最佳实践

文档版本 01

发布日期 2025-12-04

华为技术有限公司

商标声明


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。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 FunctionGraph 最佳实践汇总

本文汇总了基于函数工作流服务（FunctionGraph）常见应用场景的操作实践，为每个实践提供详细的方案描述和操作指导，帮助您轻松构建基于函数工作流的业务。

性能优化与安全类实践

表 1-1 FunctionGraph 性能优化与安全类实践

最佳实践	说明
2.1 FunctionGraph性能优化实践	本章节旨在探讨FunctionGraph性能优化的最新趋实践，从冷启动优化、函数执行优化等方面，全面剖析如何在不同场景下实现最优性能，为您提供实用的指导，帮助您在FunctionGraph上构建更高效、更稳定的应用。
2.2 FunctionGraph冷启动优化实践	本章节介绍如何优化函数的冷启动时间，提高在构建无服务器架构时的使用体验。
2.3 FunctionGraph安全最佳实践	本章节介绍FunctionGraph使用过程中的安全最佳实践，提供可操作的规范性指导以提升整体安全能力。

数据处理类实践

表 1-2 FunctionGraph 数据处理类最佳实践

最佳实践	说明
3.3 使用FunctionGraph函数对DIS数据进行格式转换并存储到CloudTable	本章节介绍如何使用函数结合数据接入服务（DIS）采集IOT实时数据流，并将采集到的数据进行格式转换，存储到表格存储服务（CloudTable Service）中。
3.4 使用FunctionGraph函数实现通过API方式上传文件	本章节以NodeJS和Python语言为例，介绍如何配置后端解析函数，结合APIG处理端侧文件上传云服务器。 适用于例如服务运行日志的上报、Web应用图片上传等Web和App应用的场景。

最佳实践	说明
3.5 使用FunctionGraph函数对IoTDA中的设备坐标数据进行转换	本章节介绍如何使用函数结合IoTDA服务，将物联网设备上报以及设备状态变动的相关数据，流转至FunctionGraph触发函数运行并进行坐标转换（将WGS84坐标转为GCJ02坐标）。 适用于例如将设备上报的数据处理后存储至如OBS；对上报的数据进行结构化，清洗后存储至数据库；根据设备状态变化进行事件通知等应用场景。
3.6 使用FunctionGraph函数对OBS中的文件进行加解密	本章节介绍如何使用函数结合华为云DEW，配合使用“OBS应用事件源”触发器实现对OBS中的文件进行加解密处理。
3.7 使用FunctionGraph函数识别LTS中的异常业务日志并存储到OBS	本章节介绍如何结合云日志服务LTS，配置提取告警日志功能的函数，识别LTS中的异常日志数据存储至OBS桶，再通过消息通知服务SMN推送告警短信和邮件，通知业务人员处理。
3.8 使用FunctionGraph函数对LTS中的日志进行实时过滤	本章节介绍如何结合云日志服务LTS，配置提取日志数据并分析和过滤关键信息后转储至LTS的函数。
3.9 使用FunctionGraph函数流对OBS中的图片进行旋转	本章节介绍如何使用函数流功能，编排函数自动化处理OBS中的图片进行旋转。 函数流功能目前支持“华东-上海一”、“亚太-新加坡”。
3.10 使用FunctionGraph函数流对图片进行压缩和打水印	本章节介绍如何使用函数流功能，编排函数自动化对图片进行压缩和打水印处理。 函数流功能目前支持“华东-上海一”、“亚太-新加坡”。

功能应用类实践

表 1-3 FunctionGraph 功能应用类最佳实践

最佳实践	说明
4.1 使用FunctionGraph函数和CTS识别非法IP的登录登出操作	本章节介绍如何结合云审计服务CTS，配置获取云服务资源操作信息并对信息进行分析和处理的函数，再通过消息通知服务SMN推送告警短信和邮件，通知业务人员处理。
4.2 使用FunctionGraph函数作为后端实现APIG的自定义认证能力	本章节介绍如何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。
4.3 使用FunctionGraph HTTP函数处理gRPC请求	本章节指导用户使用gRPC，在FunctionGraph中处理gRPC请求。 目前仅支持“拉美-圣地亚哥”区域。

最佳实践	说明
4.4 使用FunctionGraph的Java函数配置Log4j2实现日志打印	本章节介绍如何使用Java函数配置Log4j2，实现日志打印功能。
4.5 使用FunctionGraph部署AI绘画Stable Diffusion应用	本章节介绍如何通过FunctionGraph的应用中心，部署AI绘画Stable-Diffusion应用，并提供多种自定义使用AI绘画应用的方法。
4.6 使用FunctionGraph快速部署MCP Server	本章节介绍如何通过FunctionGraph的应用中心，一键部署热门开源MCP Server，并通过API网关（APIG）对外提供服务。

函数构建类实践

表 1-4 FunctionGraph 函数构建类最佳实践

最佳实践	说明
5.1 使用已有SpringBoot项目构建HTTP函数	本章节指导使用Springboot开发应用的用户，将业务通过构建HTTP函数的方式部署到FunctionGraph。
5.2 使用Go语言程序构建HTTP函数	本章节指导使用Go语言开发应用的用户，将业务通过构建HTTP函数的方式部署到FunctionGraph。
5.3 使用FunctionGraph函数访问RDS for MySQL	本章节介绍如何使用FunctionGraph函数高可靠地访问RDS for MySQL，并进行数据查询操作，同时提供示例代码供测试使用。

2 性能优化与安全类实践

2.1 FunctionGraph 性能优化实践

在Serverless技术日益普及的今天，性能优化已成为提升应用效率与用户体验的关键。

本篇旨在探讨FunctionGraph性能优化的最新实践，从**冷启动优化**、函数执行优化等方面，全面分析如何在不同场景下实现最优性能，为您提供实用的指导，帮助您在FunctionGraph上构建更高效、更稳定的应用。

代码优化

- 编写幂等代码。为函数编写幂等代码可确保函数以相同的方式处理重复事件。
- 合理使用连接池。保持连接复用，以减少新建连接的冷启动开销（如HTTP连接池、数据库连接池、Redis连接池等）。
- 避免每次调用时重新初始化变量对象（使用全局静态变量、单例等）。在函数调用中间件（如Redis、Kafka等）时，避免在handler方法中重复初始化client，应通过init方法或全局变量初始化client，以减少client的冷启动开销。
- 加强客户端异常重试机制。当调用函数返回非200状态码时（如500、429、504等），客户端依据具体业务需求，加入重试逻辑，能够进一步确保业务的可靠性。
- 使用恰当的日志记录。FunctionGraph函数中访问第三方服务、华为云服务和执行相关操作时，应记录日志，以便于后续的异常定位、性能优化及业务分析。

性能压测

对函数进行性能测试是确保选择最优配置的关键环节。在函数压测过程中，可利用平台提供的指标、日志、调用链等手段进一步分析函数性能数据，从而优化函数配置选择。具体可观测指标详情请参考**函数监控概述**。

精简代码和镜像瘦身

由于FunctionGraph在冷启动时会下载函数代码，下载代码的过程会影响启动时间。如果代码文件过大，下载时间将延长，导致FunctionGraph的启动时间增加。

如果使用自定义镜像函数，镜像越大，启动时间也会越长。因此，为了减少冷启动时间，应对应用程序进行优化，例如移除程序中不必要的代码、减少第三方库的依赖等。具体操作包括：在Node.js中执行“npm prune”命令，在Python中执行“autoflake”。

此外，某些第三方库中可能包括测试用例源代码、无用的二进制文件和数据文件等，清理这些文件可减少函数代码的下载和解压时间。

使用更大的内存

为函数配置较大内存可以提升CPU性能，从而加快函数启动和执行速度。您可以通过监测函数执行时间，评估不同内存配置对函数性能影响，进而选择最优内存大小。

具体监控信息请参考**监控指标说明**。配置内存的操作步骤请参考**配置函数信息**。

使用公共依赖包加速

在编写函数代码时，通常会引入第三方依赖库，特别是使用Python语言构建函数时。在冷启动过程，系统会下载所需依赖包，如果依赖包体积过大，会延长启动时间。

FunctionGraph提供公共依赖包和私有依赖包两种依赖包。使用公共依赖包时，FunctionGraph会预先将其下载至执行节点中，以减少依赖包的下载时间。因此，建议优先使用FunctionGraph提供的公共依赖包，尽量减少私有依赖的使用。依赖包的相关介绍请参考**配置函数依赖包**。

配置预留实例

预留实例创建完成后，将自动加载该函数的代码、依赖包及执行初始化入口函数，并持续驻留环境。因此，为函数配置预留实例能够避免冷启动导致的时延问题。函数预留实例的配置操作请参考**预留实例管理**。

使用函数初始化入口

对于需要频繁调用的函数，将初始化逻辑置于初始化入口可显著减少每次执行时间，例如初始化HTTP连接、初始化数据库连接等。函数初始化入口的配置操作请参考**配置函数初始化**。

2.2 FunctionGraph 冷启动优化实践

Serverless按需付费、自动弹性伸缩、屏蔽复杂性等特征使其逐渐成为下一代云计算新范式。但是在Serverless架构带来极大便利的同时，在实时性要求较高的应用场景下，冷启动将是面临的一个切实的挑战。当使用Serverless构建Web服务时，冷启动和Web服务初始化时间一共超过了5秒钟，那么无疑将会使您网站的用户体验大打折扣，因此设法减少冷启动时间，提高使用体验，是您在构建无服务器架构时亟待解决的问题。

Serverless实例的生命周期可以分为三个阶段：

- **初始化**：在此阶段，FunctionGraph会尝试解冻之前的执行环境，若没有可解冻的环境，FunctionGraph会进行资源创建，下载函数代码，初始化扩展和Runtime，然后开始运行初始化代码（主程序外的代码）。

- **执行**：在此阶段，实例接收事件后开始执行函数。函数运行到完成后，实例会等待下个事件的调用。
- **关闭**：如果FunctionGraph函数在一段时间内没有接收任何调用，则会触发此阶段。在关闭阶段，Runtime关闭，然后向每个扩展发送一个关闭事件，最后删除环境。

当触发FunctionGraph时，若当前没有处于激活阶段的函数实例可供调用，则会下载函数的代码并创建一个函数的执行环境。从事件触发到新的FunctionGraph环境创建完成这个周期通常称为“冷启动时间”。在Serverless架构中，冷启动问题是无法避免的。

目前FunctionGraph已经对系统侧的冷启动做了大量优化，针对用户侧请参考如下方案。

选择合适的内存

在请求并发量一定的情况下，函数内存越大，分配的CPU资源相应越多，一般冷启动表现越优。

快照冷启动

Java应用冷启动速度慢的问题尤为突出。华为云FunctionGraph创新提出的基于进程级快照的冷启动加速解决方案，致力于在用户无感知（无需/少量进行代码适配）的前提下，帮助用户突破冷启动的性能瓶颈。本优化方案直接从应用初始化后的快照进行运行环境恢复，跳过复杂的框架、业务初始化阶段，从而显著降低Java应用的启动时延，实测性能提升达90%+。

用户使用Java函数可以打开冷启动快照加速的配置开关，详情请参见**配置快照式冷启动**。华为云FunctionGraph会预先执行函数对应的初始化代码，获取其初始化执行上下文环境的快照，并进行加密缓存。后续调用该函数并触发冷启动扩容时，会直接从提前初始化后的应用快照来恢复执行环境，而非重新走一遍初始化流程，以此达到极大提升启动性能的效果。

精简代码大小和镜像瘦身

由于FunctionGraph在冷启动的时候会下载函数代码，下载代码的过程也会影响启动时间。如果代码包太大，下载时间将会变长，导致增加FunctionGraph的启动时间；如果使用自定义镜像函数，镜像越大，启动时间也会越长。所以，为了降低冷启动时间，可以对应用程序进行瘦身，比如在程序中移除不必要的代码、减少不必要的第三方库依赖等。例如，在Node.js中执行“npm prune”命令、在Python中执行“autoflake”。另外，某些第三方库中可能会包含测试用例源代码、无用的二进制文件和数据文件等，删除无用文件可以降低函数代码下载和解压时间。

公共依赖包加速

在编写应用程序时，通常会引入第三方依赖库，尤其是Python语言。在冷启动过程中会下载所需的依赖包，若依赖包太大会直接增加启动时间。FunctionGraph提供公共依赖包和私有依赖包两种模式，针对公共依赖包，FunctionGraph会预先下载到执行节点中，减少依赖包的下载时间。所以建议优先使用FunctionGraph提供的公共依赖包，尽量减少私有依赖的使用。

预热

在事件触发函数时，若此时有处于激活状态的函数实例可被调用，那么就可以避免冷启动，降低响应时间。可以使用以下两种方式预热：

- 使用定时触发器预热函数，具体使用介绍请参见**使用定时触发器**。
- 使用预留实例避免冷启动，具体使用介绍请参见**预留实例管理**。

2.3 FunctionGraph 安全最佳实践

安全性是华为云与您的共同责任。华为云负责云服务自身的安全，提供安全的云环境。您需要合理利用云服务提供的安全功能保护数据，确保安全地使用云服务。详情请参见**责任共担**。

本文提供了FunctionGraph使用过程中的安全最佳实践，旨在提供可操作的规范性指导以提升整体安全能力。依据该指导文档，能够持续评估函数的安全状态，有效结合FunctionGraph提供的多种安全功能，增强FunctionGraph的整体安全防御能力，确保存储于FunctionGraph的数据不被泄露或篡改，同时保障数据传输过程中的安全。

本文从以下维度提供建议，您可以据此评估FunctionGraph的使用情况，并根据业务需求在此基础上进行安全配置。

使用可信的代码和依赖，避免代码漏洞

- 在部署函数代码前，建议使用华为云**CodeArts Check**的代码检测功能，对代码进行静态扫描和漏洞分析，确保代码安全。
- 使用可靠来源的依赖库并定期更新，避免使用存在已知漏洞的第三方库。

保护敏感信息，防止敏感信息泄露

- 如果用户代码或配置中包含敏感信息，如AK/SK，token，密码等，强烈建议采用**加密环境变量**，否则这些信息可能在用户界面或API返回结果中以明文形式显示，从而导致敏感信息泄露。
- 对于涉及用户隐私数据（日志、个人信息等），建议在函数处理过程中进行脱敏处理，不可通过日志明文打印，以防敏感信息泄露。
- FunctionGraph可向用户提供临时代码与下载地址，并设定有效期。用户应防止临时下载地址泄露，以降低代码或库泄露的风险。

精细化权限控制和开启身份认证

- 通过华为云统一身份认证服务（IAM），为FunctionGraph函数**配置委托权限**和AK/SK等时，应遵循最小权限使用原则，确保函数仅能访问指定的资源。例如，限制函数对特定OBS桶的读写权限，防止越权访问。
- 配置APIG触发器时，建议启用IAM认证或自定义认证，确保仅授权请求能触发函数执行。此外，可通过APIG实施流量控制，防止恶意请求导致资源耗尽。

为函数配置 VPC，防止外部攻击

当用户函数需要访问华为云虚拟私有云（VPC）内的资源，如RDS时，建议为函数**配置VPC**，以确保函数与其他云服务之间的通信在隔离的网络环境中进行。

使用函数版本控制，迅速更新与回退

FunctionGraph支持对函数进行**版本管理**，建议为每个函数创建若干版本，并在生产环境中使用稳定版本。同时，通过**别名**功能，关联指定版本的函数实现版本切换，以确保在出现安全问题时能够迅速回滚。

3 数据处理类实践

3.1 使用 FunctionGraph 函数对 OBS 中的图片进行压缩

方案概述

本实践适用于将单张或批量的图片压缩处理。高质量图片文件通常占用大量存储空间和带宽，导致网站和应用的加载速度变慢，结合对象存储服务OBS，使用FunctionGraph函数可以构建一个高效的图片压缩解决方案，对存储在OBS桶中的图片进行自动压缩处理，实现存储空间优化与资源高效利用。

约束与限制

- “OBS应用事件源”触发器目前仅支持“华北-北京四”、“华北-乌兰察布一”、“华东-上海一”区域，在创建函数和OBS桶时请选择上述区域之一。
- 在操作过程中，所创建的函数和OBS桶必须位于同一个区域下。

资源和成本规划

表1介绍使用FunctionGraph函数对OBS中的图片进行压缩所需的资源和成本规划。

表 3-1 资源和成本规划

资源	资源说明	计费说明
对象存储服务 OBS	• 产品类型：对象储存 • 区域：华北-北京四 • 存储策略：单AZ存储 • 存储类别：标准存储 • 桶策略：私有 • 购买量：2	• 计费模式：本例使用按需计费。 • 具体计费项及说明请参考**对象存储服务按需计费说明**。
函数工作流 FunctionGraph	• 函数类型：事件函数 • 区域：华北-北京四 • 购买量：1	• 计费模式：按需计费。 • 函数工作流提供免费试用，每月前100万次调用免费。具体计费项及说明请参考**函数工作流按需计费说明**。

操作流程

下表介绍使用FunctionGraph函数对OBS中的图片进行压缩的总体操作流程。

表 3-2 操作流程

操作流程	说明
步骤一：创建两个OBS桶	创建两个OBS桶，源桶用于存储原始图片，目标桶用于存储压缩后图片。
步骤二：创建云服务委托	创建云服务委托，授权FunctionGraph使用其他云服务，确保能与OBS服务协同工作。
步骤三：创建图片压缩函数	创建空白函数，配置代码环境和创建OBS应用事件源触发器，以实现对OBS源桶中上传或更新的图片自动进行压缩。
步骤四：验证图片压缩	上传原始图片到OBS源桶，图片压缩函数自动执行，在OBS目标桶中生成压缩后的图片，验证图片压缩函数正常运行。

步骤一：创建两个 OBS 桶

步骤1 登录**对象存储服务控制台**，进入“对象存储”页签。

步骤2 单击“创建桶”，进入“创建桶”界面。

步骤3 在“创建桶”界面，参考**表3-3**填写OBS源桶信息。

表 3-3 OBS 源桶配置

参数名称	参数说明	取值样例
区域	必选参数。 桶所属区域。请选择靠近您业务的区域，以降低网络时延，提高访问速度。桶创建成功后，不支持变更区域。当前“OBS应用事件源”触发器仅支持“华北-北京四”、“华北-乌兰察布一”、“华东-上海一”区域。	华北-北京四
桶名称	必选参数。 桶的名称。需全局唯一，桶创建成功后，不支持修改名称。	your-bucket-input
数据冗余存储策略	必选参数。 ● 多AZ存储：数据冗余存储至多个可用区（AZ），可靠性更高。 ● 单AZ存储：数据仅存储在单个可用区（AZ），成本更低。 ● 桶创建成功后，不支持变更存储策略。	单AZ存储
存储类别	必选参数。 ● 标准存储：适用于有大量热点文件或小文件，且需要频繁访问（平均一个月多次）并快速获取数据的业务场景，支持单AZ存储和多AZ存储。 ● 低频访问存储：适用于不频繁访问（平均一年少于12次），但需要快速获取数据的业务场景，支持单AZ存储和多AZ存储。 ● 归档存储：适用于很少访问（平均一年一次），且对数据获取速率要求不高的业务场景，仅支持单AZ存储。	标准存储
桶策略	必选参数。 桶的读写权限控制。 ● 私有：除桶ACL授权外的其他用户无桶的访问权限。 ● 公共读：任何用户都可以对桶内对象进行读操作。 ● 公共读写：任何用户都可以对桶内对象进行读/写/删除操作。	私有
企业项目	必选参数。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，默认项目为default。	default

参数名称	参数说明	取值样例
功能配置	可选参数。 关于功能配置的具体说明可参考**创建桶**。 • 归档数据直读：通过归档数据直读，可以直接下载存储类别为归档存储的数据，无需提前恢复。归档数据直读会收取相应的费用。 • 服务端加密：服务端加密是指OBS服务端对客户端上传到OBS的对象进行加密存储，开启后需要选择加密密钥。 • WORM：开启WORM (一次写入多次读取) 功能后，当前桶支持配置保留策略，受保留策略保护的对象版本在指定时间段内不能被删除。 • 标签：标签用于标识OBS中的桶，以此达到对OBS中的桶进行分类的目的。	• 归档数据直读：未开启 • 服务端加密：未开启 • WORM：未开启 • 标签：-

