakey(decryptDatakeyRequest).getDataKey());
        }
        public void  decryptFile() {
            final File outEncryptFile = new File(outputPath);
            final File inFile = new File(inputPath);
            final byte[] iv = new byte[GCM_IV_LENGTH];
            final SecureRandom secureRandom = new SecureRandom();
            secureRandom.nextBytes(iv);
            doFileFinal(Cipher.DECRYPT_MODE, inFile, outEncryptFile, getDecryptPlainKey(), iv);
        }
        /**          //     * 对文件进行加解密          //     *          //     * @param cipherMode 加密模式，可选值为
Cipher.ENCRYPT_MODE或者Cipher.DECRYPT_MODE          //     * @param infile     加解密前的文
件          //     * @param outFile    加解密后的文件          //     * @param keyPlain   明文密钥          //     *
@param iv          初始化向量          //     */          void doFileFinal(int cipherMode, File infile, File outFile,
byte[] keyPlain, byte[] iv) {
            try (BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(infile.toPath()));
                 BufferedOutputStream bos = new
BufferedOutputStream(Files.newOutputStream(outFile.toPath()))) {
                final byte[] bytIn = new byte[(int) infile.length()];
                final int fileLength = bis.read(bytIn);
                assert fileLength > 0;
                final SecretKeySpec secretKeySpec = new SecretKeySpec(keyPlain, AES_FLAG);
                final Cipher cipher = Cipher.getInstance(AES_ALG);
                final GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(GCM_TAG_LENGTH *
Byte.SIZE, iv);
                cipher.init(cipherMode, secretKeySpec, gcmParameterSpec);
                final byte[] bytOut = cipher.doFinal(bytIn);
                bos.write(bytOut);
            } catch (Exception e) {
                throw new RuntimeException(e.getMessage());
            }
```

```
        }
    }
    static class obsClientHandler {
        private ObsClient obsClient = null;
        private String inputBucketName;
        private String outputBucketName;
        private String objectKey;
        private Context context;
        private String localInPath;
        private String localOutPath;
        public void init(Context context) {
            this.context = context;
        }
        void initObsclient() {
            if (obsClient == null) {
                inputBucketName = context.getUserData("input_bucket");
                outputBucketName = context.getUserData("output_bucket");
                String ACCESS_KEY = context.getAccessKey();
                String SECRET_ACCESS_KEY = context.getSecretKey();
                String OBS_ENDPOINT = context.getUserData("obs_endpoint");
                obsClient = new ObsClient(ACCESS_KEY, SECRET_ACCESS_KEY, OBS_ENDPOINT);
            }
        }
        public  void setObjectInfo(String objectKey, String inPath, String outPath) {
            this.objectKey = objectKey;
            localInPath = inPath;
            localOutPath = outPath;
        }
        public void downloadFile() {
            initObsclient();
            try {
                ObsObject obsObject = obsClient.getObject(inputBucketName, objectKey);
                InputStream inputStream = obsObject.getObjectContent();
                byte[] b = new byte[1024];
                int len;
                FileOutputStream fileOutputStream = new FileOutputStream("/tmp/" + objectKey);
                while ((len = inputStream.read(b)) != -1) {
                    fileOutputStream.write(b);
                }
                inputStream.close();
                fileOutputStream.close();
            } catch (ObsException ex) {
                ex.printStackTrace();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        public void uploadFile() {
            try {
                // 待上传的本地文件路径，需要指定到具体的文件名
                FileInputStream fis = new FileInputStream(new File("/tmp/" + objectKey + ".encrypt"));
                obsClient.putObject(outputBucketName, objectKey, fis);
                fis.close();
            } catch (FileNotFoundException e) {
                throw new RuntimeException(e);
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

创建函数

创建函数的时候，必须选择委托包含OBS和DEW访问权限的委托，否则不能使用OBS和DEW服务。

步骤1 登录函数工作流控制台，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。

步骤2 单击“创建函数”，进入创建函数流程。

步骤3 填写函数配置信息。

输入基础配置信息，完成后单击“创建函数”。

- 函数名称：输入“fss_examples_dew”
- 委托名称：选择创建委托中创建的“serverless_trust”
- 运行时语言：选择“Java8”

步骤4 进入fss_examples_dew函数详情页，配置如下信息。

1. 在“代码”页签，代码选择“上传自JAR文件”，上传样例代码编译后的jar包，上传成功后单击“确定”。
2. 在“代码”页签，单击“部署”，等待部署完成。
3. 在“设置 > 常规设置”页签，设置如下信息，完成后单击“保存”。
   - 内存：选择“128”
   - 执行超时时间：输入“3”
   - 函数执行入口：默认“com.huawei.kms.FileEncryptAndDecrypt.encrypt”
   - 所属应用：默认“default”
   - 描述：输入“文件加解密”
4. 在“设置 > 环境变量”页签，输入环境信息，完成后单击“保存”。

   dew_endpoint：dew服务的endpoint地址

   dew_key_id：用户主密钥ID。

   input_bucket: 输入文件对应的obs桶。

   output_bucket: 加解密后上传的obs桶。

   obs_endpoint: obs服务对应的endpoint。

表 11-1 环境变量

环境变量	说明
dew_endpoint	DEW服务终端节点，获取地址请参考地区和终端节点。。
dew_key_id	用户主密钥ID。
input_bucket	存放输入文件的OBS桶。
output_bucket	存放加密后上传文件的OBS桶。
obs_endpoint	OBS服务终端节点，获取地址请参考地区和终端节点。

----结束

11.4 添加事件源

OBS桶及函数创建以后，可以为函数添加事件源，添加OBS事件源是通过创建OBS触发器实现的，步骤如下。

步骤1 用户进入fss_examples_dew函数详情页，在“触发器”页签，单击“创建触发器”，弹出“创建触发器”界面。

步骤2 触发器类型选择“存储(OBS)”，填写触发器配置信息，如图11-2所示。

桶选择创建OBS桶中创建的“input_bucket”桶。

事件选择“Post”、“Put”。

图 11-2 创建 OBS 触发器

步骤3 单击“确定”，完成触发器创建。

说明

OBS触发器创建以后，当有文件上传或更新至dew-bucket-input桶时，生成事件，触发函数执行。

----结束

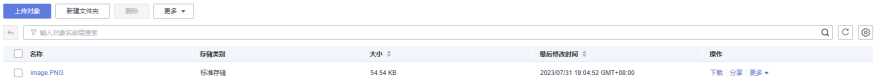
11.5 处理文件

当文件上传后更新至dew-bucket-input桶时，会生成事件，触发函数运行，将文件加解密，保存在dew-bucket-output中。

上传文件生成事件

登录对象存储服务控制台，进入dew-bucket-input桶对象界面，上传image.jpg文件，如图11-3所示。

图 11-3 上传文件



触发函数自动运行

上传文件至dew-bucket-input桶，OBS生成事件触发函数运行，对文件加解密，输出文件存放在dew-bucket-output桶中。可以在fss_examples_dew函数详情页“日志”页签查看函数运行日志。

进入dew-bucket-output桶对象界面，可以看到输出的图片image.jpg.encrypt，如图11-4所示。单击操作列的“下载”可将文件下载至本地查看处理效果。

图 11-4 输出文件

12 工作流+函数：自动化处理 OBS 中数据

12.1 案例概述

本手册基于函数流服务实践所编写，用于指导您使用函数流服务实现OBS数据处理的功能。（当前函数流暂时支持华东-上海一、亚太-新加坡）

场景介绍

用户使用函数流编排函数方式自动化处理OBS中的数据（如视频解析、图片转码、视频截图等）。

- 用户将图片上传到特定的OBS桶中。
- 函数流编排函数算子，实现下载OBS中数据进行图片转码，并以流的形式返回给客户端。

说明

保证函数和OBS桶在一个区域（区域都选择默认即可）。

实现流程

- 在OBS服务中，创建1个桶。
- 用户向OBS桶上传图片。
- 创建函数。
- 创建函数流，编排函数。
- 触发函数流执行，对图片进行转码处理。

说明

完成本教程后，您的公有云账户将存在以下资源：

1. 1个OBS桶（上传需要处理的图像）
2. 1个图片处理的函数（test-rotate）
3. 1个编排函数的函数流（test-rotate-workflow）

12.2 准备

创建函数前，需要创建1个OBS桶，用来保存用户上传的图片。

OBS桶创建以后，需要创建“委托”，给FunctionGraph函数赋权，确保FunctionGraph函数能够访问到OBS资源。

创建 OBS 桶

注意

上传图片的源桶和函数必须处于同一个区域下。

操作步骤

步骤1 在服务控制台左侧导航栏，选择“存储 > 对象存储服务”进入对象存储服务控制台，单击“创建桶”，进入“创建桶”界面。

在“创建桶”界面，填写存储桶信息。

- 区域：根据实际情况设置
- 桶名称输入：“your-bucket-input”
- 数据冗余存储策略：“单AZ存储”
- 默认存储类别：“标准存储”
- 桶策略："私有"
- 默认加密：“关闭”
- 归档数据直读：“关闭”

其余参数保持默认，单击“立即创建”，完成源桶创建。

完成桶创建以后，OBS桶列表有your-bucket-input桶。

----结束

创建委托

步骤1 在服务控制台左侧导航栏，选择“管理与监管 > 统一身份认证服务”进入统一身份认证服务控制，在左侧导航栏单击“委托”，进入“委托”界面。

单击“创建委托”，进入“创建委托”界面。

填写委托信息。

- 委托名称：输入“serverless_trust”。
- 委托类型：选择“云服务”。
- 云服务：选择“函数工作流 FunctionGraph”。
- 持续时间：选择“永久”。
- 描述：填写描述信息。

单击“下一步”，进入委托选择页面，在“配置权限”界面勾选“Tenant Administrator”，单击“确定”。

说明

Tenant Administrator：拥有该权限的用户可以对企业拥有的所有云资源执行任意操作。

单击“确定”，完成权限委托设置。

----结束

12.3 构建程序

本例提供一个图片旋转的样例代码供学习使用。

创建程序包

本例使用Golang语言实现图片旋转的功能，有关函数开发的过程请参考Golang函数开发。本例不再介绍业务功能实现的代码，样例代码目录如图12-1所示。

图 12-1 样例代码目录

```go
func Test(b []byte, ctx context.RuntimeContext) (interface{}, error) {
    ak := ctx.GetAccessKey()
    sk := ctx.GetSecretKey()
    obsAddress := os.Getenv( key: "obsAddress")
    bucket := os.Getenv( key: "bucket")
    object := os.Getenv( key: "object")
    client, err := obs.New(ak, sk, obsAddress)
    if err ≠ nil {
        log.Printf( format: "err:%v", err)
        return nil, err
    }
    output, err := client.GetObject(&obs.GetObjectInput{
        GetObjectMetadataInput: obs.GetObjectMetadataInput{
            Bucket: bucket,
            Key:    object,
        },
    })
    if err ≠ nil {
        log.Printf( format: "err:%v", err)
        return nil, err
    }
    defer output.Body.Close()
    img, err := jpeg.Decode(output.Body)
    if err ≠ nil {
        log.Printf( format: "err:%v", err)
        os.Exit( code: -1)
    }
    res := rotate180(img)
    buffer := bytes.NewBuffer( buf: nil)
    err = jpeg.Encode(buffer, res, &jpeg.Options{ Quality: 100})
    if err ≠ nil {
        fmt.Println(err)
        return nil, err
    }
    err = ctx.Write(bytes.NewReader(buffer.Bytes()))
    return struct {}{}, err
}

func rotate180(m image.Image) image.Image {
    rotate180 := image.NewRGBA(image.Rect( x0: 0,  y0: 0, m.Bounds().Dx(), m.Bounds().Dy()))
    for x := m.Bounds().Min.X; x < m.Bounds().Max.X; x++ {
        for y := m.Bounds().Min.Y; y < m.Bounds().Max.Y; y++ {
            rotate180.Set(m.Bounds().Max.X-x, m.Bounds().Max.Y-y, m.At(x, y))
        }
    }
    return rotate180
}
```