步骤4 重复步骤**步骤3**，创建OBS目标桶。桶名称命名为“your-bucket-output”，其余参数信息与源桶保持一致。

步骤5 完成桶创建后，桶列表有your-bucket-input、your-bucket-output两个桶。

----结束

步骤二：创建云服务委托

步骤1 登录**统一身份认证服务控制台**，左侧导航栏选择“委托”，进入“委托”页面后，右上角单击“创建委托”。

步骤2 在“创建委托”页面，配置如下参数：

- 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：保持默认，无需填写。

步骤3 单击“完成”，系统提示创建成功，单击“立即授权”，进入“授权”界面。

步骤4 在“选择策略”界面搜索并勾选“OBS Administrator”，单击“下一步”。

图 3-1 选择策略

步骤5 在“设置最小授权范围”界面选择“所有资源”，单击“确定”。

说明

“OBS Administrator”策略暂不支持指定区域项目资源。

步骤6 系统提示授权成功，单击“完成”，回到“委托”列表，列表中出现“serverless_trust”委托则创建成功。

----结束

步骤三：创建图片压缩函数

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”，进入创建函数界面。

步骤3 选择“创建空白函数”，参考表3-4填写函数基本信息，填写完成后单击“创建函数”。

表 3-4 配置函数参数

参数名称	参数说明	取值样例
函数类型	必选参数。 • 事件函数：通过触发器来触发函数执行。 • HTTP函数：用户可以直接发送 HTTP 请求到 URL 触发函数执行。	事件函数
区域	必选参数。 代码部署的区域。需与OBS桶区域保持一致。	华北-北京四
函数名称	必选参数。 函数的名称。可包含字母、数字、下划线和中划线，以大/小写字母开头，以字母或数字结尾，长度不超过60个字符。	fss_examples_image_thumbnail
企业项目	必选参数。 将函数添加至已创建的企业项目中，需与OBS桶的企业项目保持一致，默认项目为default。	default
委托名称	可选参数，本实践必选。 委托函数工作流服务访问其他云服务的委托名称。选择步骤二：创建云服务委托中创建的委托。	serverless_trust
运行时	必选参数。 编写函数的语言及其版本。控制台代码编辑器仅支持Node.js、Python和PHP语言。	Python3.6

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

1. 下载示例代码fss_examples_image_thumbnail_eg.zip。
2. 在“代码”页签，选择“上传代码 > Zip文件”，添加下载的“fss_examples_image_thumbnail_eg.zip”文件，单击“确定”，代码自动部署。

3. 单击页面最底部的“添加依赖包”，添加公共依赖包“pillow-7.1.2”，版本默认选择“1”，单击“确定”。
4. 在“设置 > 常规设置”页签，修改如下配置。
   - 执行超时时间：输入“40”。
   - 内存：选择“256”。

   填写完成后单击“保存”。
5. 在“设置 > 环境变量”页签，单击“编辑环境变量”，在弹出窗口中单击“添加环境变量”，添加表3-5信息，填写完成后单击“确定”。

表 3-5 环境变量

键	值	说明
output_bucket	your-bucket-output	存储压缩图片的OBS目标桶名称。
obs_endpoint	obs.cn-north-4.myhuaweicloud.com	华北-北京四区域的OBS服务终端节点（其他区域节点参考地区和终端节点）。

6. 在“设置 > 触发器”页签，单击“创建触发器”，弹出创建触发器界面，参考表3-6填写触发器基本信息，填写完成后单击“确定”。

表 3-6 配置触发器参数

参数名称	参数说明	取值样例
触发器类型	必选参数。 本实践通过添加OBS应用事件源触发器，当对OBS桶执行操作时，将生成事件触发函数执行。	OBS应用事件源
触发器名称	必选参数。 触发器的名称。只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。	Image
桶	必选参数。 选择已创建的OBS源桶，用于存储原始图片。	your-bucket-input
事件类型	必选参数。 触发事件的类型，本实践通过上传或更新桶对象触发函数执行。	“通过页面或Put请求创建或覆盖桶对象。”、“使用Post请求创建或覆盖桶对象。”
对象名前缀	可选参数。 用来限制以此关键字开头的对象的事件通知，该限制可以实现对OBS对象名的过滤。	保持默认，不填写。

参数名称	参数说明	取值样例
对象名后缀	可选参数。 用来限制以此关键字结尾的对象的事件通知，该限制可以实现对OBS对象名的过滤。	保持默认，不填写。
对象名编码	必选参数。 选择是否对对象名进行编码。	默认开启。

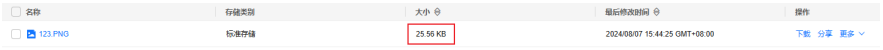
----结束

步骤四：验证图片压缩

步骤1 进入**对象存储服务控制台**，单击“your-bucket-input”桶，进入“对象”页签。

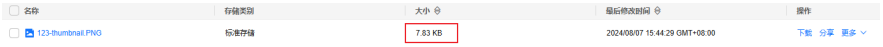
步骤2 单击“上传对象”，存储类别选择“标准存储”，添加待压缩图片后单击“确定”，上传成功后如**图3-2**所示。

图 3-2 上传图片



步骤3 进入“your-bucket-output”桶对象界面，查看压缩后的图片大小。

图 3-3 压缩图片



须知

为避免产生不必要的存储费用，完成本次体验后，依据实际使用需求，可删除两个OBS桶中存储的图片。请注意，删除后无法恢复，请谨慎操作。

----结束

3.2 使用 FunctionGraph 函数为 OBS 中的图片打水印

3.2.1 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。（当前“OBS应用事件源”仅支持华北-北京四、华北-乌兰察布一、华东-上海一。）

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应用事件源。
- 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应用事件源，用来关联函数和OBS桶

3.2.2 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：输入您自定义的桶名称，此处以“bucket-input-fg”为例。
- 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“bucket-output-fg”。

步骤4 完成桶创建以后，OBS桶列表有bucket-input-fg、bucket-output-fg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：输入您自定义的委托名称，此处以“serverless_trust”为例。
- 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“OBS Administrator”。

步骤5 单击“下一步”，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

3.2.3 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3-4**所示。

图 3-4 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 选择“创建空白函数”，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”。

- 函数类型：事件函数
- 函数名称：输入您自定义的函数名称，此处以“fss_examples_image_watermark”为例。
- 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 3-5 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。
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。

表 3-7 环境变量

键	值	说明
obs_output_bucket	bucket-output-fg	键：index.py文件中定义的存放输出水印图片的OBS桶参数。 值：**创建OBS桶**中创建的存放输出水印图片的OBS桶。
obs_region	例如“cn-north-4”	键：index.py文件中定义的存放输出水印图片的OBS桶的区域，应与函数所属区域保持一致。 值：OBS桶obs_output_bucket所在的区域，更多Region区域详情请参见**地区和终端节点**。

----结束

3.2.4 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应用事件源实现的，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“OBS应用事件源”，填写触发器配置信息，如**图3-6**所示。

- 触发器名称：自定义。
- 桶：选择**创建OBS桶**中创建的“bucket-input-fg”桶。
- 事件类型：选择“通过页面或Put请求创建或覆盖桶对象。”、“使用Post请求创建或覆盖桶对象。”。

图 3-6 创建 OBS 应用事件源

步骤3 单击“确定”，完成触发器创建。

说明

OBS应用事件源创建以后，当有图片上传或更新至bucket-input-fg桶时，生成事件，触发函数执行。

----结束

3.2.5 处理图片

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

上传图片生成事件

登录**对象存储服务控制台**，进入bucket-input-fg桶对象界面，上传image.jpg图片，如**图3-7**所示。

图 3-7 上传图片

触发函数自动运行

上传图片至bucket-input-fg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bucket-output-fg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bucket-output-fg桶对象界面，可以看到输出的图片image.jpg，如图3-8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3-9所示。

图 3-8 输出图片

图 3-9 效果图

3.3 使用 FunctionGraph 函数对 DIS 数据进行格式转换并存储到 CloudTable

3.3.1 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。

3.3.2 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。

基本信息中输入您自定义的名称，此处以“vpc-cloudtable”为例，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

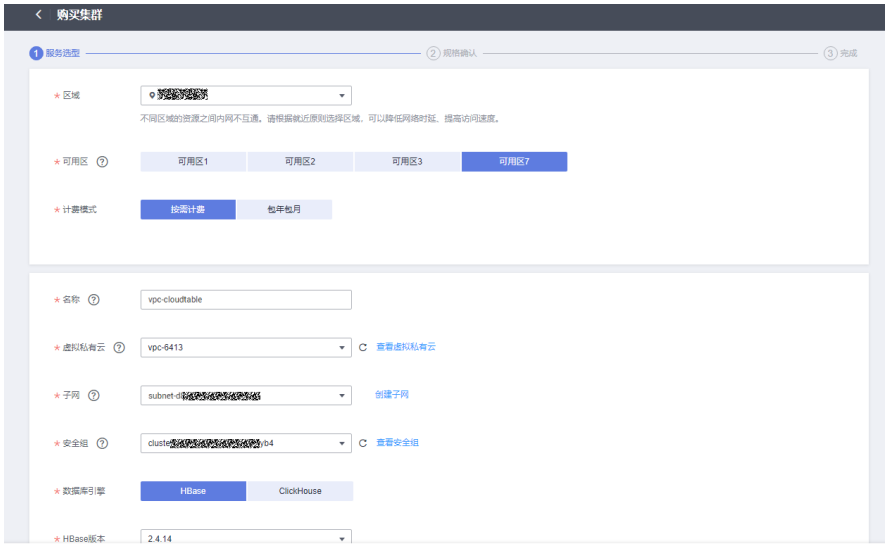
创建集群

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。

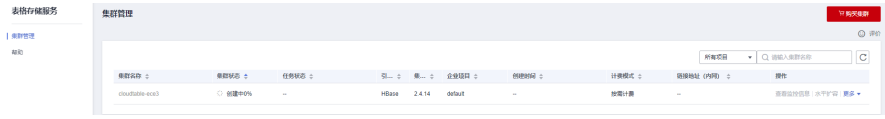
- 区域：使用系统默认。
- 名称：输入您自定义的名称，此处以“cloudtable-dis”为例。
- 虚拟私有云：选择**创建虚拟私有云**中创建的“vpc-cloudtable”。
- 其他配置保持默认，无需修改。

图 3-10 购买集群



步骤3 确认配置信息无误，单击“提交”，创建集群。

图 3-11 创建集群



说明

创建集群需要较长时间，可以从图3-11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：输入您自定义的委托名称，此处以“DISDemo”为例。
- 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“DIS Administrator”和“cloudtable Administrator”。

说明

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

步骤5 单击“下一步”，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

3.3.3 构建程序

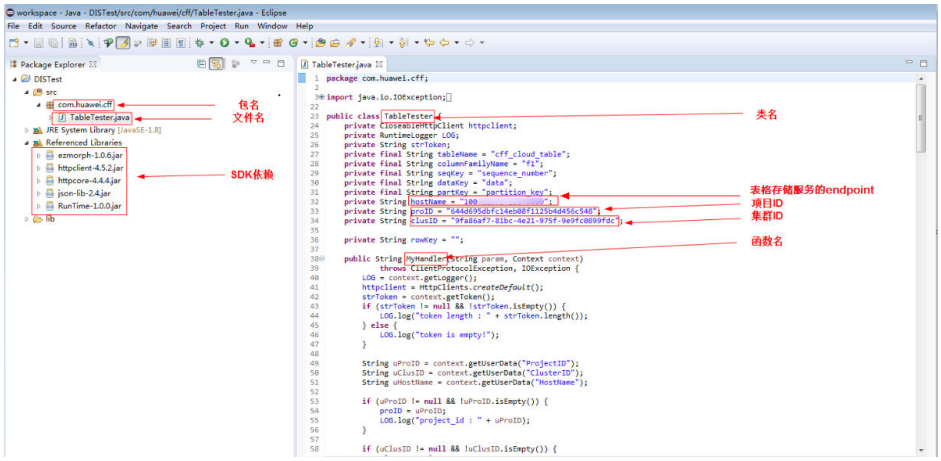
本例提供了DIS数据流格式转换的**源码**和**程序包**（包含函数依赖），使用空白模板创建函数，用户可以下载、学习使用。

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3-12**所示。

图 3-12 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3-13**所示，在“项目列表”获得项目ID，如**图3-14**所示。

图 3-13 我的凭证

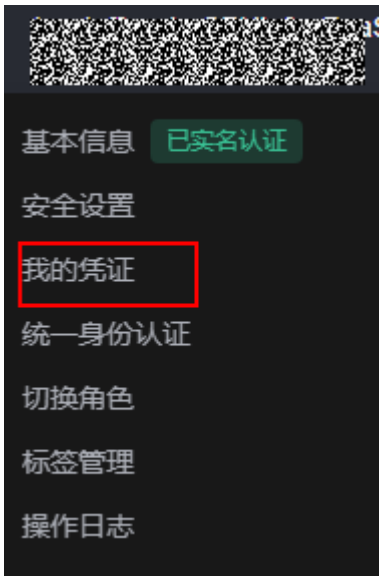
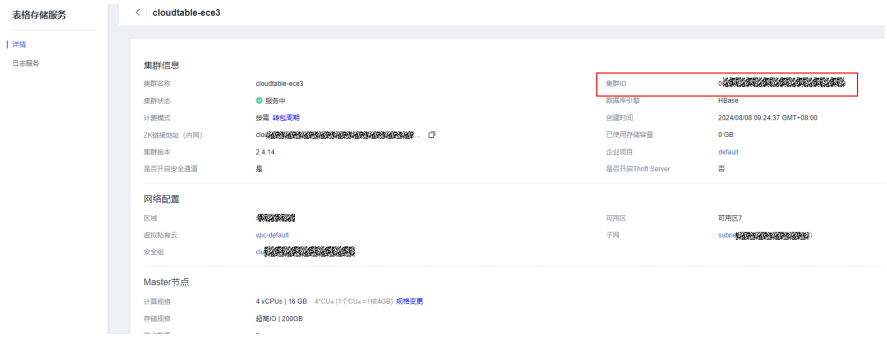


图 3-14 项目 ID

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3-15**所示。

图 3-15 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 3-16 Export

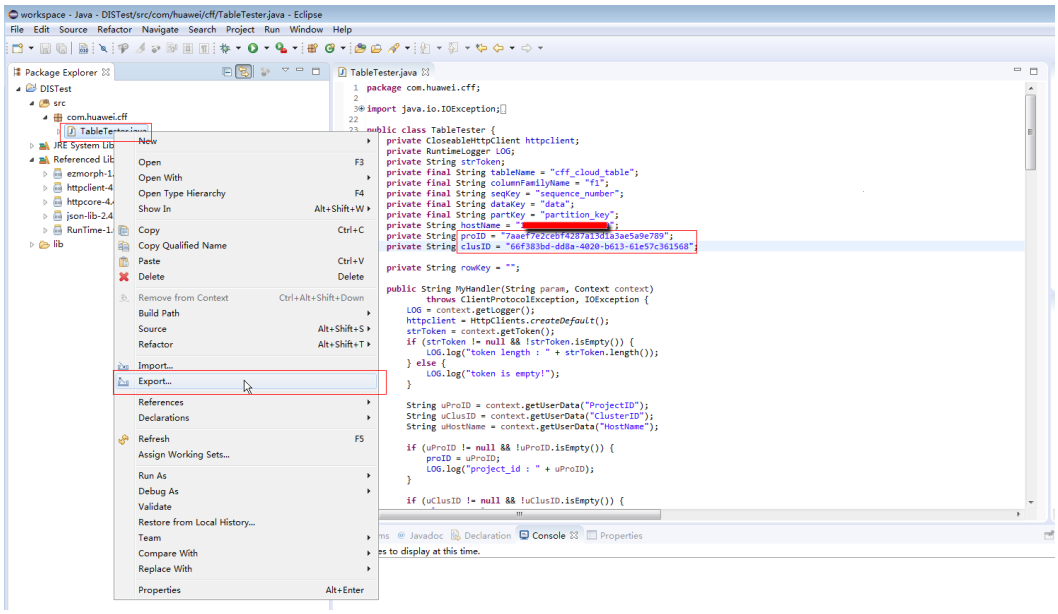


图 3-17 选择类型

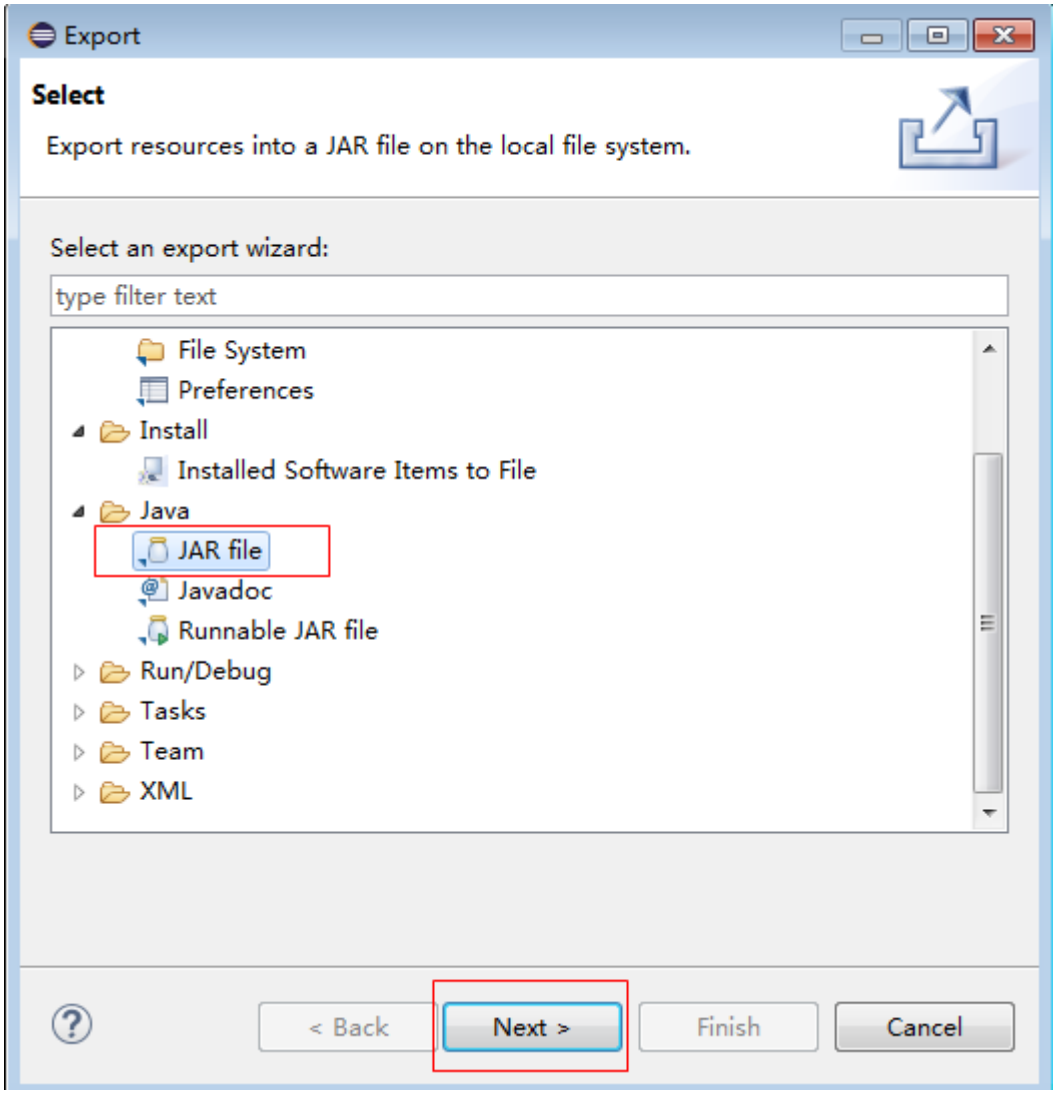
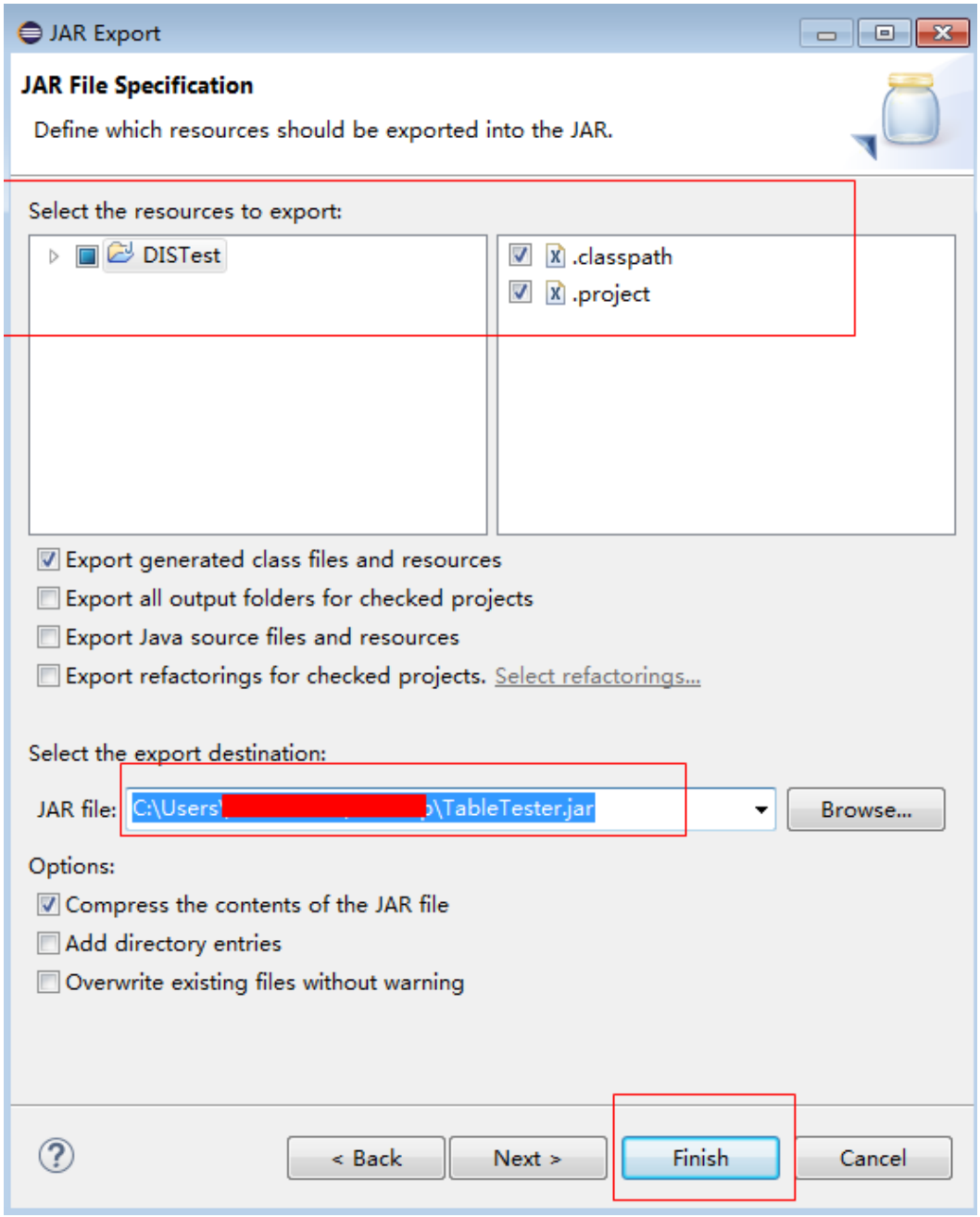