创建函数

创建函数的时候，必须选择委托包含OBS访问权限的委托，否则不能使用OBS服务。

步骤1 登录函数工作流控制台，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。

单击“创建函数”，选择“创建空白函数”进入创建函数流程。

填写函数配置信息。

输入基础配置信息，完成后单击“创建函数”。

- 函数名称：输入“test-rotate”
- 委托名称：选择创建委托中创建的“serverless_trust”
- 运行时语言：选择“Go1.x”

  进入test-rotate函数详情页，配置如下信息。

a. 在“代码”页签，代码选择“上传自ZIP文件”，上传样例代码“**go-test.zip**”编译后的二进制文件，完成后单击“部署”。

b. 在“设置 > 常规设置”页签，设置如下信息，完成后单击“保存”。

- 内存：选择“256”
- 执行超时时间：输入“40”
- 函数执行入口：默认“handler”，无需修改
- 所属应用：默认“default”
- 描述：输入“旋转图片”

c. 在“设置 > 环境变量”页签，输入环境信息，完成后单击“保存”。

键bucket：handler.go文件中定义的拉取图片的OBS桶参数，值your-bucket-output：创建OBS桶中创建的存放图片OBS桶;

键object：handler.go文件中定义的拉取图片名称参数，值your-picture-name

键obsAddress：handler.go文件中定义的拉取图片的OBS桶的地址参数，值obs.region.myhuaweicloud.com。

----结束

表 12-1 环境变量说明

环境变量	说明
bucket	handler.go文件中定义的拉取图片的OBS桶参数。
object	handler.go文件中定义的拉取图片名称参数。
obsAddress	handler.go文件中定义的拉取图片的OBS桶的地址参数，键obsAddress值的格式为obs.{region}.myhuaweicloud.com，region的值，请参考**地区和终端节点**

----结束

创建函数流

步骤1 返回函数工作流控制台，在左侧导航栏选择“函数流”，进入函数流列表界面。

单击“创建快速函数流”，进入创建快速函数流流程。

图 12-2 创建快速函数流

步骤2 拖拽一个函数节点，点击函数节点配置元信息：

- 应用：默认“default”；
- 函数：选择上一步创建好的函数test-rotate；
- 版本：默认“latest”；
- 其他参数默认值即可。

图 12-3 配置元信息

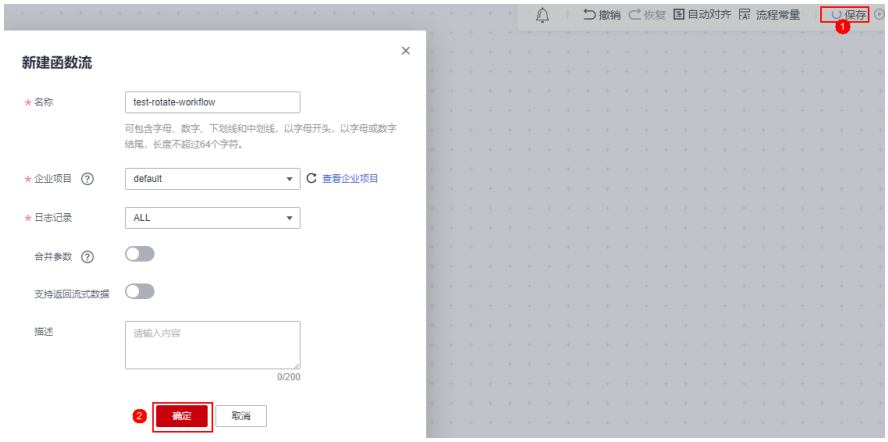
参数配置完成后，单击“确定”。

步骤3 函数流节点创建完成后，单击右上角“保存”，配置如下函数流基本信息，完成后单击“确定”，完成函数流创建。

- 名称：test-rotate-workflow；
- 企业项目：默认“default”；
- 日志记录：默认“ALL”；

其他参数保持默认值。

图 12-4 保存函数流



----结束

12.4 处理图片

图片上传至your-bucket-input桶，使用工具模拟客户端触发函数流运行，将上传图片旋转180°，并以流数据返回给客户端。

上传图片

登录**对象存储服务控制台**，进入your-bucket-input桶对象界面，上传image.jpeg图片如**图12-5**，上传完成后如**图12-6**所示。

图 12-5 示例

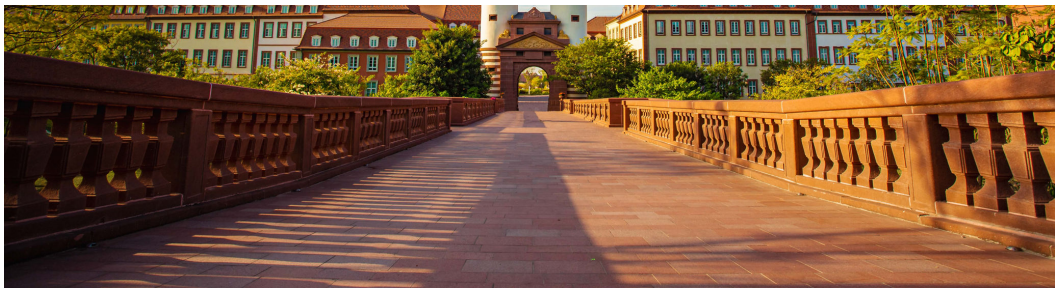
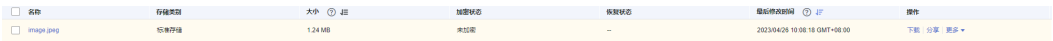
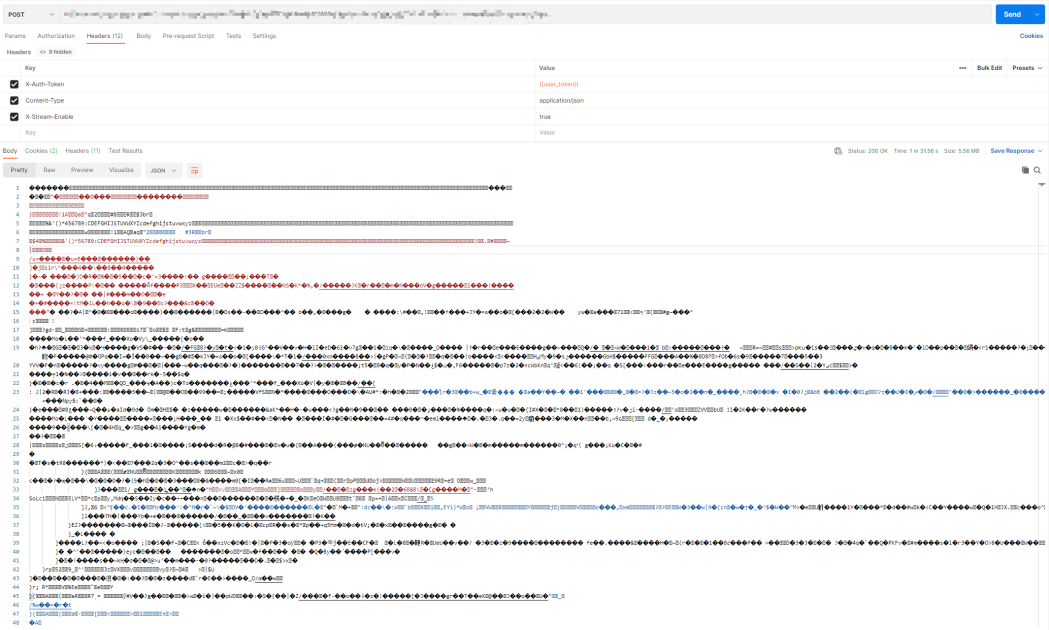


图 12-6 上传图片



使用 postman 触发函数流执行



上面的字节流保存成图片后如下图所示：

13 函数+LTS：日志实时过滤

13.1 案例概述

本章节作为最佳实践的整体介绍，包含以下内容：

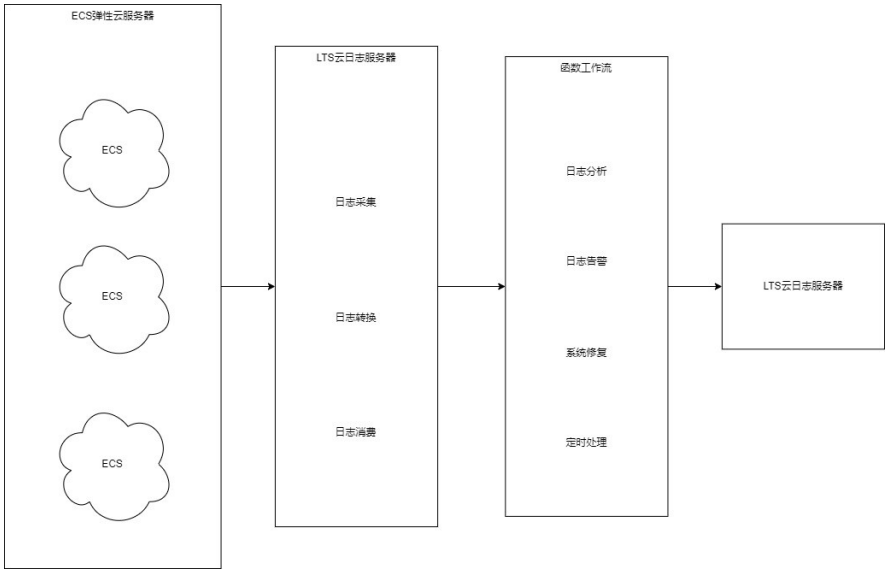
- 场景介绍和案例价值点
- 准备
- 构建程序
- 添加事件源
- 处理结果
- 应用扩展

场景介绍

通过云日志服务LTS，快速完成ECS等服务器的任务运行日志的采集、加工和转换。

通过函数工作流服务中的函数创建LTS触发器获取日志数据，经由自定义函数对日志中的关键信息进行分析和处理，把过滤后的日志转存到另外的日志流中，如图13-1所示。

图 13-1 处理流程



案例价值点

- 通过云日志服务LTS，快速完成日志采集和转换。
- 基于Serverless无服务架构的函数计算提供事件触发、弹性伸缩、无需运维、按需付费的数据加工、分析。
- 把过滤后的日志转存到另外的日志流，原日志流根据设置的过期时间自动删除，降低日志存储费用。

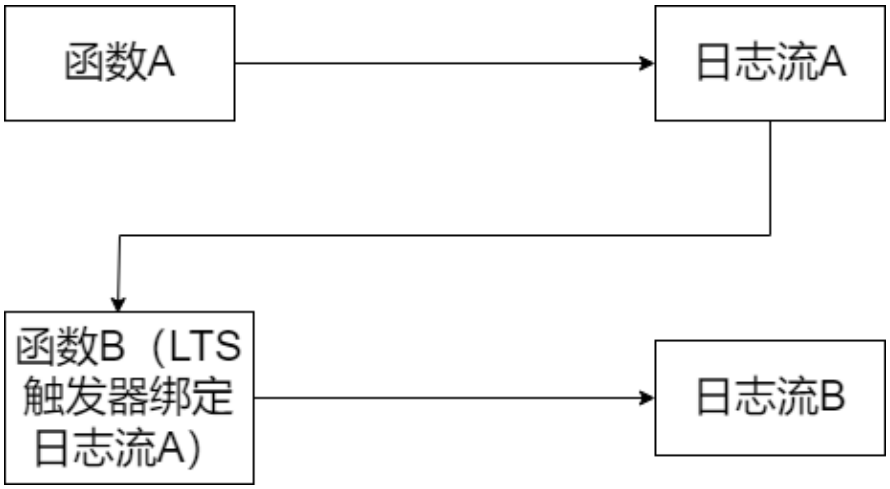
13.2 准备

本例提供了日志实时过滤功能的程序包及依赖包，用户可以下载 **lts_cleanse.zip**（包含函数A代码文件write_log.py、函数B代码文件lts_cleanse.py及依赖包huaweicloudsdklts）、**lts_cleanse.zip.sha256** 学习使用。