图 3-18 发布



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3-19**所示。使用Table Tester.jar替换DIS Test.jar，替换文件目录后如**图3-20**所示。打ZIP包，如**图3-21**所示，得到disdemo.zip文件。

图 3-19 文件目录

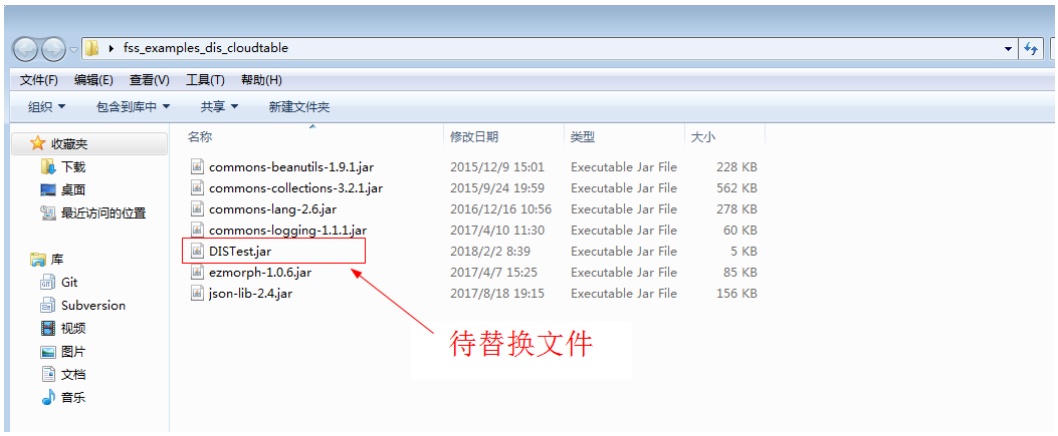


图 3-20 替换后文件目录

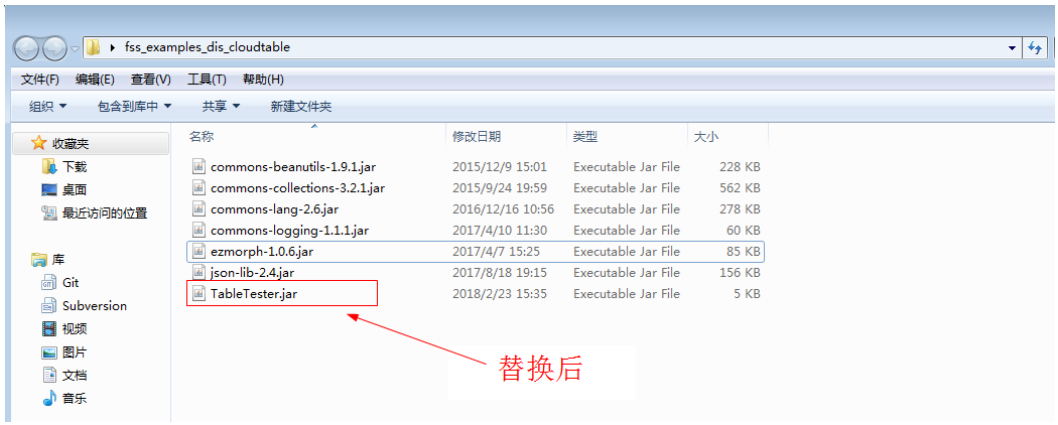
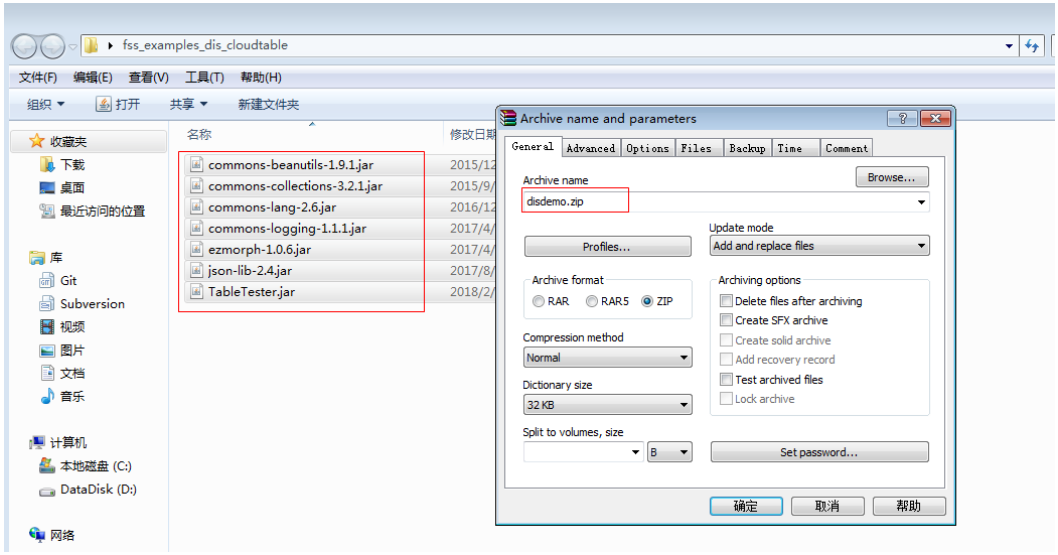


图 3-21 打 ZIP 包



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 选择“创建空白函数”，填写函数基本信息，完成后单击“创建函数”。

- 函数类型：事件函数。
- 函数名称：输入您自定义的函数名称，此处以“DISDemo”为例。
- 委托名称：选择3.3.2 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”。

----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。

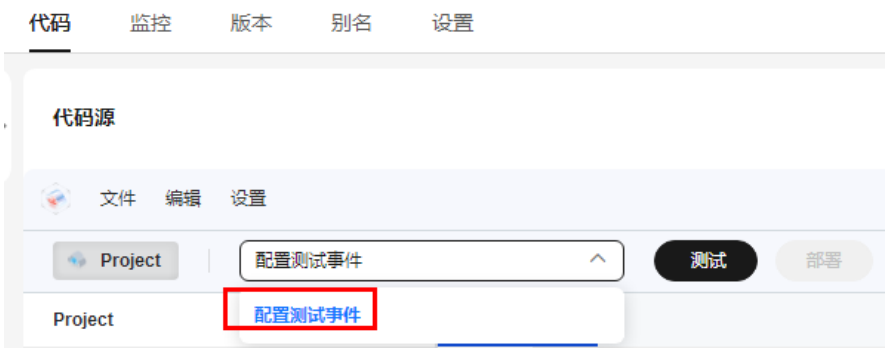
----结束

3.3.4 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3-22所示，弹出“配置测试事件页”。

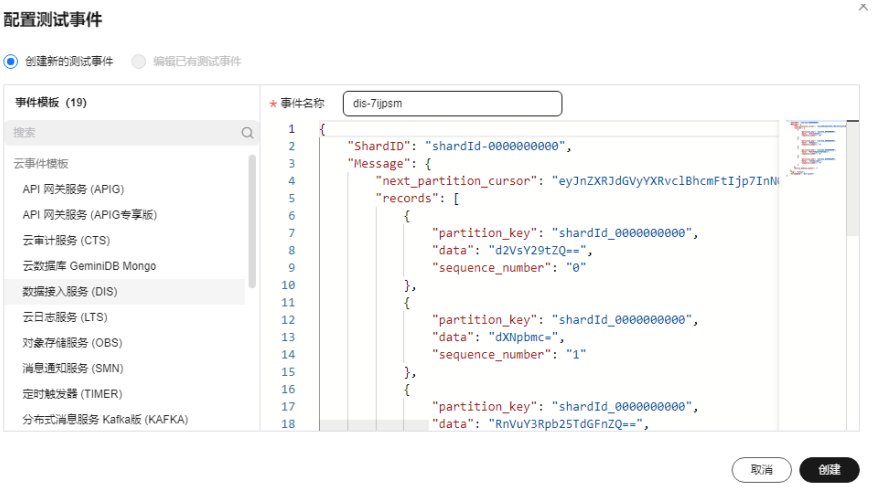
图 3-22 配置测试事件



步骤2 在“配置测试事件页”，输入配置信息，如图3-23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：输入您自定义的事件名称，此处以“dis-test”为例。

图 3-23 测试事件



步骤3 单击“创建”，完成测试事件配置。

----结束

3.3.5 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3-24所示。

图 3-24 配置测试事件

步骤2 函数执行成功后，部分函数日志如图3-25所示，全部的日志信息，可以到“日志”页签查询。

图 3-25 函数执行结果

----结束

3.4 使用 FunctionGraph 函数实现通过 API 方式上传文件

3.4.1 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。

3.4.2 资源规划

表 3-8 资源规划

产品	配置示例
API网关APIG	• 区域：新加坡 • 规格：专享版APIG实例
函数工作流 FunctionGraph	• 区域：新加坡 • 计费模式：按需计费

3.4.3 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：购买专享版APIG，绑定APIG触发器，测试文件上传及处理流程。

3.4.3.1 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：输入您自定义的函数名称，此处以“upload-file-1”为例。
   - 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：输入您自定义的依赖包名称，例如“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”，触发器类型选择“API 网关服务(APIG 专享版)”。
   - 实例：选择API实例，若无可用实例，可单击“创建实例”进入创建页面。
   - 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。
   - 请求协议：选择“HTTPS”。
   - 请求方法：选择“ANY”。
   - 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 3-26 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

3.4.3.2 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：输入您自定义的函数名称，此处以“upload-file-1”为例。
   - 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”，触发器类型选择“API 网关服务(APIG 专享版)”。
   - 实例：选择API实例，若无可用实例，可单击“创建实例”进入创建页面。
   - 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。
   - 请求协议：选择“HTTPS”。
   - 请求方法：选择“ANY”。
   - 后端超时（毫秒）：默认5000毫秒。

步骤3 端到端测试

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
```

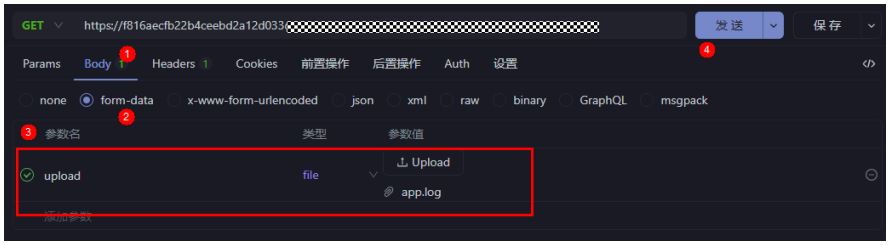
- 以curl工具为例（curl -F的方式主要用的是linux环境），执行如下命令测试：

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

图 3-27 示例

- 以postman工具为例，配置如下参数，配置完成后单击“发送”。

图 3-28 示例



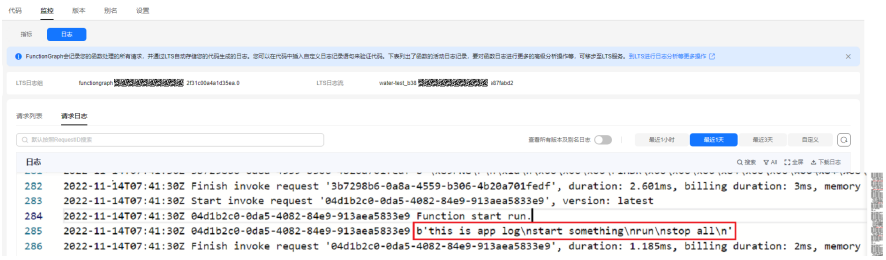
参数名：选择“upload”。

类型：选择“file”。

参数值：单击“Upload”，上传刚才创建好的app.log文件。

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。

图 3-29 查看日志



----结束

3.5 使用 FunctionGraph 函数对 IoTDA 中的设备坐标数据进行转换

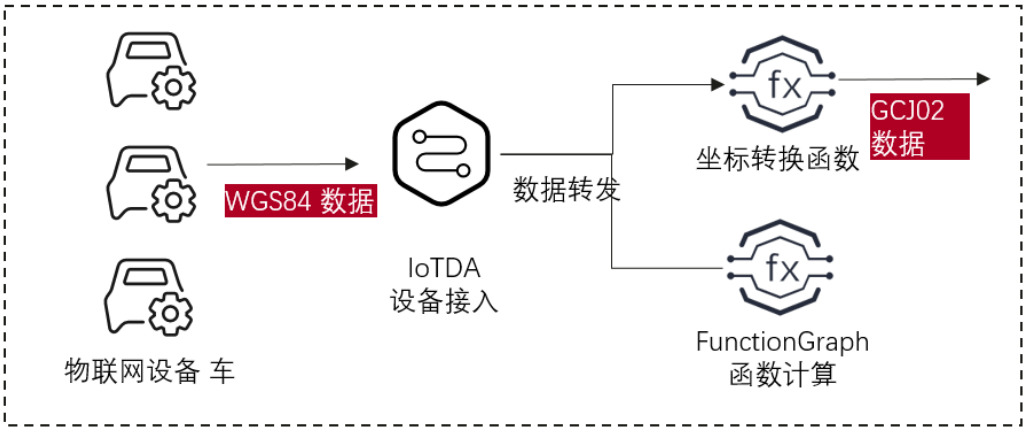
3.5.1 案例概述

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，提供了使用IoTDA + FunctionGraph做坐标转换的示例（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。

3.5.2 准备

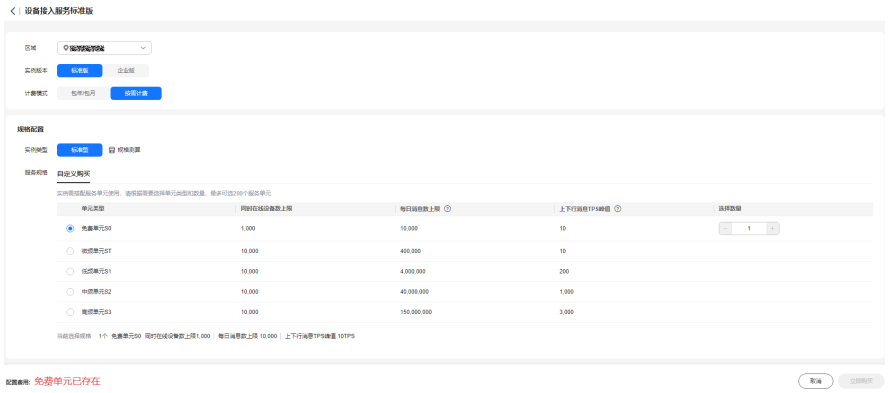
创建IoTDA 转发规则前，需要先创建IoTDA实例，在正常的使用中还需要创建产品，设备。在本案例中仅进行测试，只需要先创建IoTDA实例。

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“购买实例 ”，进入参数配置界面，请您根据实际业务需求进行配置。

图 3-30 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 选择“创建空白函数”，函数类型选择“事件函数”，输入您自定义的函数名称，此处以“iotdemo”为例，选择熟悉的运行时，案例这里使用Python 3.9 ，然后单击创建。

----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IoTDA实例列表中实例名称“总览”页面，然后选择“规则 > 数据转发”，单击“创建规则”。

图 3-31 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：用户自定义。
- 数据来源：选择“设备消息”。
- 触发事件：选择“设备消息上报”。
- 资源空间：保持默认。

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 。

图 3-32 添加转发目标

步骤6 单击启动规则。

----结束

3.5.3 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):
```

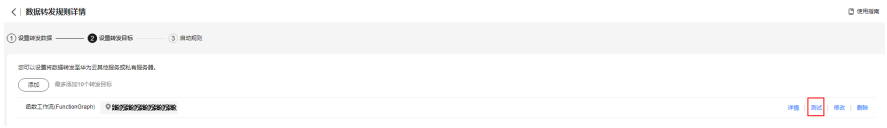
```
    data = event["notify_data"]["body"]
    lat = data["lat"]
    lng = data["lng"]
    print(f" WGS84: ({lng},{lat})")
    gcj_lng, gcj_lat = transform(lng, lat)
    print(f" GCJ02: ({gcj_lng},{gcj_lat})")
    body = {
        "gcj_lng": gcj_lng,
        "gcj_lat": gcj_lat
    }
    return {
        "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，在IoTDA实例列表中单击实例名称进入“总览”页面，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 3-33 转发规则测试



步骤3 输入测试数据单击 “连通性测试” 。

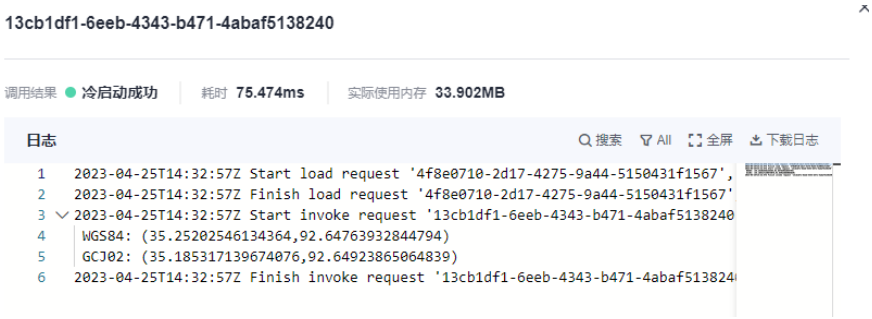
```
{
   "resource": "device.message",
   "event": "report",
   "event_time": "string",
   "notify_data": {
      "header": {
         "app_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
         "device_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
         "node_id": "ABC123456789",
         "product_id":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

图 3-34 连通性测试结果

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

图 3-35 查看日志

图 3-36 查看请求 id 详情



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

3.6 使用 FunctionGraph 函数对 OBS 中的文件进行加解密

3.6.1 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、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应用事件源。（当前“OBS应用事件源”仅支持华北-北京四、华北-乌兰察布一、华东-上海一。）
- 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应用事件源，用来关联函数和OBS桶

3.6.2 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：输入您自定义的桶名称，此处以“dew-bucket-input”为例。
- 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 >密码安全中心”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：输入您自定义的委托名称，此处以“serverless_trust”为例。
- 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“OBS Administrator”和“DEW KeypairFullAccess”。

步骤5 单击“下一步”，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

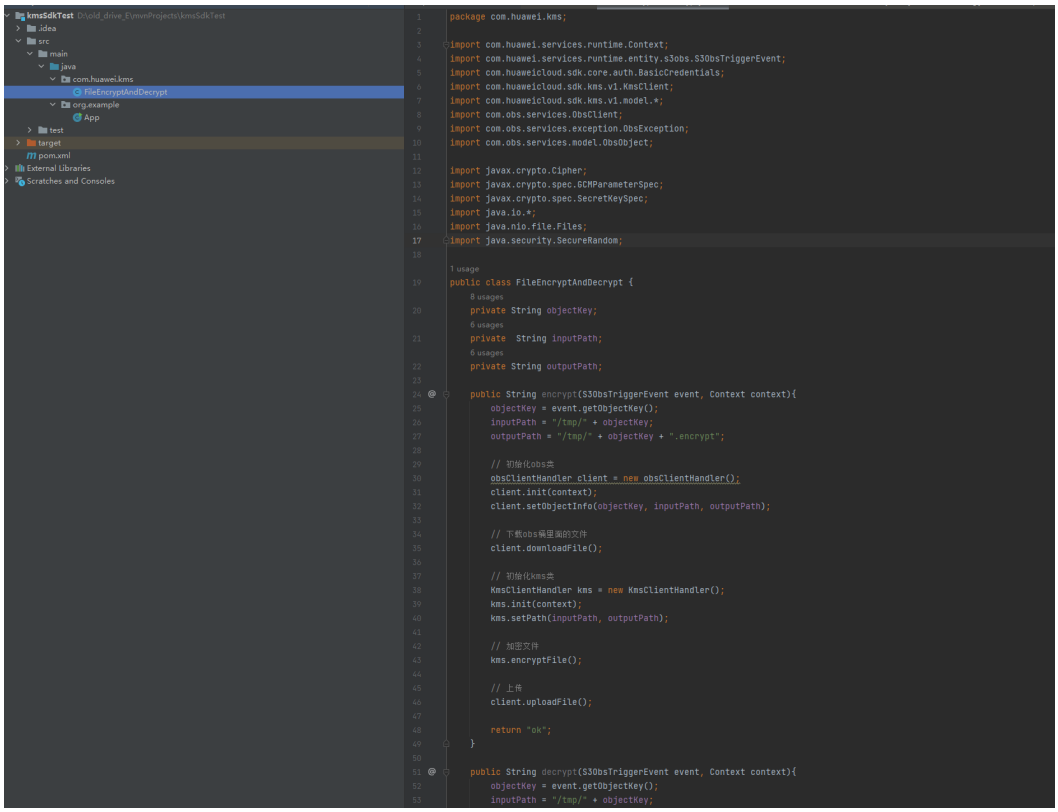
3.6.3 构建程序

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3-37**所示。

图 3-37 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 SECURITY_TOKEN;
        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.getSecurityAccessKey();
                SECRET_ACCESS_KEY = context.SecuritygetSecretKey();
                SECURITY_TOKEN = context.getSecurityToken();
                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).withSecurityToken(SECURITY_TOKEN).
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 SECURITY_ACCESS_KEY = context.getSecurityAccessKey();
                String SECURITY_SECRET_KEY = context.getSecuritySecretKey();
                String SECURITY_TOKEN = context.getSecurityToken();
                String OBS_ENDPOINT = context.getUserData("obs_endpoint");
                obsClient = new ObsClient(SECURITY_ACCESS_KEY, SECURITY_SECRET_KEY, SECURITY_TOKEN,
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 选择“创建空白函数”，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”。

- 函数类型：事件函数。
- 函数名称：输入您自定义的函数名称，此处以“fss_examples_dew”为例。
- 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“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”
3. 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 3-9 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

3.6.4 添加事件源

OBS桶及函数创建以后，可以为函数添加事件源，是通过创建“OBS应用事件源”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“OBS应用事件源”，填写触发器配置信息，如图3-38所示。

桶选择创建OBS桶中创建的“dew-bucket-input”桶。

事件选择“通过页面或Put请求创建或覆盖桶对象”和“使用Post请求创建或覆盖桶对象”。

图 3-38 创建 OBS 应用事件源

步骤3 单击“确定”，完成触发器创建。

说明

OBS应用事件源创建以后，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

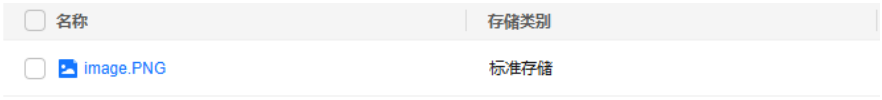
3.6.5 处理文件

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.png文件，如**图3-39**所示。

图 3-39 上传文件

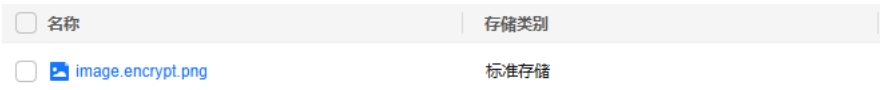


触发函数自动运行

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.encrypt.png，如**图3-40**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 3-40 输出文件



3.7 使用 FunctionGraph 函数识别 LTS 中的异常业务日志并存储到 OBS

3.7.1 案例概述

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。

场景介绍

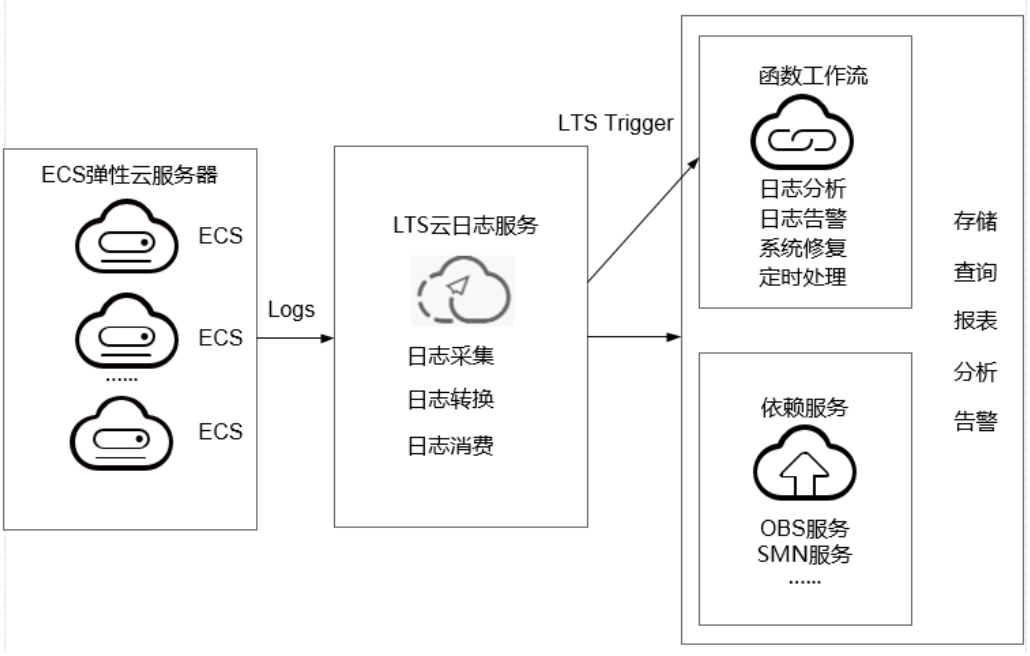
通过LTS云日志服务，快速完成ECS等服务器的任务运行日志采集、加工和转换。

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3-41**。

图 3-41 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。

应用扩展

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。

3.7.2 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3-42**所示，创建过程请参考**创建OBS桶**。

图 3-42 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 3-43 创建委托

3. 开始配置委托。
   - 委托名称：输入您自定义的委托名称，此处以“LtsOperation”为例。
   - 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“SMN Administrator”并勾选。

   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。

3.7.3 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 3-10 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**。

3.7.4 添加事件源

创建 LTS 触发器

选择3.7.2 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3-44所示。

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。

图 3-44 创建 LTS 触发器

3.7.5 处理结果

处理告警信息

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3-45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3-46所示。

图 3-45 告警消息邮件通知

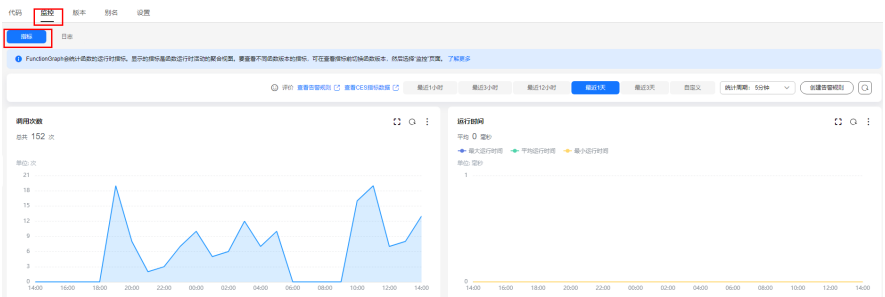
Get warning message.The content of message is:["{\"ip\": \"192.168.1.98\", \"line_no\": 616, \"host_name\": \"ecs-testagent.novalocal\", \"time\": 1530009653059, \"path\": \"/usr/local/telescope/log/common.log\", \"message\": \"2018-06-26/18:40:53 [WRN] [config.go:82] The projectId or instanceId of config.json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

图 3-46 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3-47所示。

图 3-47 函数指标



3.8 使用 FunctionGraph 函数对 LTS 中的日志进行实时过滤

3.8.1 案例概述

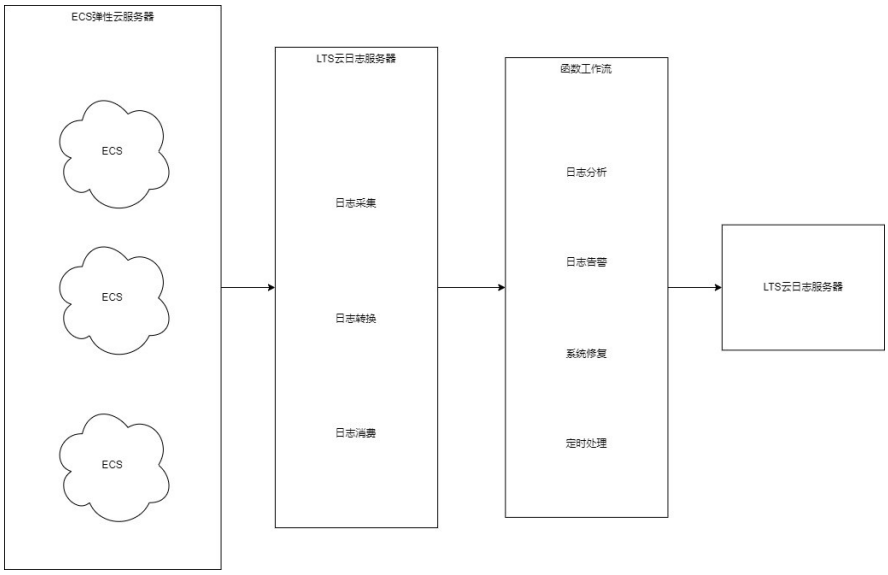
本案例展示了函数工作流服务配合使用云日志服务LTS实现日志云端处理并转储消息到LTS的功能。

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3-48所示。

图 3-48 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。

应用扩展

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。

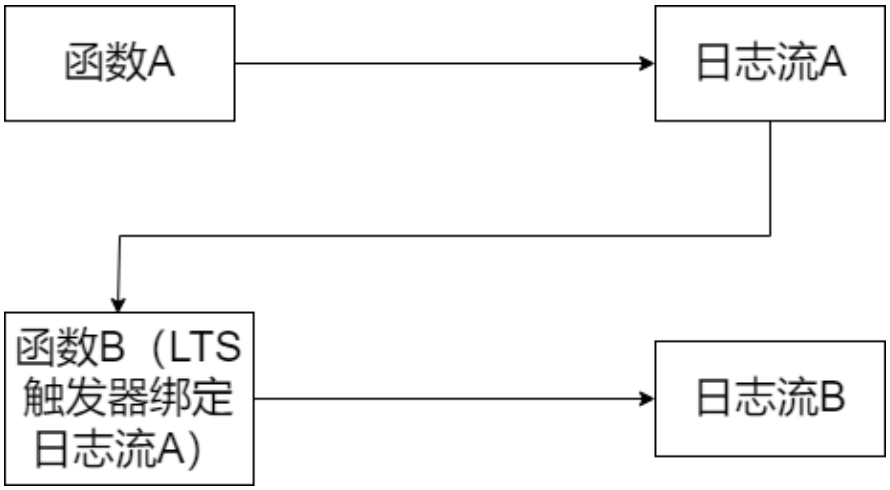
3.8.2 准备

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 3-49 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3-50**所示。

图 3-50 创建委托

步骤3 开始配置委托。

- 委托名称：输入您自定义的委托名称，此处以“LtsOperation”为例。
- 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并勾选“LTS Administrator”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”，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

3.8.3 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 > 主机”；
2. 在页面右上方，单击“安装ICAgent”；
3. 在弹出的“安装ICAgent”窗口中获取接入点IP。

图 3-51 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**。

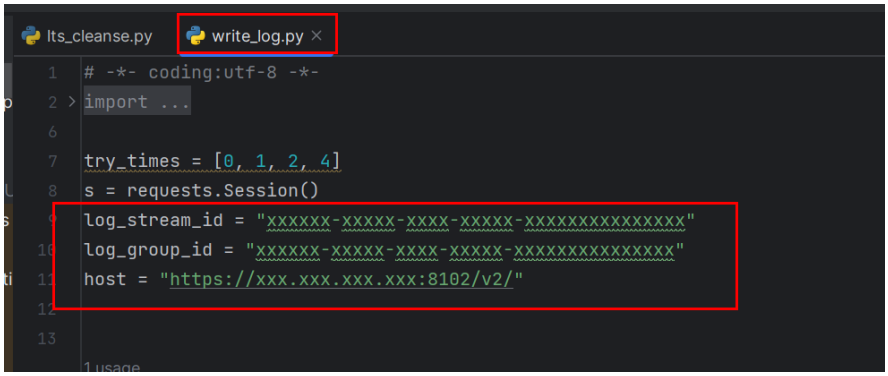
（3）制作函数B需要的 lts 依赖包，具体添加依赖方法请参考**如何在函数平台创建依赖包**和**如何为函数添加依赖包**。制作依赖包时可以参考命令“pip install huaweicloudsdklts”。同时，**示例代码**中包含了已适用于python3.9的huaweicloudsdklts依赖。

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3-52**所示。

图 3-52 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3-53**和**图3-54**所示。

图 3-53 lts_cleanse.py

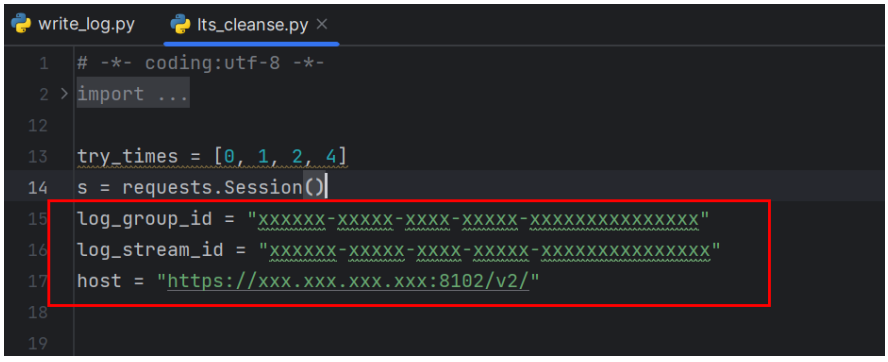


图 3-54 为函数 B 添加依赖包

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。

3.8.4 添加事件源

创建 LTS 触发器

选择3.8.2 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3-55所示。

图 3-55 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。

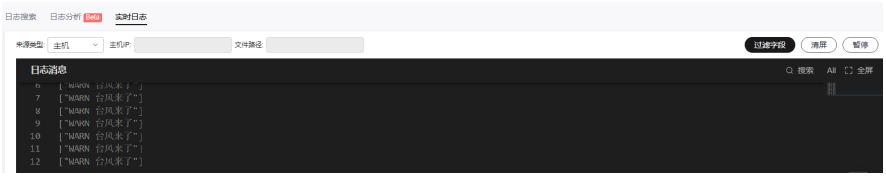
3.8.5 处理结果

处理告警信息

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3-56和图3-57是过滤前和过滤后的实时日志对比。

图 3-56 过滤前日志

图 3-57 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 3-58 函数指标 1

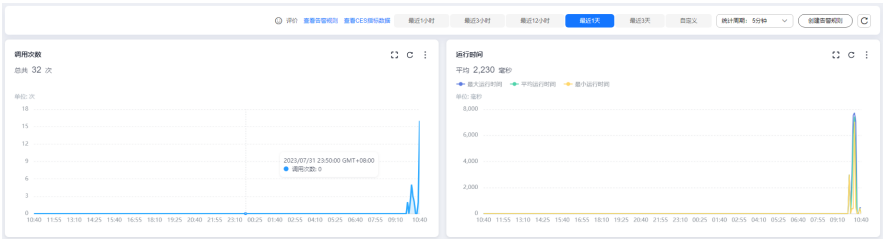
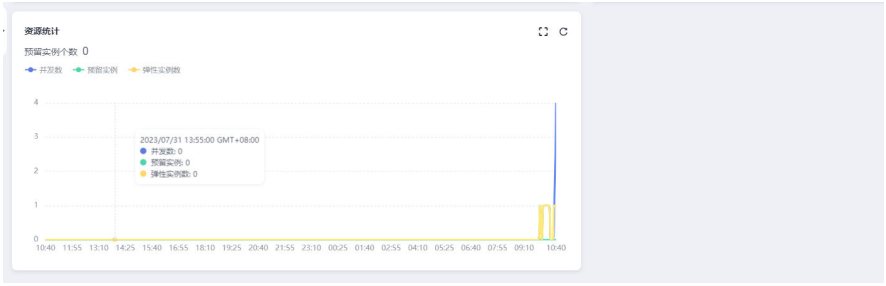


图 3-59 函数指标 2

图 3-60 函数指标 3



3.9 使用 FunctionGraph 函数流对 OBS 中的图片进行旋转

3.9.1 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）

3.9.2 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：输入您自定义的桶名称，此处以“your-bucket-input”为例。
- 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：输入您自定义的委托名称，此处以“serverless_trust”为例。
- 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“OBS Administrator”。