日志采集和存储

- 在云日志服务创建日志组，此处以test1206、test-1121为例，创建过程请参考**创建日志组**。
- 在云日志服务创建日志流，此处以test-206、test-1121为例，创建过程请参考**创建日志流**。
- 创建函数A，负责写入日志到test-206。函数A代码样例请参考write_log.py。
- 创建函数B，挂载LTS触发器，接收test-206的日志，处理日志并发结果写入test-1121。函数B代码样例请参考lts_cleanse.py。
- 在云日志服务配置Agent，快速将ECS等服务器上日志采集到指定的日志组，配置过程请参考**安装ICAgent**。

图 13-2 流程图



创建委托

步骤1 登录统一身份认证服务控制台。

步骤2 在统一身份认证服务的左侧导航窗格中，选择“委托”菜单，单击右上方的“+创建委托”，如图13-3所示。

图 13-3 创建委托

步骤3 开始配置委托。

- 委托名称：LtsOperation。
- 委托类型：选择“云服务”。
- 云服务：选择“函数工作流 FunctionGraph”。
- 持续时间：选择“永久”。
- 描述：填写描述信息。

步骤4 单击“下一步”，进入委托权限选择页面，在右方搜索框中搜索“LTS Administrator”权限和“Tenant Administrator”并勾选，如图13-4所示。

图 13-4 选择权限

说明

选择“LTS Administrator”，由于该策略有依赖，在勾选LTS Administrator时，还会自动勾选依赖的策略：Tenant Guest。

步骤5 单击“下一步”，请根据业务需要选择权限的作用范围。

----结束

13.3 构建程序

前提条件

（1）函数中的IP地址为LTS的接入点，获取接入点IP方法如下：

1. 登录云日志服务 LTS控制台，在左侧导航栏选择“主机管理”；
2. 在“主机管理”页右上方，单击“安装ICAgent”；

图 13-5 安装 ICAgent

3. 在弹出的“安装ICAgent”窗口中获取接入点IP。

图 13-6 接入点 IP

（2）函数中的log_group_id和log_stream_id变量值的获取，请参考**获取账号ID、项目ID、日志组ID、日志流ID**。

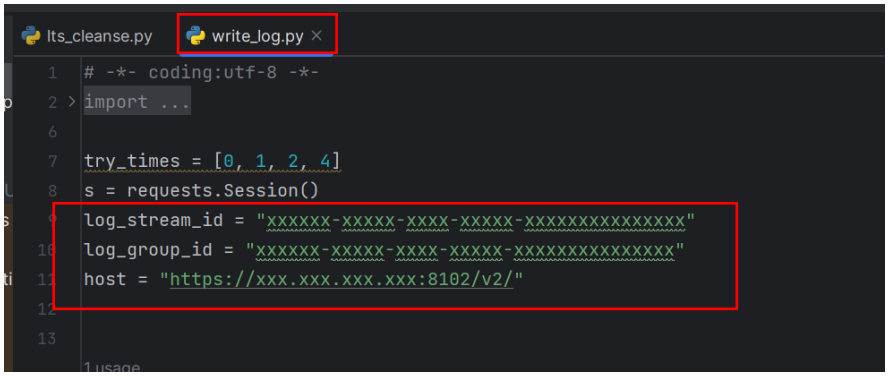
（3）制作函数B需要的 lts 依赖包，具体添加依赖方法请参考**如何在函数平台创建依赖包**和**如何为函数添加依赖包**。制作依赖包时可以参考命令“pip install huaweicloudsdklts”。同时，**示例代码**中包含了已适用于python3.9的huaweicloudsdklts依赖。

创建功能函数

创建实现日志提取功能的函数，将示例代码包上传。创建过程请参考**创建事件函数**，运行时语言选择“Python3.9”，委托名称选择**创建委托**中的“LtsOperation”。

创建函数A，代码样例请参考write_log.py。函数A代码中host、log_group_id和log_stream_id使用对应接入点和创建好的日志组test-1206、日志流test-206的ID，如**图13-7**所示。

图 13-7 write_log.py



创建函数B，代码样例请参考lts_cleanse.py。函数B代码中host、log_group_id和log_stream_id使用对应接入点和创建好的日志组test-1121、日志流test-1121的ID，并为函数B添加依赖huaweicloudsdklts，如**图13-8**和**图13-9**所示。