步骤2 单击“下一步”，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

3.9.3 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3-61**所示。

图 3-61 样例代码目录

```
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”。

- 函数类型：事件函数。
- 函数名称：输入您自定义的函数名称，此处以“test-rotate”为例。
- 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。

  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 3-11 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。

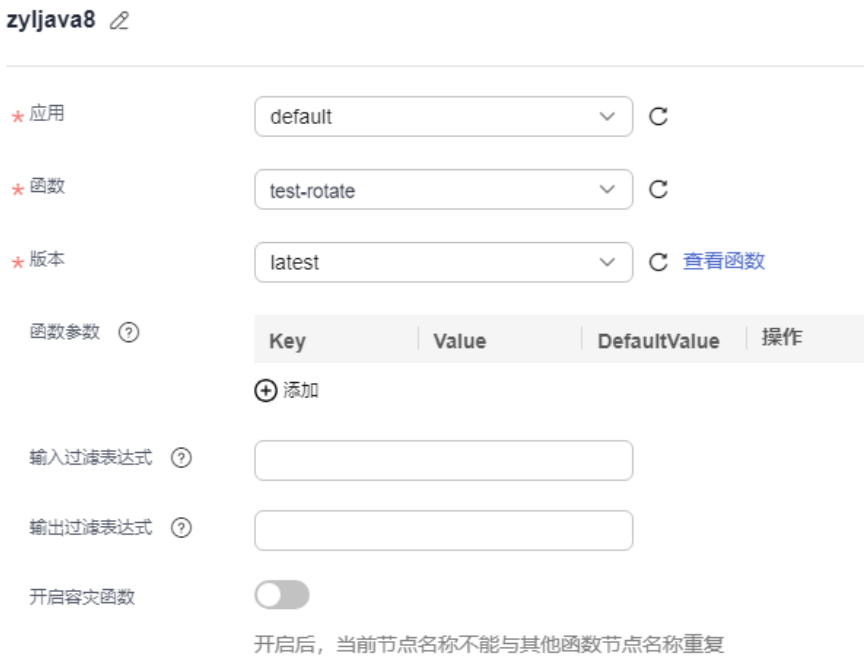
单击“创建快速函数流”，进入创建快速函数流流程。

图 3-62 创建快速函数流

步骤2 拖拽一个函数节点，单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 3-63 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 3-64 保存函数流

----结束

3.9.4 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3-65**，上传完成后如**图3-66**所示。

图 3-65 示例

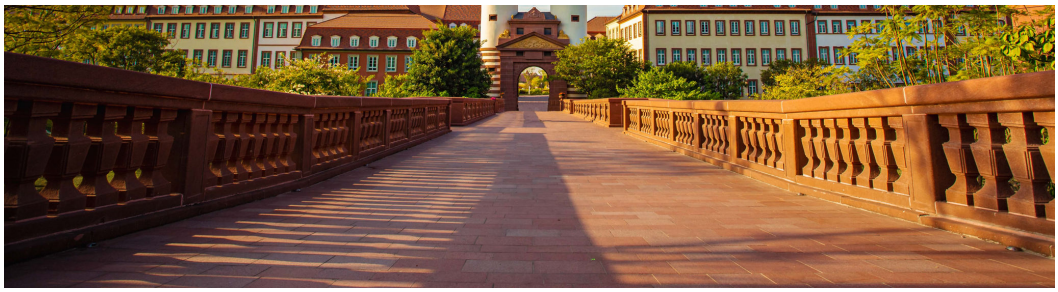
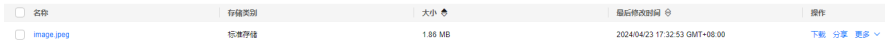
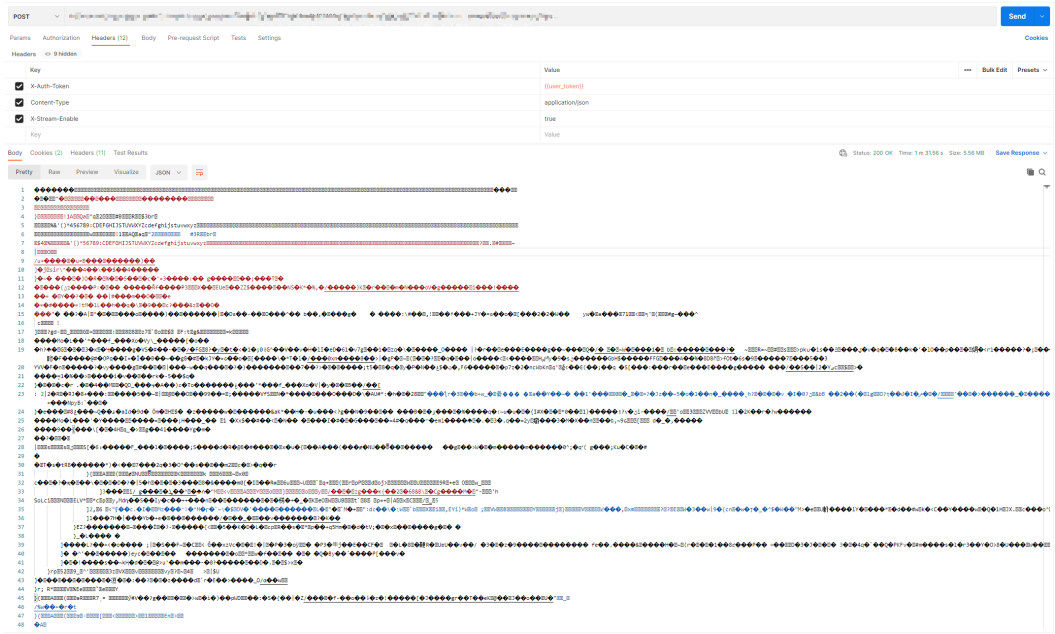


图 3-66 上传图片



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：

3.10 使用 FunctionGraph 函数流对图片进行压缩和打水印

本章节主要介绍如何使用函数流实现流式大文件处理。您可以根据实际业务场景来创建快速函数流实现。

背景与价值

Serverless Workflow由于自身可编排、有状态、持久化、可视化监控、异常处理、云服务集成等特性，适用于很多应用场景，比如：

- 复杂度高需要抽象的业务（订单管理，CRM 等）
- 业务需要自动中断 / 恢复能力，如多个任务之间需要人工干预的场景（人工审批，部署流水线等）

- 业务需要手动中断 / 恢复（数据备份 / 恢复等）
- 需要详细监控任务执行状态的场景
- 流式处理（日志分析，图片 / 视频处理等）

  当前大部分 Serverless Workflow 平台更多关注控制流程的编排，忽视了工作流中数据流的编排和高效传输，上述场景中，由于数据流相对简单，所以各大平台支持都比较好，但是对于文件转码等存在超大数据流的场景，当前各大平台没有给出很好的解决方案。华为云FunctionGraph函数工作流针对该场景，提出了Serverless Streaming 的流式处理方案，支持毫秒级响应文件处理。

技术原理

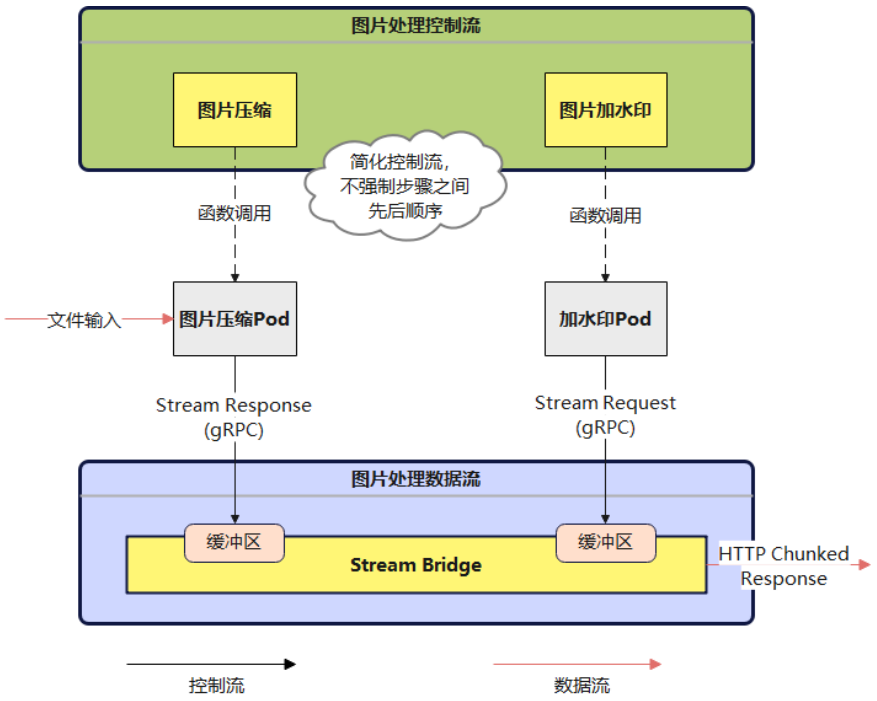
华为云FunctionGraph函数工作流提出 Serverless Streaming 的流式可编排的文件处理解决方案，步骤与步骤之间通过数据流驱动，更易于用户理解。本章通过图片处理的例子解释该方案的实现机制。

如果需要驱动一个工作流执行，工作流系统需要处理两个部分：

- 控制流：控制工作流的步骤间流转，以及步骤对应的 Serverless 函数的执行。确保步骤与步骤之间有序执行。
- 数据流：控制整个工作流的数据流转，通常来说上一个步骤的输出是下一个步骤的输入，比如上述图片处理工作流中，图片压缩的结果是打水印步骤的输入数据。

在普通的服务编排中，由于需要精准控制各个服务的执行顺序，所以控制流是工作流的核心部分。然而在文件处理等流式处理场景中，对控制流的要求并不高，以上述图片处理场景举例，可以对大图片进行分块处理，图片压缩和加水印的任务不需要严格的先后顺序，图片压缩处理完一个分块可以直接流转到下一个步骤，而不需要等待图片压缩把所有分块处理完再开始加水印的任务。

基于上述理解，华为云FunctionGraph工作流的 Serverless Streaming 方案架构设计如下图所示：



在 Serverless Streaming 的流程中，弱化控制流中步骤之间的先后执行顺序，允许异步同时执行，步骤与步骤之间的交互通过数据流驱动。其中数据流的控制通过 Stream Bridge 组件来实现。同时函数 SDK 增加流式数据返回接口，用户不需要将整个文件内容返回，而是通过 gRPC Stream 的方式将数据写入到 Stream Bridge，Stream Bridge 用来分发数据流到下一个步骤的函数 Pod 中。

操作步骤

步骤1 创建一个图片压缩的函数，其中代码在处理返回数据通过 ctx.Write() 函数将结果以流式数据的形式返回：

说明

目前只支持go函数！

```
func ImageTransform(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    reader, writer := io.Pipe()
    file, err := downloadOriginFile(ctx)
    if err != nil {
        return "nok", err
    }

    go func() {
        defer writer.Close()
        encodeImageFile(writer, file, ctx)
    }()

    defer reader.Close()
    // Write result stream back to client
    err = ctx.Write(reader)
    return "ok", err
}

// downloadOriginFile used to download the origin video file
func downloadOriginFile(ctx context.RuntimeContext) (*os.File, error) {
}

// encodeImageFile used to encode the origin image file to target resolution, result output will writes to writer stream
func encodeImageFile(writer io.Writer, originFile *os.File, ctx context.RuntimeContext)  {
}
```

FunctionGraph 通过 ctx.Write() 函数提供了流式返回的能力，对开发者来说，只需要将最终结果通过流的方式返回，而不需要关注网络传输的细节。

步骤2 在 FunctionGraph 的函数流控制台完成工作流编排，举例如下。

步骤3 调用工作流的同步执行接口，获取最终结果的文件流，数据将以 chunked 流式返回的方式返回到客户端。

----结束

4 功能应用类实践

4.1 使用 FunctionGraph 函数和 CTS 识别非法 IP 的登录登出操作

4.1.1 案例概述

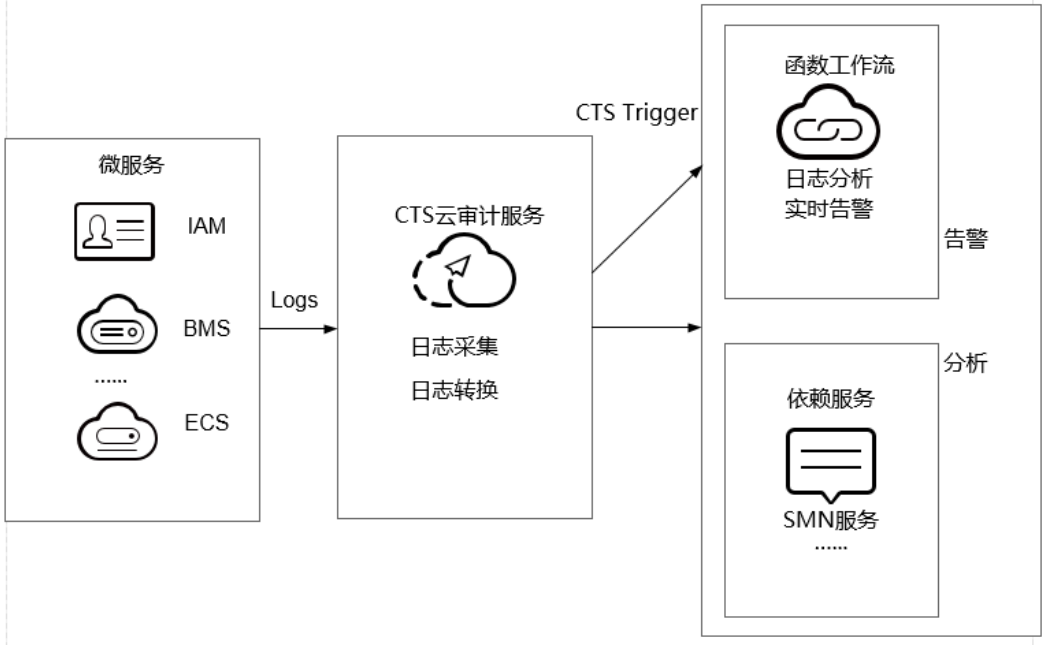
场景介绍

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4-1**所示。

图 4-1 处理流程



案例价值点

- 通过云审计服务CTS，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。
- 结合消息通知服务SMN提供日志、告警功能。

约束与限制

本实践涉及云审计服务CTS和消息通知服务SMN的操作，相关约束限制请参考以下章节内容：

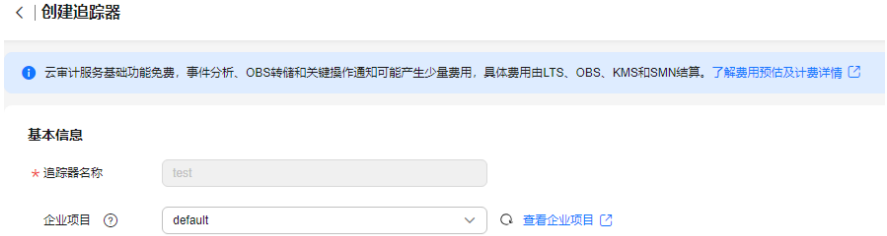
- **云审计服务事件追踪器**
- **创建关键操作通知**

4.1.2 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4-2**所示。开通方法与配置请参考**追踪器配置**。

图 4-2 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：输入您自定义的委托名称，此处以“serverless_trust”为例。
- 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“CTS Administrator”和“SMN Administrator”。

说明

- SMN Administrator：拥有该权限的用户可以对SMN服务下的资源执行任意操作。
- 选择“CTS Administrator”，由于该策略有依赖，在勾选时，还会自动勾选依赖的策略：Tenant Guest。

步骤5 单击“下一步”，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

告警消息推送

具体配置流程及其约束限制请参考**创建关键操作通知**。

- 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。

4.1.3 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4-1所示。

表 4-1 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。

4.1.4 添加事件源

选择4.1.2 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4-3所示。

图 4-3 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。

4.1.5 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4-4所示。

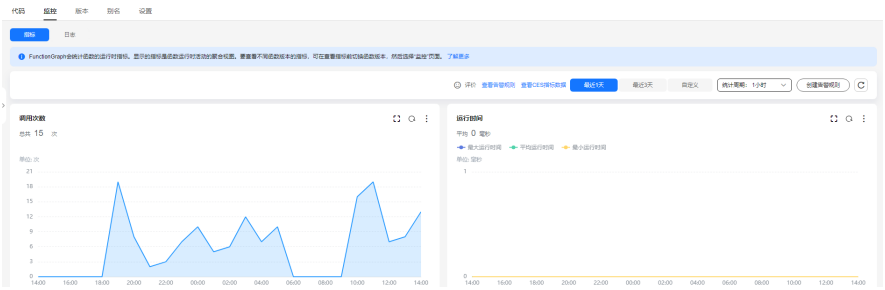
图 4-4 告警消息邮件通知

```
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4-5所示。

图 4-5 函数指标



4.2 使用 FunctionGraph 函数作为后端实现 APIG 的自定义认证能力

4.2.1 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。

4.2.2 资源规划

请保证以下资源在同一区域。

表 4-2 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

4.2.3 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”。

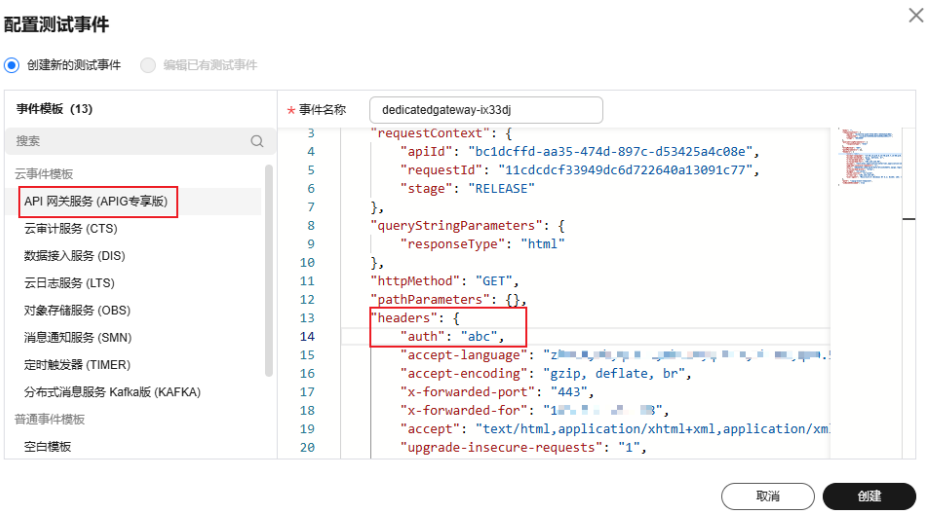
- 模板：选择“使用空白模板”。

- 函数类型：事件函数。
- 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。

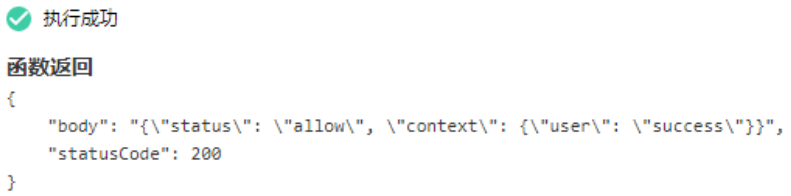
步骤5 配置测试事件，测试用于前端自定义认证的函数。单击“配置测试事件”，选择事件模板。根据实际情况修改后保存测试模板(本示例在"headers"中添加"auth":"abc")，完成后单击“创建”。

图 4-6 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 4-7 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 4-8 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)
```

4.2.4 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4-9、图4-10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 4-9 前端定义配置

图 4-10 安全配置

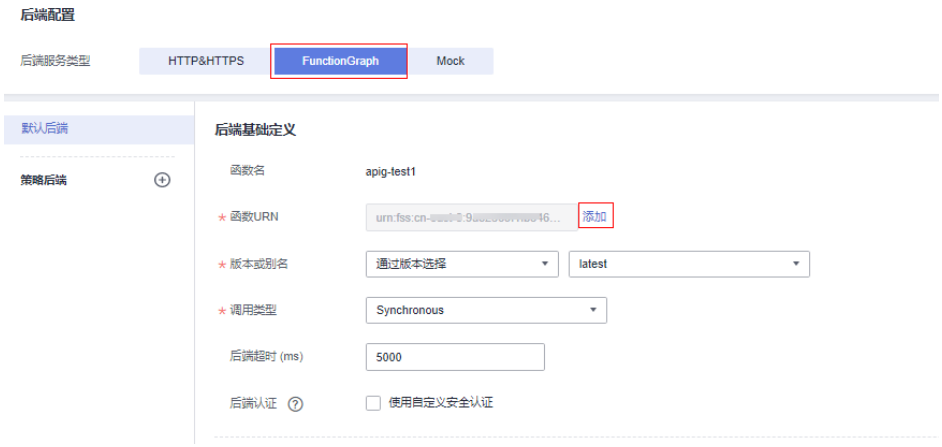
说明

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4-11**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 4-11 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 4-12 发布 API

----结束

4.2.5 调试并调用 API

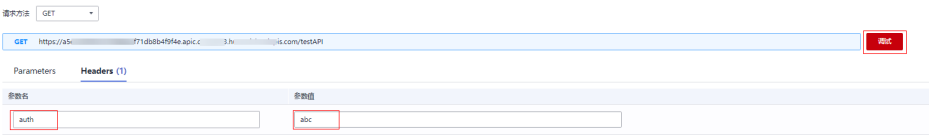
API网关提供了在线调试的功能，因此一般建议在API网关上完成API配置之后，可以先通过此功能确认API是否配置成功。

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 4-13 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4-14。

图 4-14 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

4.3 使用 FunctionGraph HTTP 函数处理 gRPC 请求

方案概述

本章节主要指导用户使用gRPC，在FunctionGraph中处理gRPC请求。

本章节以**gRPC example code**项目中“example/helloworld”为例，使用HTTP函数的方式在FunctionGraph中处理gRPC请求。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FunctionGraph上。

说明

- 当前仅拉美-圣地亚哥支持。
- 用户默认没有gRPC权限，如果需要使用，请在**工单系统**提交工单添加白名单。

操作流程

1. **构建代码包**

   创建源文件“main.go”，代码如下：

```
// Package main implements a grpc_server for Greeter service.
package main

import (
    "context"
    "flag"
    "fmt"
    "log"
    "net"

    pb "helloworld/helloworld"

    "google.golang.org/grpc"
```

```
)

var (
    port = flag.Int("port", 8000, "The grpc_server port")
)

// server is used to implement helloworld.GreeterServer.
type server struct {
    pb.UnimplementedGreeterServer
}

// SayHello implements helloworld.GreeterServer
func (s *server) SayHello(ctx context.Context, in *pb.HelloRequest) (*pb.HelloReply, error) {
    log.Printf("Received: %v", in.GetName())
    return &pb.HelloReply{Message: "Hello " + in.GetName()}, nil
}

func main() {
    flag.Parse()
    lis, err := net.Listen("tcp", fmt.Sprintf("127.0.0.1:%d", *port))
    if err != nil {
       log.Fatalf("failed to listen: %v", err)
    }
    s := grpc.NewServer()
    pb.RegisterGreeterServer(s, &server{})
    log.Printf("grpc_server listening at %v", lis.Addr())
    if err := s.Serve(lis); err != nil {
       log.Fatalf("failed to serve: %v", err)
    }
}
# bootstrap
$RUNTIME_CODE_ROOT/grpc-server
```

在“main.go”中，使用**8000**端口启动了一个gRPC服务器，并注册了“helloworld.GreeterServer”，调用该服务将返回“Hello XXX”。

2. **编译打包**

   a. 在**linux**机器下，将上述代码编译 **go build -o grpc-server main.go**。然后，将grpc-server和bootstrap打包为xxx.zip。

   b. 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 grpc-server 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
```

3. **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**。

4. **创建APIG触发器**

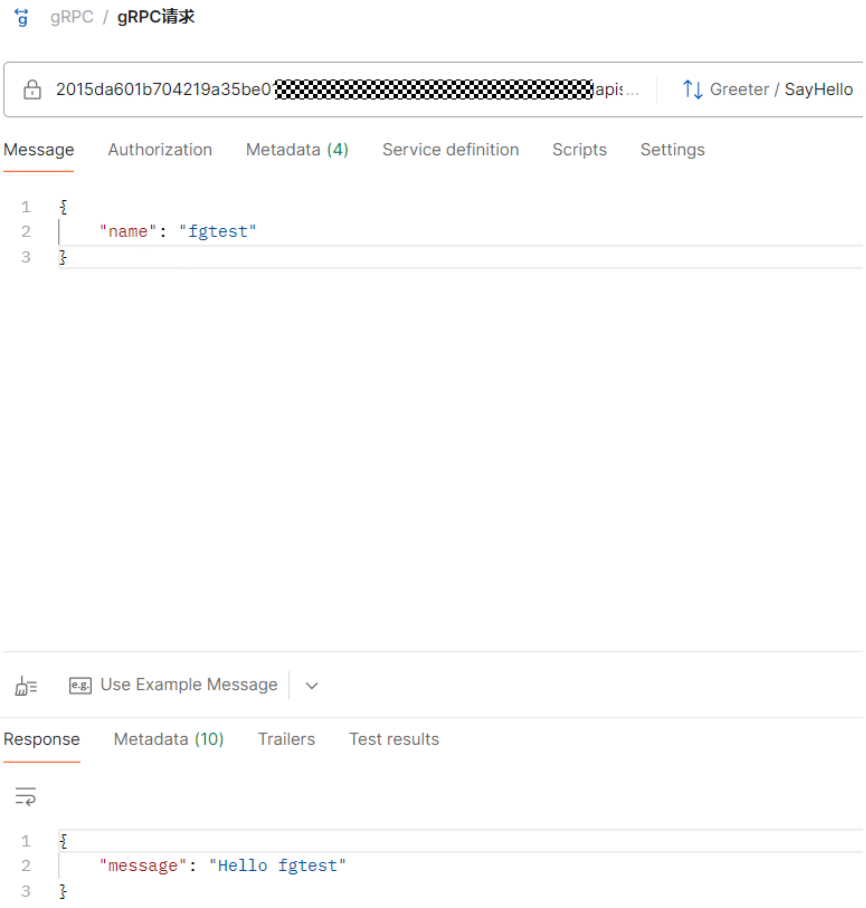
   请参见**使用APIG触发器**，创建APIG触发器，“请求协议”建议选择“gRPC”，“安全认证”建议选择“None”,方便调试

图 4-15 APIG 触发器

5. **调用测试**

   用postman去调试gRPC。

图 4-16 gRPC 请求结果



4.4 使用 FunctionGraph 的 Java 函数配置 Log4j2 实现日志打印

案例概述

函数工作流服务支持Java函数配置log4j2，本章节将介绍如何使用函数和log4j2的能力实现日志打印。

步骤一：下载程序包

本例使用Java语言实现日志打印，请下载**Log_demo.jar**样例代码直接使用，无需其他修改。

关键示例代码如下所示，可参考了解。

```
package org.example;

import com.huawei.services.runtime.Context;
import lombok.extern.slf4j.Slf4j;
import org.apache.logging.log4j.core.config.Configurator;
import org.apache.logging.log4j.util.LoaderUtil;

import java.net.URISyntaxException;
import java.util.Objects;

@Slf4j
public class LogTest {

    public void init(Context context) {
        try {

Configurator.reconfigure(Objects.requireNonNull(LoaderUtil.getThreadContextClassLoader().getResource("log4j2-custom.xml")).toURI());
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
    }

    public void handler(String event, Context context) {
        log.debug("debug log");
        log.info("info log");
        log.warn("warn log");
        log.error("info log");
    }
}
```

其中初始化入口添加了如下代码：

```
Configurator.reconfigure(Objects.requireNonNull(LoaderUtil.getThreadContextClassLoader().getResource("log4j2-custom.xml")).toURI());
```

步骤二：创建函数

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”，右上角单击“创建函数”进入创建函数界面。

步骤2 选择“创建空白函数”，基本信息配置如下：

- 函数类型：选择“事件函数”。
- 区域：根据实际情况选择。
- 函数名称：自定义。
- 运行时：选择“Java 8”。

其他参数保持默认，单击“创建函数”。

步骤3 上传函数代码。

函数创建完成后，进入详情页面，在“代码”页签下选择“上传代码 > Zip > 文件”，添加**步骤一：下载程序包**中下载的Zip代码包进行部署。

步骤4 开启类隔离。

成功部署代码包后，如图4-17所示，选择“设置 > 高级设置”，开启“类隔离”，单击“保存”。

图 4-17 开启类隔离

步骤5 设置函数执行入口。

如图4-18所示，选择“设置 > 常规设置”，将“函数执行入口”参数设置为：org.example.LogTest.handler，单击“保存”。

图 4-18 设置函数执行入口

步骤6 设置函数初始化入口。

如图4-19所示，选择“设置 > 生命周期”，开启“初始化配置”，将“函数初始化入口”参数设置为：org.example.LogTest.init，单击“保存”。

图 4-19 设置函数初始化入口

----结束

步骤三：调试函数

步骤1 配置完所有参数后，如图4-20所示，选择“代码”页签，单击“配置测试事件”，默认使用“空白模板”，单击“创建”。

图 4-20 配置测试事件

步骤2 选择创建好的测试事件，单击“测试”，在右侧“执行结果”可查看函数成功打印日志结果。

----结束

4.5 使用 FunctionGraph 部署 AI 绘画 Stable Diffusion 应用

4.5.1 使用 FunctionGraph 部署 AI 绘画 Stable Diffusion 方案概述

AI 绘画 Stable Diffusion 及其应用场景

Stable Diffusion是一种开源的文本到图像的生成模型，能够根据用户提示生成高质量、独特的图像，提供了广泛的创意可能性。通过Stable Diffusion WebUI，可视化图像生成过程，使AI绘画更加易用和可控。

使用FunctionGraph应用中心部署AI绘画Stable Diffusion应用，可根据具体使用需求进行相应的部署操作。

- **快速部署**：无需拥有深厚的技术背景，使用默认模型和临时域名快速部署Stable Diffusion模型，立即享受文生图的乐趣。
- **自定义域名**：通过绑定自定义域名，将Stable Diffusion应用部署到自定义域名下开放内网或外网访问，实现更多的自定义功能。
- **自定义模型**：应用支持挂载文件系统上传自定义模型使用，搭配不同模型能力从而获得更个性化的图像生成效果。
- **进阶使用**：如需在具体业务中使用AI绘画应用，本文还提供**多用户资源共享**和**多用户资源隔离**、**API模式访问应用**以及**启用WebUI认证**提升业务性能的相关使用方法。

方案优势

- 轻松部署

  部署流程简便，结合弹性的Serverless解决方案，无需服务器管理和运维，即可体验AI绘画功能。
- 开源和定制化

  提供多种自定义和进阶使用场景，轻松实现高阶个性化的AI绘画。

约束与限制

当前使用FunctionGraph应用中心部署AI绘画Stable Diffusion应用仅限于“华东-上海一”区域，请确保所有相关资源均部署于该区域。

使用 Moderation 审核生成结果

Stable Diffusion是一种AIGC推理模型，使用其生成图片的最终结果会因提示词、模型选择的不同存在较大的不确定性，容易存在违规风险，建议在使用过程中配合华为云Moderation对生成结果进行审核，以降低风险，详细使用指南请参考**图像内容审核（V3）**。

4.5.2 使用 FunctionGraph 部署 AI 绘画 Stable Diffusion 资源和成本规划

本实践根据使用需求的不同，涉及的计费服务有所不同，请参考**表4-3**根据具体需求规划资源与成本。

表 4-3 资源和成本规划

资源	资源说明	计费说明	是否必须
函数工作流 FunctionGraph	• 函数类型：容器镜像HTTP函数 • 区域：华东-上海一 • 购买量：2（应用创建成功后自动生成函数）	• 计费模式：按需计费。 • 函数工作流提供免费试用，每月前100万次调用免费。具体计费项及说明请参考**函数工作流按需计费说明**。	必须。

资源	资源说明	计费说明	是否必须
API网关 APIG	• 版本：专享版API网关 • 区域：华东-上海一 • 购买量：1	• 计费模式：根据业务需求选择包年/包月或按需计费。 • 具体计费方式及标准请参考**专享版API网关计费模式概述**。	必须。
云解析服务 DNS	解析公网域名	免费。	使用华为云云解析服务进行公网域名解析时必须。
虚拟私有云 VPC	• 区域：华东-上海一 • 子网数量：1 • 购买量：1	• 虚拟私有云：免费。 • 子网：免费。	使用自定义模型和多用户使用时必须。
弹性文件服务 SFS	• 区域：华东-上海一 • 文件系统类型：SFS Turbo文件系统 在售 • 类型：本例使用 250MB/S/TiB • 容量：1.2 TB • 购买量：1	• 计费模式：本例使用按需计费。 • 具体计费项请参考**弹性文件服务按需计费说明**。	上传和使用自定义模型时必须。
弹性云服务器 ECS	• 区域：华东-上海一 • 操作系统：公共镜像 EulerOS 2.5 64bit(40GiB) • 安全组数量：1 • 购买量：1	• 计费模式：本例使用按需计费。 • 创建安全组：免费。 • 实例类型、存储规格和是否开启公网访问请根据业务需求选择，具体计费项及标准请参考**弹性云服务器按需计费说明**。	使用ECS作为NFS服务器时必须。

4.5.3 使用 FunctionGraph 部署 AI 绘画 Stable Diffusion 操作流程

表4-4介绍本实践中使用FunctionGraph部署AI绘画Stable Diffusion应用的操作流程与说明，其中进阶使用流程主要面向具体的业务使用需求场景，请参考相关说明选择使用。

表 4-4 部署 AI 绘画 Stable Diffusion 应用操作流程说明

操作流程	说明
部署和使用AI绘画Stable Diffusion应用	使用FunctionGraph应用中心的AI绘画Stable Diffusion模板创建应用，即可使用默认模型和临时域名进行AI绘画。
绑定自定义域名（可选）	如需绑定自定义域名访问AI绘画应用，还需进行以下操作： 1. 准备自定义域名 2. 配置域名解析 3. 绑定自定义域名
上传自定义模型（可选）	如需使用自定义模型进行AI绘画，还需进行以下操作： 1. 创建虚拟私有云VPC和子网 2. 创建SFS Turbo文件系统 3. 初始化自定义模型挂载文件系统 4. 上传与加载自定义模型
进阶使用：使用ECS作为NFS服务器实现多用户资源隔离	如需实现多用户使用场景下的资源隔离，FunctionGraph函数支持挂载ECS服务器作为文件系统来源，通过设置共享NFS路径可有效管理多用户模型资源，可通过以下操作实现： 1. 购买ECS服务器 2. 设置ECS下的NFS共享 3. 在Stable Diffusion应用的函数中挂载ECS 4. 上传与加载模型
进阶使用：通过挂载同一SFS文件系统实现多用户资源共享	如需实现多用户使用场景下的资源共享，各用户可以挂载同一个SFS Turbo文件系统，以实现文件系统内的模型资源共享，同时通过设置个人应用的结果保存路径实现推理结果隔离，可通过以下操作实现： 1. 创建多用户配置文件 2. 修改环境变量使用新配置文件 3. 修改结果保存路径
进阶使用：启用WebUI认证	如需加强应用使用的安全性，可以通过配置函数环境变量的方式启用WebUI认证，访问WebUI时将需要输入用户名和密码才可以进行绘图操作。
进阶使用：使用API模式访问应用	如需使用API模式访问应用，可以通过配置函数环境变量的方式启用并配置并发参数。

4.5.4 部署和使用 AI 绘画 Stable Diffusion 应用

在FunctionGraph应用中心，使用AI绘画Stable-Diffusion模板创建应用并配置相关委托，创建成功后即可使用内置默认模型和临时域名进行AI绘画。

步骤一：为 FunctionGraph 创建云服务委托

使用FunctionGraph应用中心部署Stable Diffusion应用，需FunctionGraph服务与其他云服务协同。因此，部署前须配置委托，允许FunctionGraph使用必要的其他云服务资源。

步骤1 登录**统一身份认证服务控制台**，左侧导航栏选择“委托”，进入“委托”页面后，右上角单击“创建委托”。

步骤2 在“创建委托”页面，配置如下参数：

- 委托名称：填写“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（可选）：填写“AI绘画应用”。

步骤3 单击“完成”，系统提示创建成功，单击“立即授权”，进入“授权”界面。

步骤4 在“选择策略”界面根据具体需求搜索**表4-5**中的策略并勾选，勾选完成后单击“下一步”。

表 4-5 策略及相关说明

策略	策略权限说明	是否必选
SWR Admin	容器镜像服务(SWR)管理员，拥有该服务下的所有权限。	必选。
VPC Administrator （系统将同时勾选该系统角色依赖的Server Administrator系统角色，无需手动取消）	VPC Administrator：虚拟私有云服务管理员。 Server Administrator：服务器管理员。	上传和使用自定义模型时必选。
SFS FullAccess	弹性文件服务所有权限。	上传和使用自定义模型挂载SFS文件系统时必选。
SFS Turbo FullAccess	弹性文件服务SFS Turbo的所有权限。	上传和使用自定义模型挂载SFS文件系统时必选。

步骤5 在“设置最小授权范围”界面如**图4-21**所示，选择“指定区域项目资源”，勾选“cn-east-3 [华东-上海一]”，单击“确定”。

图 4-21 指定区域项目资源

步骤6 系统提示授权成功，单击“完成”可查看授权记录。

----结束

步骤二：购买专享版 APIG 实例

请根据业务需要，参考**创建APIG实例**购买相应规格的专享版APIG实例。购买过程中请参照以下注意事项：

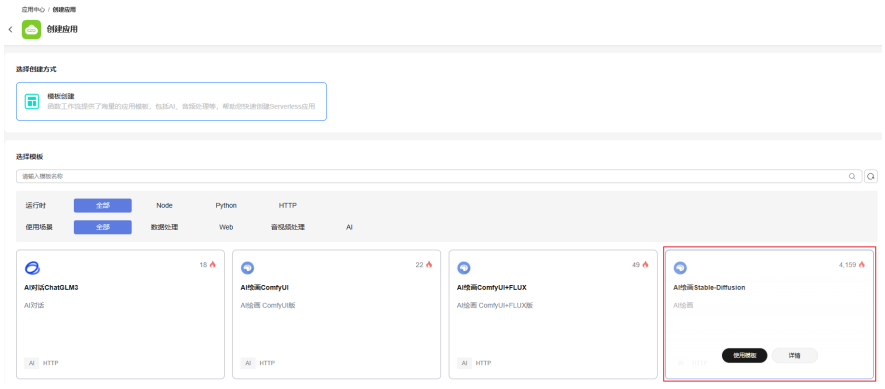
- 区域：选择“华东-上海一”。
- 可用区：与所创建的子网可用区保持一致，本例使用“可用区1”。
- 公网入口：本例需“开启公网入口”，请根据实际需求选择入公网带宽。

步骤三：使用 AI 绘画 Stable-Diffusion 模板创建应用

步骤1 登录**函数工作流控制台**，区域选择“华东-上海一”。在左侧导航栏选择“应用中心”，单击“创建应用”，进入模板选择页面。

步骤2 如**图4-22**所示，找到“AI绘画Stable-Diffusion”模板，单击“使用模板”，请仔细阅读弹出的说明后进行勾选操作，单击“同意并继续创建”。

图 4-22 选择使用 AI 绘画 Stable-Diffusion 模板



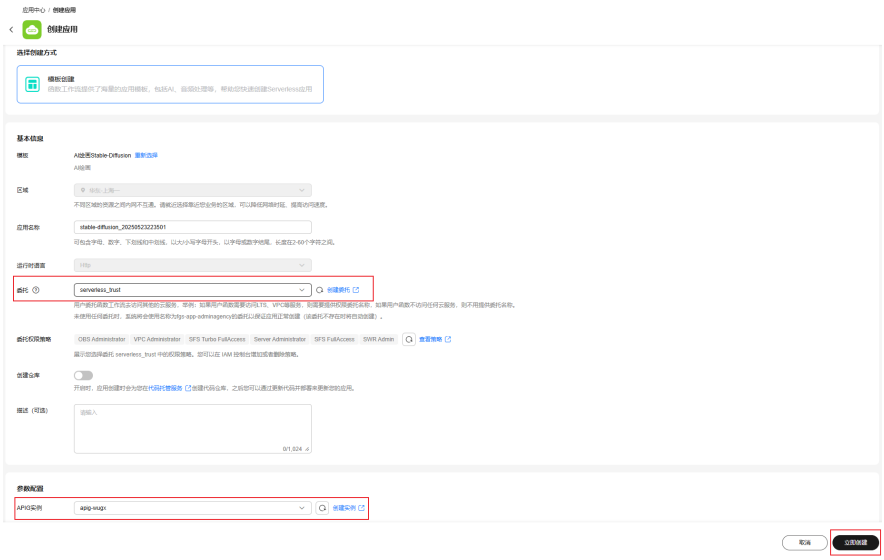
说明

如果系统弹出“服务开通”提示弹窗，请阅读说明后单击“立即开通”。

步骤3 进入“应用配置”页面，填写应用参数。完成填写后如**图4-23**所示，单击页面右下角的“立即创建”按钮。

- 应用名称：自定义填写或采用默认名称。
- 委托名称：选择**步骤一：为FunctionGraph创建云服务委托**创建的“serverless_trust”委托。
- APIG实例：选择**步骤二：购买专享版APIG实例**创建的APIG实例。

图 4-23 应用配置



步骤4 等待应用创建完成，创建成功的应用包含函数、API网关、触发器等资源，如**图4-24**所示，其中函数服务的关键资源功能说明请参考**表4-6**。

为方便快速体验，应用中心会为您分配一个临时域名，此临时域名仅可用于测试使用，有效期30天。若想开放应用长期访问，需绑定自定义域名，具体部署步骤请参考**绑定自定义域名（可选）**。

图 4-24 应用创建完成

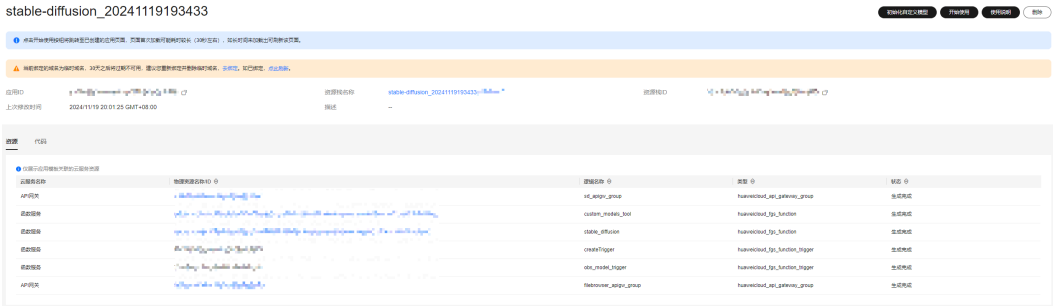


表 4-6 关键函数服务功能

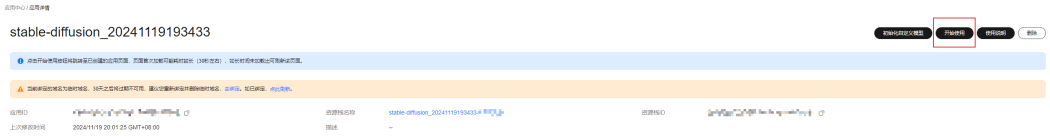
函数资源逻辑名称	功能说明
stable_diffusion	AI绘图功能主体，可通过其APIG触发器访问Stable Diffusion WebUI界面。
custom_models_tool	可通过其APIG触发器管理Stable Diffusion应用资源，如模型、插件的上传和图片下载等。

----结束

步骤三：使用默认模型和临时域名进行 AI 绘画

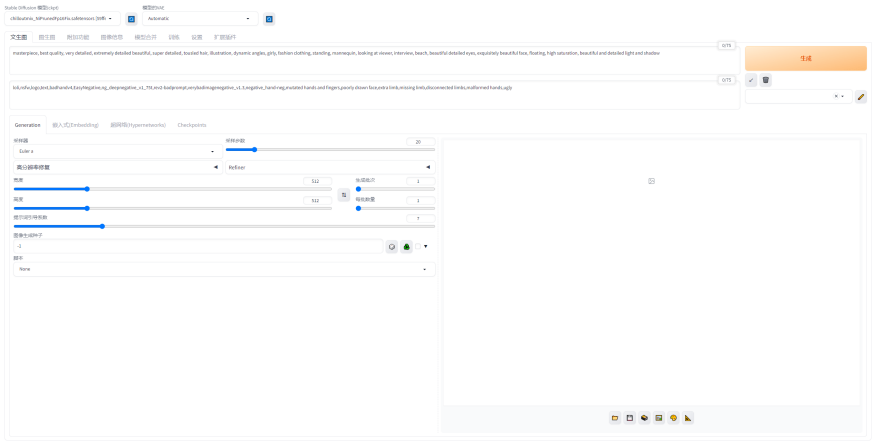
步骤1 如图4-25所示，单击应用详情界面的“开始使用”，进入Stable Diffusion WebUI界面。首次加载冷启动时间会持续30s左右，如遇到加载超时问题，可以通过刷新页面解决。

图 4-25 开始使用 Stable Diffusion



步骤2 在“文生图”标签下，输入相应的提示词和反向提示词（中英文均可），单击右侧的“生成”按钮，即可生成与提示词描述相符的图像。

图 4-26 Stable Diffusion WebUI 界面



须知

通过上述步骤创建的应用，仅可使用应用内置的默认模型进行AI绘画，如果您需要使用更多自定义模型，需要为应用挂载外部文件系统为应用提供持续使用能力，操作步骤请参考**上传自定义模型（可选）**。