图 13-8 lts_cleanse.py

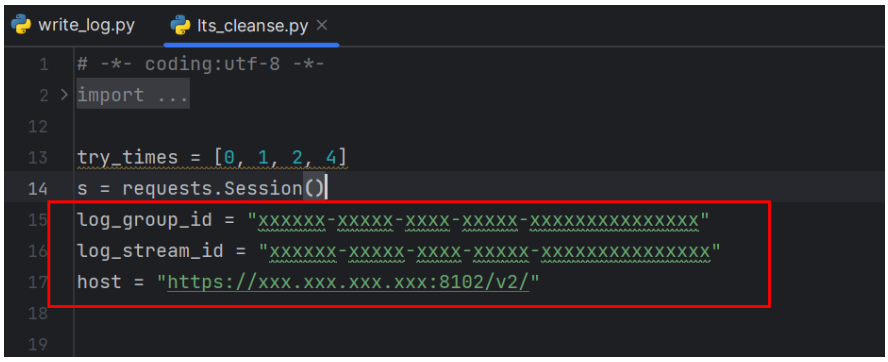
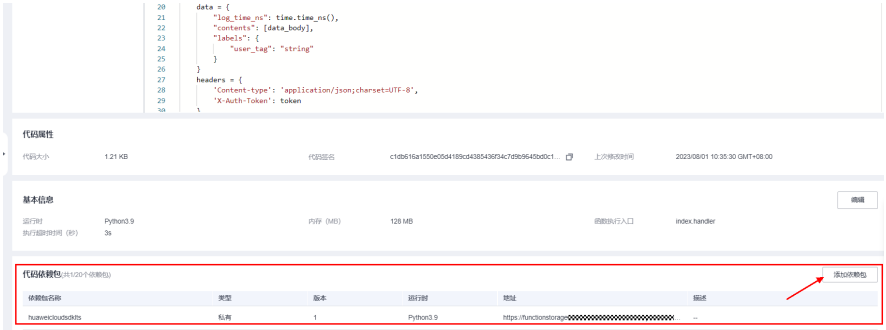


图 13-9 为函数 B 添加依赖包

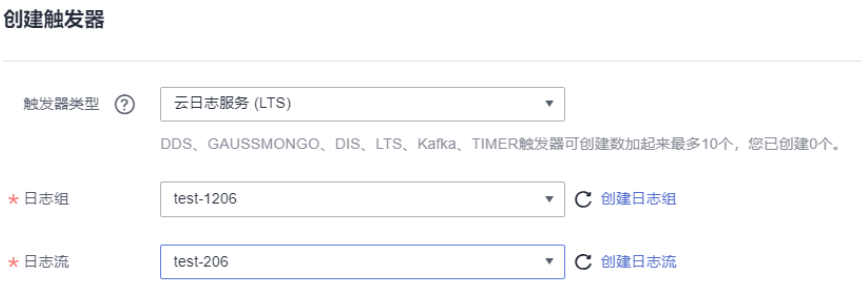


函数实现的功能是：将收到的日志事件数据进行base64解码，然后提取出包含“WRN”、“WARN”、“ERR”或“ERROR”关键字的告警日志，将此级别的日志投递至创建好的LTS日志流中集中存储。可根据您的业务日志的具体内容配置相应的日志提取条件。

13.4 添加事件源

选择准备中创建的日志组和日志流，创建LTS触发器，LTS触发器配置如图13-10所示。

图 13-10 创建 LTS 触发器



云日志服务LTS的消费端在日志累积大小或日志累积时间满足条件时消费LTS日志数据，并根据订阅该组LTS日志数据的函数URN触发函数执行。

13.5 处理结果

若日志包含“WRN”、“WARN”、“ERR”或“ERROR”关键字的告警日志，则过滤出来并转储到准备好的日志流中。以下图13-11和图13-12是过滤前和过滤后的实时日志对比。

图 13-11 过滤前日志

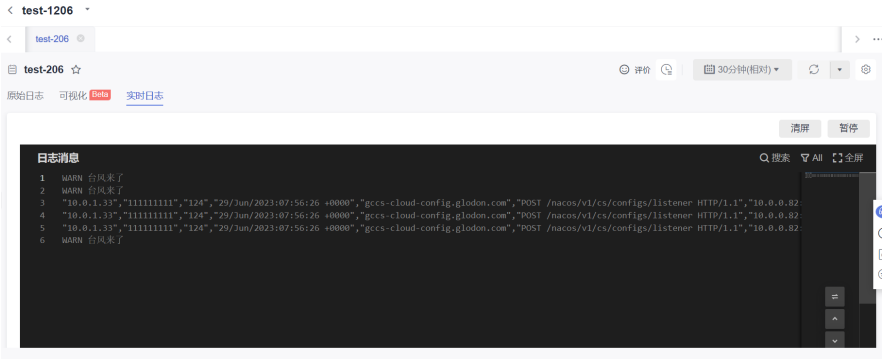
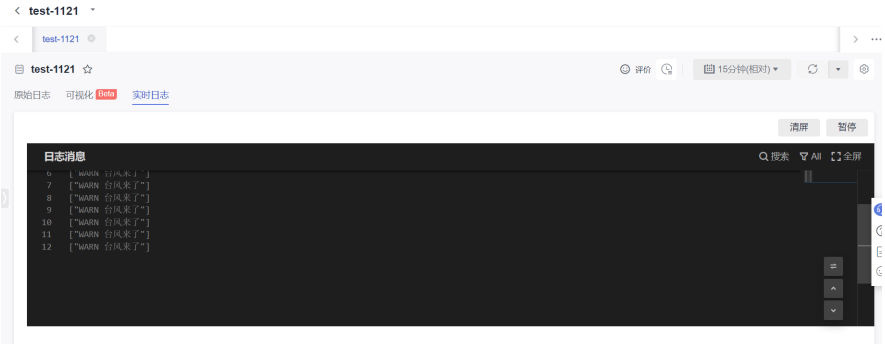


图 13-12 过滤后日志



您可以通过函数指标查看函数的调用情况，如下 3 张图所示。

图 13-13 函数指标 1

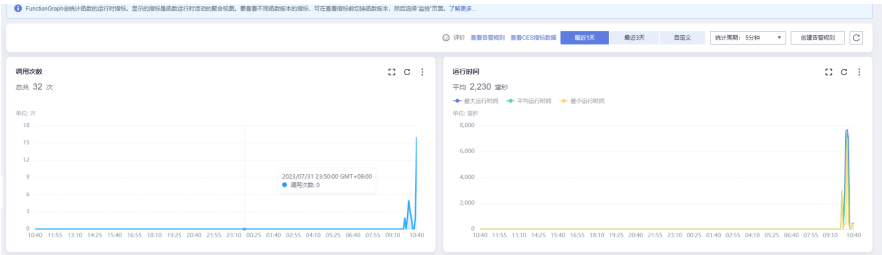


图 13-14 函数指标 2

图 13-15 函数指标 3

13.6 应用扩展

本案例展示了函数工作流服务配合使用云日志服务LTS实现日志云端处理并转储消息到LTS的功能。函数工作流服务+LTS云日志服务的应用广泛，如以下应用场景：利用函数的TIMER触发器，定时对指定LTS日志流中的日志数据进行个性化分析和处理，删除冗余的日志，节省空间和费用。

14 AS&FunctionGraph 支持优雅关机

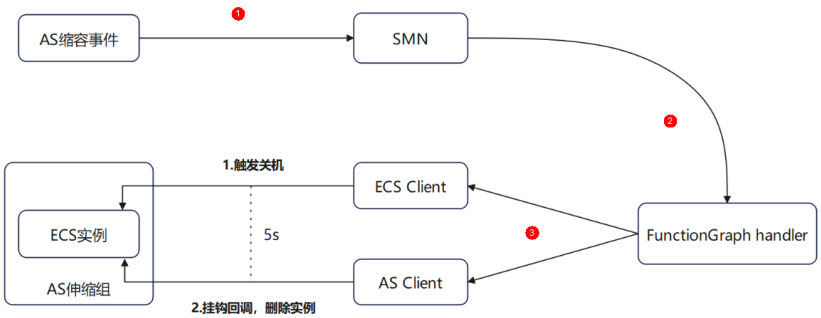
14.1 案例概述

在弹性伸缩服务中，伸缩组的实例缩容过程期间，先对实例进行关机，待客户关机后的清理工作完成后，再由弹性伸缩继续移除并删除对应实例，达到优雅关机的效果。

应用场景

1. 通过配置AS缩容事件的消息通知，转发缩容消息至SMN消息通知服务。
2. 再通过函数工作流服务，接收SMN通知转发过来的伸缩组的缩容消息，经过自定义函数获取伸缩实例等信息。
3. 调用ECS服务对缩容实例进行关机操作。关机操作结束后，等待5秒（这里可以根据业务需要进行修改），再调用AS服务继续伸缩组的缩容操作，对实例进行删除。

图 14-1 AS 优雅关机流程图



14.2 准备

1. 获取弹性伸缩服务优雅关机的程序包。
2. 创建委托ASOperation，添加“ECS FullAccess”以及“AutoScaling FullAccess”权限，具体详情请参考创建委托。

创建委托

1. 登录**统一身份认证服务控制台**。
2. 在统一身份认证服务的左侧导航栏中，选择“委托”页签，单击右上方的“+创建委托”。

图 14-2 创建委托

3. 开始配置委托。
   - 委托名称：ASOperation。
   - 委托类型：选择“云服务”。
   - 云服务：选择“函数工作流 FunctionGraph”。
   - 持续时间：选择“永久”。
   - 描述：填写描述信息。
4. 单击“下一步”，进入委托选择页面，在右侧搜索框中搜索“ECS FullAccess”与“AutoScaling FullAccess”权限并勾选。
5. 单击“下一步”，请根据业务需要选择权限的作用范围。
6. 单击“确定”，完成权限委托设置。

14.3 构建程序

本例提供了支持优雅关机功能的程序包，使用空白模板创建函数，用户可以**下载（as_graceful_shutdown.zip）**学习使用。

创建函数

1. 登录**函数工作流控制台**，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。
2. 单击“创建函数”，进入创建函数流程。
3. 填写函数配置信息。输入基础配置信息，完成后单击“创建函数”。
   - 函数名称：输入“as_graceful_shutdown”。
   - 委托名称：选择**创建委托**中创建的“ASOperation”。
   - 运行时语言：选择“Python3.6”。
4. 进入as_graceful_shutdown函数详情页，在“代码”页签，代码选择“上传自ZIP文件”，上传样例代码中的“**as_graceful_shutdown.zip**”，完成后单击“部署”。然后，在“设置 > 常规设置”页签，设置如下信息，完成后单击“保存”。
   - 内存：选择“128”。
   - 执行超时时间：输入“10”。

- 函数执行入口：默认“index.handler”，无需修改。
- 所属应用：默认“default”。
- 描述：输入“AS优雅关机”。

设置环境变量

进入as_graceful_shutdown函数，在“设置 > 环境变量”页签，配置环境变量，说明如表14-1所示，完成后单击“保存”。

表 14-1 环境变量说明

环境变量	说明
region	伸缩组所在的区域
projectId	伸缩组所在的Project ID
ak	AK（Access Key ID）：访问密钥ID。与私有访问密钥关联的唯一标识符；访问密钥ID和私有访问密钥一起使用，对请求进行加密签名。如何获取AK请参考获取AK/SK。
sk	SK（Secret Access Key）：与访问密钥ID结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。如何获取SK请参考获取AK/SK。

添加依赖包

1. 制作“huaweicloudsdk_ecs_core_py3.6”与“huaweicloudsdk_as_core_py3.6”依赖包。制作依赖包详细操作请参见配置函数依赖。

   说明

   函数工作流服务除了支持用户自定义创建依赖包外，平台也提供了现成依赖包供用户使用，下载请参见如下：

   - **huaweicloudsdk_ecs_core_py3.6**
   - huaweicloudsdk_as_core_py3.6（该依赖包暂时无提供下载链接，敬请等待平台后续补充）

2. 返回函数工作流控制台，在左侧导航栏选择“函数 > 依赖包管理 > 创建依赖包”，分别创建“huaweicloudsdk_ecs_core_py3.6”与“huaweicloudsdk_as_core_py3.6”依赖包。
3. 填写依赖包信息，输入依赖包信息，完成后单击“确定”。
   - 依赖包名称：输入“huaweicloudsdk_ecs_core_py3.6或者huaweicloudsdk_as_core_py3.6”。
   - 上传方式：选择“上传ZIP文件”。
   - 文件上传：选择需要上传的zip包文件。
   - 运行时语言：选择“Python3.6”。