----结束

4.5.5 绑定自定义域名（可选）

如需开放AI绘画Stable Diffusion应用的内网或外网访问，需要在创建成功的应用中绑定自定义域名。

前提条件

完成**部署和使用AI绘画Stable Diffusion应用**步骤，应用创建成功。

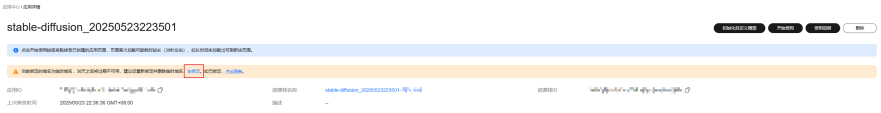
步骤一：准备自定义域名

申请公网域名，请通过域名注册商申请，确保域名可用即可。

步骤二：配置域名解析

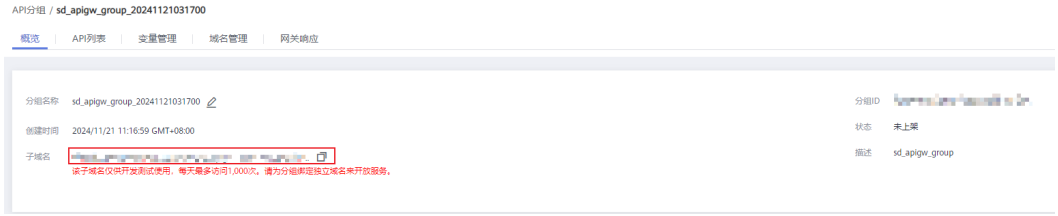
步骤1 应用创建成功后，如图4-27所示单击应用详情界面域名提示中的“去绑定”，弹出API网关的管理控制台页面。

图 4-27 进入绑定界面



步骤2 在弹出的API网关的管理控制台页面中，如图4-28所示，选择“概览”标签，单击复制“子域名”以备用。

图 4-28 复制子域名



步骤3 登录云解析服务DNS控制台，如图4-29所示，选择“公网域名”页签，单击已购买的域名所在行右侧的“管理解析”。

说明

如果使用非华为云注册的域名，可参考**创建公网域名**进行操作。

图 4-29 管理解析

步骤4 如图4-30所示，在“解析记录”页签下，单击“添加记录集”。

图 4-30 添加记录集

步骤5 在右侧弹出的“添加记录集”窗口配置以下信息：

- 记录类型：选择“CNAME - 将域名指向另外一个域名”。
- 主机记录：请参考**添加CNAME类型记录集**填写。
- 记录值：**步骤2**复制的子域名。

其他配置保持默认即可，如**图4-31**所示填写完成后单击“确定”即可完成域名解析。

图 4-31 配置记录集信息

----结束

步骤三：绑定自定义域名

步骤1 返回API网关的管理控制台页面，如图4-32所示，进入“域名管理”页签，单击“绑定独立域名”，输入自定义域名后单击“确定”。成功绑定后，可以将原先的临时域名进行“解除绑定”操作。

图 4-32 绑定独立域名

步骤2 返回函数工作流控制台中的应用详情界面，如图4-33所示，单击“点此刷新”，即可使用自定义域名访问应用。

图 4-33 刷新已绑定域名

----结束

4.5.6 上传自定义模型（可选）

FunctionGraph应用中心的AI绘画Stable Diffusion应用中内置了默认模型，如需使用更多自定义模型进行AI绘画，需要在创建成功的应用中初始化自定义模型挂载文件系统，即可上传自定义模型至文件系统进行AI绘画。

前提条件

1. FunctionGraph的云服务委托中包含“SWR Admin”、“VPC Administrator”、“Server Administrator”、“SFS FullAccess”和“SFS Turbo FullAccess”权限。
2. 完成**部署和使用AI绘画Stable Diffusion应用**步骤，应用创建成功。

步骤一：创建虚拟私有云 VPC 和子网

步骤1 登录**华为云网络控制台**，单击“创建虚拟私有云”，进入“创建虚拟私有云”界面。

步骤2 在“创建虚拟私有云”界面参考**表4-7**填写参数，其他参数保持默认即可。

表 4-7 VPC 和子网参数配置

参数类别	参数	参数说明	取值样例
基本信息	区域	必选参数。 VPC及其子网部署的区域。当前Stable Diffusion应用仅支持在“华东-上海一”部署。	华东-上海一

参数类别	参数	参数说明	取值样例
	名称	必选参数。 VPC的名称。要求如下： ● 长度范围为1~64位。 ● 名称由中文、英文字母、数字、下划线（_）、中划线（-）、点（.）组成。	vpc-fg
	IPv4网段	必选参数。 设置VPC的IPv4网段范围，可以根据页面建议选择，VPC网段的选择需要考虑以下两点： ● IP地址数量：要为业务预留足够的IP地址，防止业务扩展给网络带来冲击。 ● IP地址网段：当前VPC与其他VPC、云下数据中心连通时，要避免网络两端的IP地址冲突，否则无法正常通信。	192.168.x.x/16
	企业项目	必选参数。 企业项目管理提供了一种按企业项目管理云资源的方式，帮助您实现以企业项目为基本单元的资源及人员的统一管理，默认项目为default。	default
子网设置1	子网名称	必选参数。 子网的名称。要求如下： ● 长度范围为1~64位。 ● 名称由中文、英文字母、数字、下划线（_）、中划线（-）、点（.）组成。	subnet-fg
	可用区	必选参数。 在同一VPC网络内可用区与可用区之间内网互通，可用区之间能做到物理隔离。如业务需求高推荐选择多个可用区，本例以选择一个可用区为例。	可用区1（center）
	子网IPv4网段	必选参数。 子网的IPv4网段范围。子网的网段必须在VPC网段范围内，子网网段的掩码长度范围为“子网所在VPC的掩码~29”，可以根据页面建议选择。	192.168.x.x/24

步骤3 参数配置完成后，单击“立即创建”，完成虚拟私有云VPC和子网的创建。

----结束

步骤二：创建 SFS Turbo 文件系统

步骤1 登录**华为云弹性文件服务控制台**，选择“SFS Turbo”，单击“创建文件系统”，进入“创建文件系统”界面。

步骤2 在“创建文件系统”界面，参考**表4-8**填写参数。其他参数保持默认即可，如需使用其他参数请参考**创建SFS Turbo文件系统**。

表 4-8 文件系统参数说明

参数	参数说明	取值样例
计费模式	必选参数。 • 按需计费：适用于灵活使用场景。 • 包年/包月：适用于可预估资源使用周期的场景。	按需计费
区域	必选参数。 文件系统部署的区域。当前AI绘画应用仅支持在“华东-上海一”部署，且需与创建的虚拟私有云VPC保持一致。	华东-上海一
项目	必选参数。 项目部署的区域。根据区域选择默认同步设置。	华东-上海一（默认）
可用区	必选参数。 与创建的子网可用区保持一致。	可用区1
类型	必选参数。 根据推荐场景和实际情况选择文件系统类型和性能。本例支持选择所有文件系统类型，推荐选择适合大多数使用场景的 250 MB/s/TiB类型。	250 MB/s/TiB
容量	必选参数。 单个文件系统的最大容量。请根据实际需求选择，输入值应位于1.2至1023.6的区间内且必须为1.2的整数倍。	1.2
企业项目	必选参数。 与创建虚拟私有云VPC时的选择保持一致。	default
选择网络	必选参数。 文件系统所属的VPC和子网。选择**创建虚拟私有云VPC和子网**中创建的VPC与子网。	vpc-fg; subnet-fg(192.168.x.x/24)

参数	参数说明	取值样例
名称	必选参数。 文件系统的名称。要求如下： ● 长度范围为4~64位，并以字母开头。 ● 只能由英文字母、数字、下划线“_”和中划线“-”组成。	sfs-turbo-fg

步骤3 参数配置完成后，单击“立即创建”，再次确认信息后单击“提交”，等待文件系统创建任务提交成功即可。

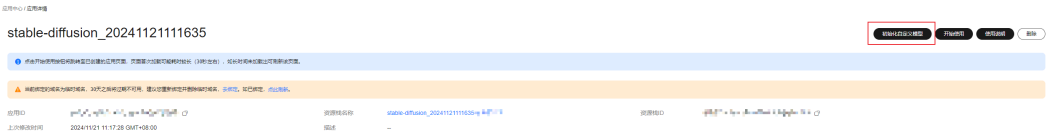
----结束

步骤三：初始化自定义模型

步骤1 登录函数工作流控制台，区域选择“华东-上海一”。在左侧导航栏选择“应用中心”，单击创建成功且需初始化的应用名称，进入应用详情页面。

步骤2 在应用详情页面，如图4-34所示单击“初始化自定义模型”，请仔细阅读弹出的说明后进行勾选操作，单击“确定”弹出初始化窗口。

图 4-34 初始化自定义模型



步骤3 在“初始化自定义模型”窗口填写如下参数：

- VPC：选择“vpc-fg（192.168.x.x/16）”。
- 子网：选择“subnet-fg（192.168.x.x/24）”。
- 文件系统来源：选择“SFS Turbo”。
- 文件系统名称：选择“sfs-turbo-fg”。

其他参数保持默认，如图4-35所示，配置完成后单击“确定”。

图 4-35 初始化自定义模型

步骤4 返回应用详情界面，“初始化自定义模型”按键变更为“上传模型”按键即初始化成功，单击“开始使用”进入WebUI界面，系统将在文件系统中自动创建与部署应用所需的目录和文件。

说明

成功进入Stable Diffusion WebUI界面后无需进行操作，此操作用于加载文件系统中的目录和文件，便于后续上传自定义模型。

----结束

步骤四：上传与加载自定义模型

步骤1 返回应用详情界面，单击“上传模型”进入文件管理页面，默认用户名和密码均为“admin”，登录后请在“设置”页签修改密码，保证数据安全。

步骤2 与上传自定义模型相关的部分关键目录如表4-9所示，可将模型文件上传到对应目录下。

表 4-9 部分关键目录路径

路径	用途
sd/models/Stable-diffusion	保存checkpoint模型文件。
sd/models/VAE	保存VAE模型文件。
sd/models/Lora	保存Lora模型文件。
sd/extensions	保存插件。

路径	用途
sd/outputs	保存生成结果。

步骤3 上传完成后，返回Stable Diffusion WebUI界面，单击页面中相应的“刷新”按钮加载模型，或重新打开WebUI界面。加载成功后即可查看和选择新增模型进行AI绘画。加载过程可能耗时较长，请耐心等待。

----结束

4.5.7 进阶使用：使用 ECS 作为 NFS 服务器实现多用户资源隔离

应用场景

FunctionGraph的函数除了支持挂载弹性文件系统SFS外，也支持挂载ECS服务器共享出的NFS共享路径。在多用户使用场景下，使用ECS能够有效地进行多用户的资源管理，通过配置特定的权限，满足用户之间需要强隔离的使用需求。

前提条件

1. 每位用户的FunctionGraph云服务委托中需包含“SWR Admin”、“VPC Administrator”、“Server Administrator”、“SFS FullAccess”和“SFS Turbo FullAccess”权限。
2. 每位用户均需完成**部署和使用AI绘画Stable Diffusion应用**步骤，应用创建成功。
3. 完成**创建虚拟私有云VPC和子网**。

步骤一：购买 ECS 服务器

进入**购买ECS服务器**页面，购买过程中请参照以下注意事项。实例类型和是否开启公网访问可根据业务需要自行选择，其他参数可参考**设置ECS购买参数**。

- 基础配置：如**图4-36**所示，本示例使用“按需计费”，区域选择“华东-上海一”。

图 4-36 基础配置

- 操作系统：本示例镜像选用EulerOS 2.5 64bit(40GiB)。

说明

不同镜像版本下，部分Linux命令可能存在差异。

- 储存与备份：鉴于多数模型文件的大小在1GB到10GB以上，建议依据具体需求选择系统盘容量，并参考**图2 系统盘选择**新增数据盘进行挂载。

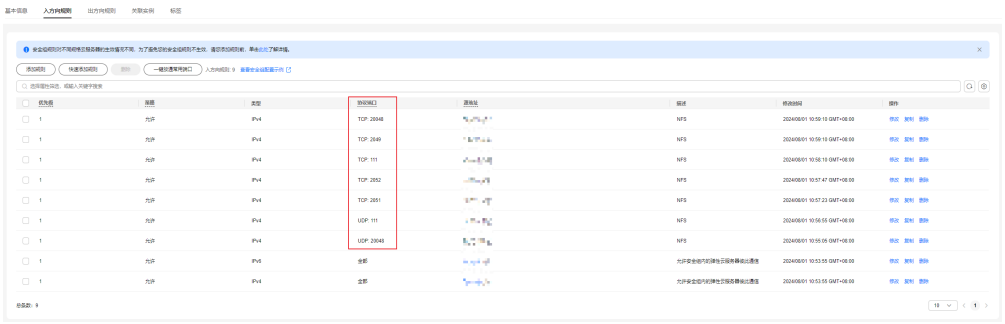
图 4-37 系统盘选择

- 网络：虚拟私有云与主网卡请选择**创建虚拟私有云VPC和子网**中所创建VPC与子网，如**图4-38**所示。

图 4-38 网络配置

- 安全组：请参考**图4-39**新建安全组，入方向规则允许子网内IP访问端口111、2049、2051、2052、20048以支持NFS服务；其他端口，如22端口用于SSH和SFTP、21端口用于FTP等，具体协议端口和源地址请根据实际需要配置。

图 4-39 安全组设置



步骤二：设置 ECS 下的 NFS 共享

ECS购买完成后即可进行NFS共享设置，以下示例中以两个用户user1和user2的场景进行介绍，可根据实际需要增减用户数。

1. 添加用户user1和user2并创建home目录。
```
useradd -m user1 && useradd -m user2
```
2. 修改user1和user2的密码。
```
passwd user1
passwd user2
```


3. 创建用户的共享目录，并修改共享目录操作权限为777。
```
mkdir /home/user1/share && chmod 777 /home/user1/share
mkdir /home/user2/share && chmod 777 /home/user2/share
```

> **说明**
>
> 共享目录作为用户home目录的子目录，限制其他用户操作，确保函数挂载该目录后拥有操作权限。因此，设置权限为777不会导致权限过度。

4. 安装NFS服务。
```
yum install rpcbind nfs-utils // 使用apt或其他包管理工具的镜像请使用相应的命令
```

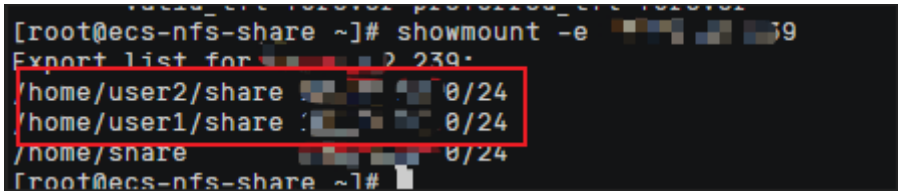
5. 编辑/etc/exports，写入如下内容：
```
/home/user1/share xx.xx.xx.xx/xx(rw) // 网段处请填写之前创建的子网网段
/home/user2/share xx.xx.xx.xx/xx(rw) // 网段处请填写之前创建的子网网段
```

6. 启动NFS服务。
```
systemctl start rpcbind nfs
```

7. 设置开机自启。
```
echo "xx.xx.xx.xx:/home/user1/share /nfs nfs4 defaults 0 0" >> /etc/fstab // IP处请填ECS在子网中的IP
echo "xx.xx.xx.xx:/home/user2/share /nfs nfs4 defaults 0 0" >> /etc/fstab // IP处请填ECS在子网中的IP
mount -av
```

8. 查看共享信息，显示如图4-40所示即表示创建NFS共享成功。
```
showmount -e xx.xx.xx.xx  // IP处请填搭建服务器主机的私有地址
```

图 4-40 查看共享信息



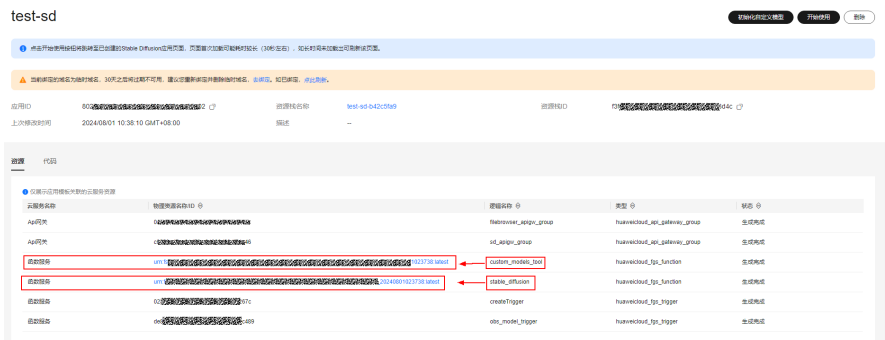
步骤三：在 Stable Diffusion 应用中挂载 ECS

登录函数工作流控制台，进入应用中心，用户user1和user2已分别在各自的账号下成功创建了一个AI绘画Stable Diffusion应用。以下以user1为例进行介绍，user2的操作步骤相同。

步骤1 user1进入应用详情页，在“资源”列表中分别找到逻辑名称为“stable_diffusion”和“custom_models_tool”的函数资源，如图4-41所示，单击链接进入函数详情页面。

两个函数的操作相同，此处以“stable_diffusion”函数为例。

图 4-41 user1 的函数服务



步骤2 进入函数详情页，如图4-42所示，单击“设置 > 网络配置”，打开“函数访问VPC内资源”开关并配置VPC和子网，请选择**步骤一：购买ECS服务器**中使用的VPC和子网，单击“保存”。

图 4-42 网络配置

步骤3 在“设置”页签的左侧导航栏中，单击“磁盘挂载 > 添加挂载”，配置完成后单击“确定”。

- 文件系统来源：选择“ECS”。
- 云服务器名称：选择**步骤一：购买ECS服务器**创建的云服务器。
- 共享目录路径：填写“/home/user1/share”（user2的应用则填写/home/user2/share）。
- 函数访问路径：填写“/mnt/auto”。

说明

如已挂载SFS Turbo文件系统，可在成功挂载ECS后取消挂载SFS Turbo文件系统，并及时释放SFS Turbo文件系统资源避免持续收费。

步骤4 参照**步骤1**~**步骤3**完成“custom_models_tool”的函数设置。

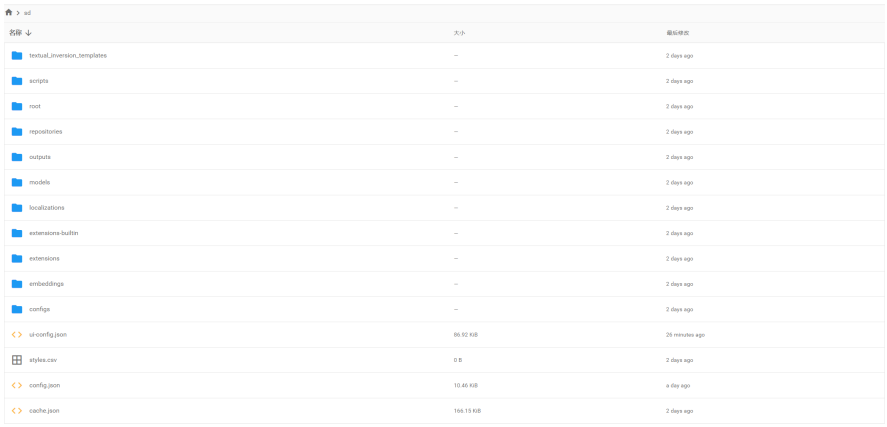
----结束

步骤四：上传与加载模型

步骤1 返回应用详情页，单击“开始使用”进入WebUI界面。函数会自动在挂载目录中创建应用所需目录和文件。

步骤2 页面成功加载后，返回应用详情页，单击“上传模型”打开文件管理工具，默认用户名和密码均为“admin”，登录后请在“设置”页签修改密码，保证数据安全。与应用相关的sd目录内容如图4-43所示。

图 4-43 文件管理工具



步骤3 将模型、插件等文件分别上传至对应目录。部分关键目录可参考**表4-9**。

步骤4 重新加载Stable Diffusion WebUI界面，即可看到新传入的模型。

步骤5 单击右上角“生成”开始进行AI绘画，如**图4-44**所示，结果图片会自动保存到“/home/user1/share/sd/outputs/txt2img/202x-xx-xx”目录下。

图 4-44 图片保存目录

----结束

4.5.8 进阶使用：通过挂载同一 SFS 文件系统实现多用户资源共享

应用场景

模型文件普遍占用大量内存，在多用户使用场景下，每个用户复制一份模型文件会导致不必要的存储空间浪费。通过将不同用户的应用挂载到同一SFS文件系统下，能够满足多人共享模型文件资源的需求，同时为了避免不同用户使用期间的推理结果相互影响，可以在Stable Diffusion WebUI中修改结果保存路径。

前提条件

1. 每位用户的FunctionGraph云服务委托中需包含“SWR Admin”、“VPC Administrator”、“Server Administrator”、“SFS FullAccess”和“SFS Turbo FullAccess”权限。
2. 每位用户均需完成**部署和使用AI绘画Stable Diffusion应用**步骤，应用创建成功。
3. 每位用户均需使用同一个SFS文件系统完成**初始化自定义模型挂载文件系统**，所挂载的SFS文件系统下已存在sd目录。