图 14-3 创建依赖包

4. 用户进入as_graceful_shutdown函数详情页，在“代码”页签，单击页面最底部的“添加依赖包”。
5. 添加“huaweicloudsdk_ecs_core_py3.6”与“huaweicloudsdk_as_core_py3.6”依赖包。

14.4 配置消息通知

1. 登录“消息通知服务 SMN”服务控制台，选择“主题”，单击右上方的“创建主题”。
2. 输入主题信息，完成后单击“确定”。
   - 主题名称：“AStoFunctionGraph”
   - 企业项目：“default”（或者根据实际情况选择）

图 14-4 创建主题

3. 登录“弹性伸缩 AS”服务控制台，进入需要配置优雅关机的伸缩组的控制界面，选择“生命周期挂钩”，单击“添加生命周期挂钩”。

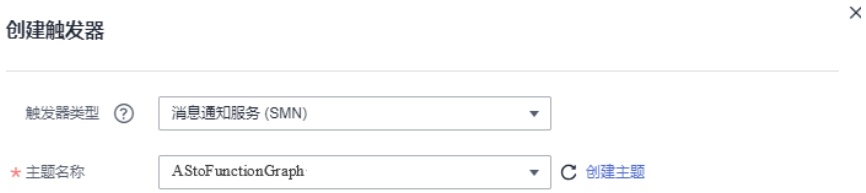
图 14-5 生命周期挂钩

4. 为伸缩组配置生命周期挂钩，完成后单击“确定”。这样缩容的实例就会被生命周期挂钩挂起，并发送消息通知至SMN主题。
   - 挂钩类型：“实例终止”。
   - 默认回调操作：“继续”。
   - 超时时间：“300”。
   - 通知主题：“AStoFunctionGraph”。

图 14-6 添加生命周期挂钩

5. 登录“函数工作流 FunctionGraph”服务控制台，进入函数“as_graceful_shutdown”详情页，在“设置 > 触发器”页签，单击“创建触发器”，弹出创建触发器界面。
6. 触发器类型选择SMN，主题名称选择SMN通知主题“AStoFunctionGraph”，这样SMN主题接收到的消息通知就会触发handler函数进行处理。

图 14-7 创建触发器



14.5 处理展示

触发伸缩活动

在伸缩组管理页面手动修改伸缩组的期望实例数，使其小于伸缩组的当前实例数，触发缩容活动。

查看伸缩实例为移出挂起状态，代表实例被生命周期成功挂起。

图 14-8 伸缩实例状态

触发函数运行

1. 在FunctionGraph的函数管理页面，选择监控-日志，观察FunctionGraph是否收到通知和函数的执行结果。

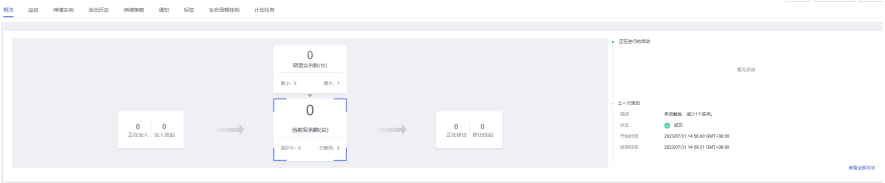
图 14-9 函数日志

2. 在伸缩组管理页面，选择伸缩实例，查看之前被挂起的伸缩实例状态为关机。

图 14-10 伸缩实例状态

3. 在伸缩组管理页面，等待伸缩组自动触发实例的移出和删除操作。

图 14-11 实例被移出并删除



15 使用 Go 构建 FunctionGraph HTTP 函数

方案概述

本章节主要指导使用Go语言开发应用的用户，将业务部署到FunctionGraph。

由于HTTP函数本身不支持Go语言直接代码部署，因此本章节将以转换成二进制的方式为例，将Go编写的程序部署到FuntionGraph上。

操作流程

构建代码包

创建源文件main.go，代码如下：

```
// main.go
package main

import (
    "fmt"
    "net/http"

    "github.com/emicklei/go-restful"
)

func registerServer() {
    fmt.Println("Running a Go Http server at localhost:8000/")

    ws := new(restful.WebService)
    ws.Path("/")

    ws.Route(ws.GET("/hello").To(Hello))
    c := restful.DefaultContainer
    c.Add(ws)
    fmt.Println(http.ListenAndServe(":8000", c))
}

func Hello(req *restful.Request, resp *restful.Response) {
    resp.Write([]byte("nice to meet you"))
}

func main() {
    registerServer()
}
# bootstrap
/opt/function/code/go-http-demo
```

在main.go中，使用**8000**端口启动了一个HTTP服务器，并注册了path为“/hello”的API，调用该API将返回"nice to meet you"。

编译打包

1. 在**linux**机器下，将上述代码编译 **go build -o go-http-demo main.go**。然后，将go-http-demo和bootstrap打包为xxx.zip。
2. 在**windows**机器下使用Golang编译器完成打包，具体步骤如下：

```
# 切换编译环境方式
# 查看之前的golang编译环境
go env
# 设置成linux对应的
set GOARCH=amd64
go env -w GOARCH=amd64
set GOOS=linux
go env -w GOOS=linux

# go build -o [目标可执行程序] [源程序]
# 例子
go build -o go-http-demo main.go

# 还原之前的编译环境
set GOARCH=amd64
go env -w GOARCH=amd64
set GOOS=windows
go env -w GOOS=windows
```

创建HTTP函数并上传代码

创建1个HTTP函数，并上传已打包的xxx.zip包。请参见**创建HTTP函数**。

创建APIG触发器

请参见**使用APIG触发器**，创建APIG触发器，“安全认证”建议选择“None”，方便调试。

图 15-1 APIG 触发器

调用测试

将刚才创建的APIG触发器的URL+代码中注册的“/hello”复制到浏览器地址栏，可以看到页面返回结果如下：

图 15-2 请求结果