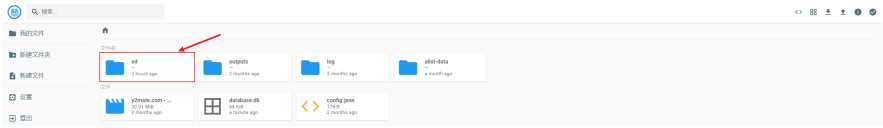
步骤一：创建多用户配置文件

本示例以两个用户user1和user2的使用场景进行介绍，可根据实际需要增减用户数。

步骤1 选择任一用户，进入已完成初始化自定义模型的Stable Diffusion应用详情界面，单击“上传模型”，登录后进入文件系统。

步骤2 如图4-45所示，进入sd目录。

图 4-45 进入 sd 目录



步骤3 如图4-46所示，找到config.json文件，选中并复制到任意目录，此处仍置于sd目录下，重命名为config_user1.json。如图4-47所示，同样为用户user2复制一份，命名为config_user2.json。

图 4-46 复制 config.json 文件

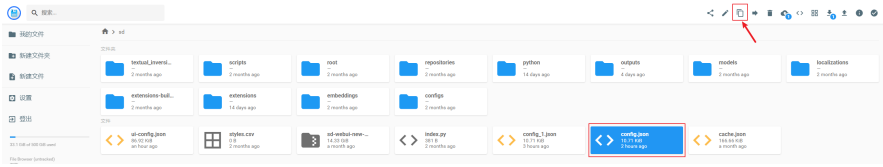
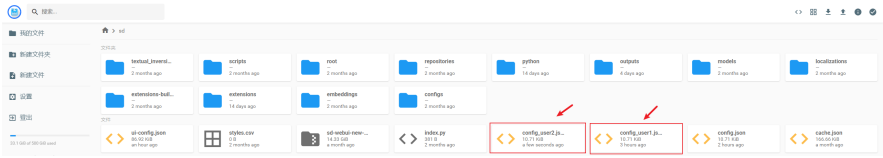


图 4-47 复制生成 config_user1.json 和 config_user2.json 文件



----结束

步骤二：修改环境变量使用新配置文件

步骤1 进入Stable Diffusion应用详情页，在“资源”列表中找到逻辑名称为“stable_diffusion”的函数资源，单击链接进入函数详情页面。

步骤2 在“设置 > 环境变量”页签，单击“编辑环境变量”，在弹出窗口中单击“添加环境变量”。user1参考表4-10填写，user2参考表4-11填写，填写完成后单击“确定”。

表 4-10 user1 使用新配置文件的环境变量

键	值
EXTRA_ARGS	--ui-settings-file=/mnt/auto/sd/config_user1.json

表 4-11 user2 使用新配置文件的环境变量

键	值
EXTRA_ARGS	--ui-settings-file=/mnt/auto/sd/config_user2.json

----结束

说明

如需同时**启用WebUI认证**和**使用API模式访问应用**，环境变量内容可同时设置，请参考**表4-14**中环境变量值的形式进行设置。

步骤三：修改结果保存路径

完成上述配置后，两位用户即可共享同一SFS文件系统下的模型文件。为进一步隔离不同用户的推理结果，可在WebUI中选择“设置 > 保存路径”，修改自己的结果保存路径。

4.5.9 进阶使用：启用 WebUI 认证

通过应用中心部署的Stable Diffusion应用为便于快速体验，默认未启用WebUI认证。为避免域名泄露导致函数被滥用，建议通过**配置函数环境变量**的方式启用WebUI认证。

启用 WebUI 认证

步骤1 进入Stable Diffusion应用详情页，在“资源”列表中找到逻辑名称为“stable_diffusion”的函数资源，单击链接进入函数详情页面。

步骤2 在“设置 > 环境变量”页签，单击“编辑环境变量”，在弹出窗口中单击“添加环境变量”，添加下表信息，填写完成后单击“确定”。

表 4-12 启用 WebUI 认证的环境变量

键	值	说明
EXTRA_ARGS	--gradio-auth user1:password1	“user1”处填写用户名，“password1”处填写需要设置的密码。

步骤3 设置完成后，访问WebUI时将需要输入您设置的用户名和密码才可以进行绘图操作。

----结束

4.5.10 进阶使用：使用 API 模式访问应用

通过应用中心部署的Stable Diffusion应用默认未启用API访问，可以通过**配置函数环境变量**的方式启用。

使用 API 模式访问应用

步骤1 进入Stable Diffusion应用详情页，在“资源”列表中找到逻辑名称为“stable_diffusion”的函数资源，单击链接进入函数详情页面。

步骤2 在“设置 > 环境变量”页签，单击“编辑环境变量”，在弹出窗口中单击“添加环境变量”，添加下表信息，填写完成后单击“确定”。

表 4-13 使用 API 模式访问应用的环境变量

键	值	说明
EXTRA_ARGS	--api --api-auth username1:password1,username2:password2 --nowebui	“username1”和“username2”处填写用户名，“password1”和“password2”处填写需要设置的密码，通过英文逗号（,）分隔多用户的用户名和密码。

步骤3 设置完成后，使用API模式访问应用时将需要输入您设置的用户名和密码。

----结束

配置并发参数

请参考**配置函数的并发处理**配置并发参数，WebUI模式和API模式的推荐参数如下：

- WebUI模式
  - 单实例并发数：>=100。经测试，在单人使用场景下单实例并发量为15左右，如果是多人使用场景推荐将单实例并发数设置为100以上。
  - 单函数最大实例数：1。在WebUI模式下，出图过程中会持续监控任务进度。若存在多个实例，可能引发请求混乱，进而造成进度显示与最终结果呈现的障碍。因此，需将单函数最大实例数设定为1。
- API模式
  - 单实例并发数：1-5。确保单一实例不会积累过多的待处理请求，当并发量达到上限时，将触发新实例的启动，以保证图像生成的效率。
  - 单函数最大实例数：默认400。可根据实际需要调整。

使用 API 模式访问应用并启用 WebUI 认证

如需同时使用API模式访问应用和启用WebUI认证，环境变量内容可参考**表4-14**设置。

表 4-14 使用 API 模式访问应用并启用 WebUI 认证

键	值	说明
EXTRA_ARGS	--api --api-auth user1:password1 --gradio-auth user1:password1	“username1”和“username2”处填写用户名，“password1”和“password2”处填写需要设置的密码，通过英文逗号（,）分隔多用户的用户名和密码。

4.5.11 免责声明

1. 本应用使用到的**Stable-Diffusion**、**Stable-Diffusion-WebUI**以及**镜像构建工程**等项目均为社区开源项目，关于开源项目的问题还需用户到开源社区寻求帮助或者自行解决，华为云仅提供算力支持。
2. 本实践仅作为简易示例供用户参考和学习，如需应用于实际生产环境，请参考镜像构建工程自行完善和优化。使用过程中遇到的函数工作流问题，可通过**提交工单**进行咨询。
3. 本应用部署后会为您创建APIG网关，根据有关规定，建议在应用创建成功后根据提示绑定自定义域名，使用您的自有域名访问WebUI界面。

4.6 使用 FunctionGraph 快速部署 MCP Server

什么是 MCP Server

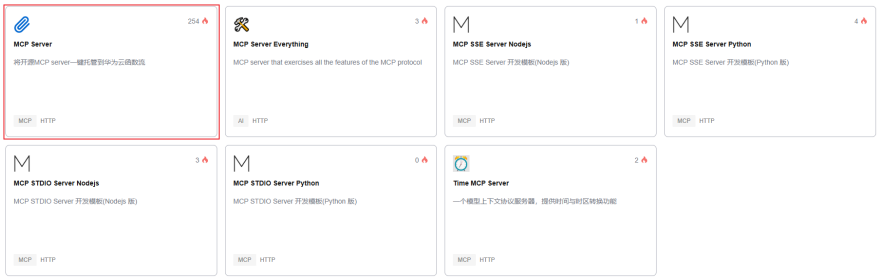
MCP（Model Context Protocol，模型上下文协议）是一种开源协议，旨在以标准化的方式向大语言模型（LLM）提供上下文信息。MCP Server（即MCP服务器）基于模型上下文协议运行，能够使大型语言模型与外部数据源及工具无缝集成，通过标准化的交互，帮助模型获取丰富的上下文信息。其应用类型广泛，例如文件系统服务器可协助AI分析项目文件，Web搜索服务器能帮助AI获取最新信息等。

使用 FunctionGraph 快速部署 MCP Server 方案概述

在企业数字化转型过程中，采用传统的云服务器本地部署模型时，需要预先评估流量峰值规划所需资源。但由于业务流量的不确定性，这类静态资源配置方式容易导致服务器利用率不足，造成资源闲置，进而影响成本效益。FunctionGraph作为华为云的Serverless函数计算服务，利用其资源弹性优势，为MCP Server的托管提供了一种高效、灵活且可靠的解决方案。通过Serverless架构，FunctionGraph能够根据实际流量自动调整资源分配，提高资源利用率，减少资源闲置，优化成本。

FunctionGraph应用中心提供了一键部署热门开源MCP Server的应用模板，支持通过API网关（APIG）对外提供服务。简化了部署流程的同时，FunctionGraph能够自动处理日志与监控，使开发者能够专注于核心业务逻辑的开发。

图 4-48 FunctionGraph 应用中心 MCP Server 模板



约束与限制

- 当前使用FunctionGraph应用中心部署MCP Server应用仅限于“华北-北京四”区域，请确保该实践中的相关资源均部署于该区域。

- 单个应用仅支持运行一个MCP Server实例。

资源和成本规划

表1介绍使用FunctionGraph函数部署MCP Server应用所需的资源和成本规划。

表 4-15 资源和成本规划

资源	资源说明	计费说明
函数工作流 FunctionGraph	• 函数类型：HTTP函数 • 示例区域：华北-北京四 • 创建量：1	• 计费模式：按需计费。 • 函数工作流提供免费试用，每月前100万次调用免费。具体计费项及说明请参考**函数工作流按需计费说明**。
API网关 APIG	• 版本：专享版API网关 • 示例区域：华北-北京四 • 公网入口：开启 • 购买量：1	• 计费模式：本例使用按需计费。 • 实例规格和入公网带宽请根据业务需求选择，具体计费项及标准请参考**专享版API网关计费模式概述**。

步骤一：创建 APIG 专享版实例

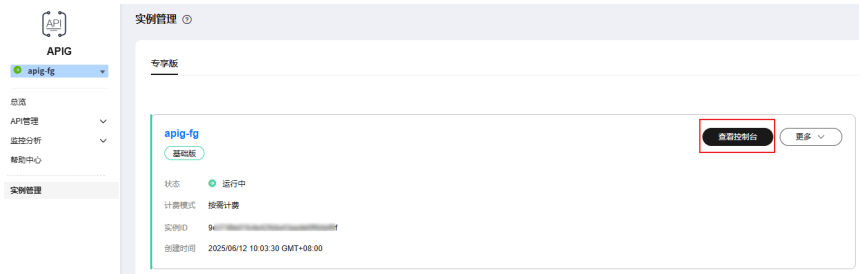
1. 进入API网关控制台**购买实例**页面，请参考**创建APIG实例**购买任意规格的专享版APIG实例。

   创建过程中请参照以下注意事项，创建一个名为“apig-fg”的APIG专享版实例，其他参数可根据实际需要选择：

   - 区域选择“华北-北京四”。
   - 需**开启公网入口**，并根据实际需求设置入公网带宽。

2. 在APIG控制台左侧导航栏选择“实例管理”，单击“查看控制台”。

   图 4-49 查看 APIG 实例控制台

3. 选择“配置参数”页签，单击“sse_strategy”参数右侧的“编辑”，将参数运行时修改为“On”并单击“保存”，开启SSE传输策略开关。

步骤二：创建 MCP Server 应用

1. 登录**函数工作流控制台**，区域选择“华北-北京四”。
2. 在左侧导航栏中选择“应用中心”，单击“创建应用”进入选择模板界面。
3. 找到“MCP Server”模板，单击“使用模板”，进入创建应用界面。
4. 参考**表4-16**配置应用参数，配置完成后单击“立即创建”。

图 4-50 创建 MCP Server

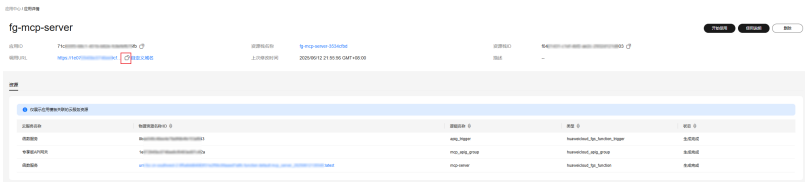
表 4-16 创建 MCP Server 应用配置说明

参数	取值样例	说明
模板	MCP Server	默认展示已选择的函数模板。如需更换函数模板，请单击“重新选择”。
区域	华北-北京四	选择应用创建的区域。本应用支持“华北-北京四”区域创建。 不同区域的资源之间内网不互通，请就近选择靠近您业务的区域，可以降低网络时延、提高访问速度。
应用名称	fg-mcp-server	输入自定义的应用名称。 可包含字母、数字、下划线和中划线，以大/小写字母开头，以字母或数字结尾，长度在2-60个字符之间。

参数	取值样例	说明
运行时语言	http	默认展示该模板内置的运行时语言，无法更换。
委托	fgs-app-adminagency	选择选择函数的委托，通过委托函数工作流来访问其他云服务。 若未创建过函数应用中心默认委托“fgs-app-adminagency”，可先选择“未使用任何委托”，完成其他配置项后单击“立即创建”，系统会弹出提示创建名称为“fgs-app-adminagency”的委托以保证应用正常创建。
创建仓库	关闭	开启时，应用创建时会为您在代码托管服务创建代码仓库，之后您可以通过更新代码并部署来更新您的应用。
描述（可选）	-	自定义填写函数应用的描述。 最长支持填入1024个字符。
APIG实例	apig-fg	选择**步骤一：创建APIG专享版实例**创建的APIG实例。
运行环境	uvx	支持选择以下两种运行环境，可根据实际情况选择： • npx：基于Node.js生态，无需全局安装依赖，通过临时调用npm包执行命令启动。 • uvx：在隔离环境中临时安装并运行Python包提供的命令行工具。
MCP服务配置	{ "mcpServers": { "fetch": { "command": "uvx", "args": ["mcp-server-fetch"] } } }	使用JSON格式填写MCP服务配置，可根据实际情况自定义配置。 JSON配置文件定义了如何从 MCP 服务器获取数据，以下为本示例的JSON配置文件参数解释： • “mcpServers”：配置文件的主对象，表示与 MCP 服务器相关的配置。 • “fetch”：“获取”（fetch）操作的相关配置，定义如何从 MCP 服务器获取数据。 • “command”：指定执行获取操作的命令名称或工具名称。本例使用uvx运行环境。 • “args”：为“command”命令提供的参数列表，用于告知uvx命令需要执行的具体任务。

5. 应用成功创建后，单击复制“调用URL”可用于客户端。
   测试环境默认提供30天临时域名，实际生产环境请准备自定义域名并绑定使用。

图 4-51 MCP Server 应用

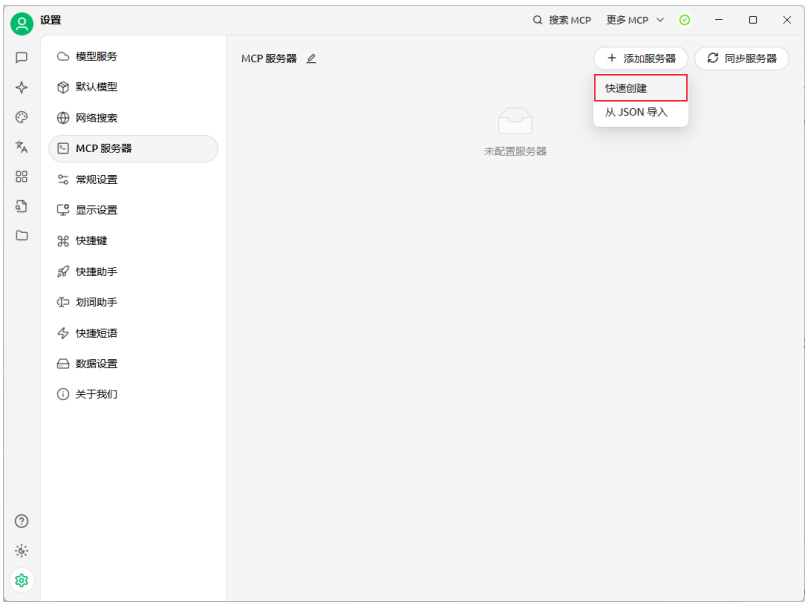


步骤三：配置客户端开始 AI 对话

本例以Cherry Studio作为客户端进行AI对话，请先安装适用于您设备的Cherry Studio客户端，并确保已配置合适的模型能在Cherry Studio使用模型进行基本对话。

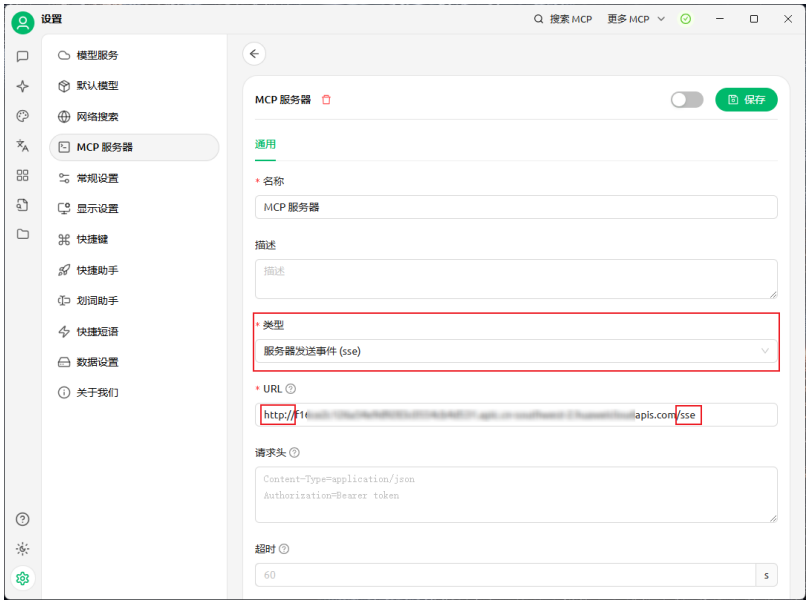
1. 打开Cherry Studio客户端，进入设置界面，单击“MCP服务器”，选择“添加服务器 > 快速创建”。

图 4-52 添加服务器



2. 类型选择“服务器发送事件（sse）”，“URL”处输入5复制的URL，将*https*改为*http*，并在URL末尾添加*sse*，配置完成如图4-53所示，单击“保存”。

图 4-53 配置 MCP 服务器

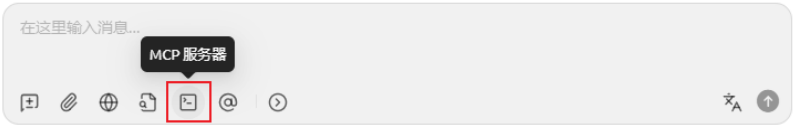


3. 配置完成后可进入助手界面开始AI对话。如图4-54所示，当前未启用MCP服务器进行模型对话，大模型未给出正常答案。

图 4-54 未启用 MCP Server 对话

4. 单击聊天框处的MCP服务器按钮，选择2配置的MCP服务器。

图 4-55 配置 MCP 服务器



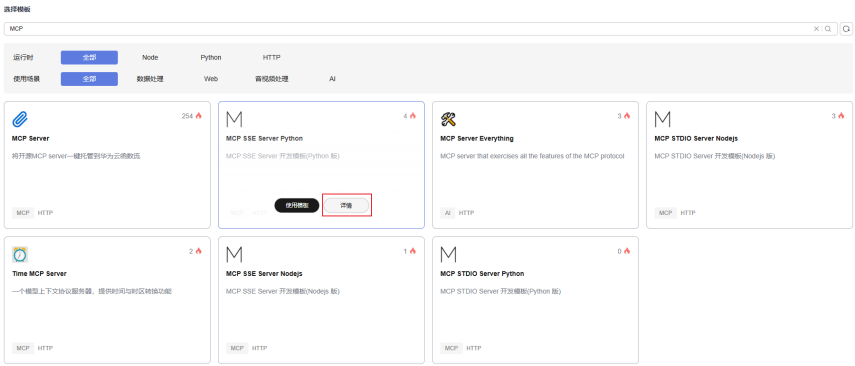
5. 启用MCP服务器后再次对话，如图4-56所示，模型调用fetch工具读取官方文档链接内容后正确回答了问题。

图 4-56 启用 MCP 服务器对话

更多应用中心 MCP 相关模板

FunctionGraph应用中心提供了多类热门的MCP应用模板，助力您在FunctionGraph上迅速开发MCP相关业务。可单击模板的“详情”查看使用说明。

图 4-57 MCP 相关模板



免责声明

- 本应用通过**supergateway**、**mcp-proxy**和**镜像构建工程**开源项目实现功能转换，不承担开源项目相关责任，关于开源项目的问题请访问开源社区解决，华为云仅提供算力支持。
- 本实践作为简易示例供参考和学习，如需应用于实际生产环境，请进行充分测试并评估成本，使用过程中遇到的函数工作流问题，可通过**提交工单**进行咨询。

5 函数构建类实践

5.1 使用已有 SpringBoot 项目构建 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**RESTful Web Service**项目为例，使用HTTP函数的方式部署到FunctionGraph上。

操作流程

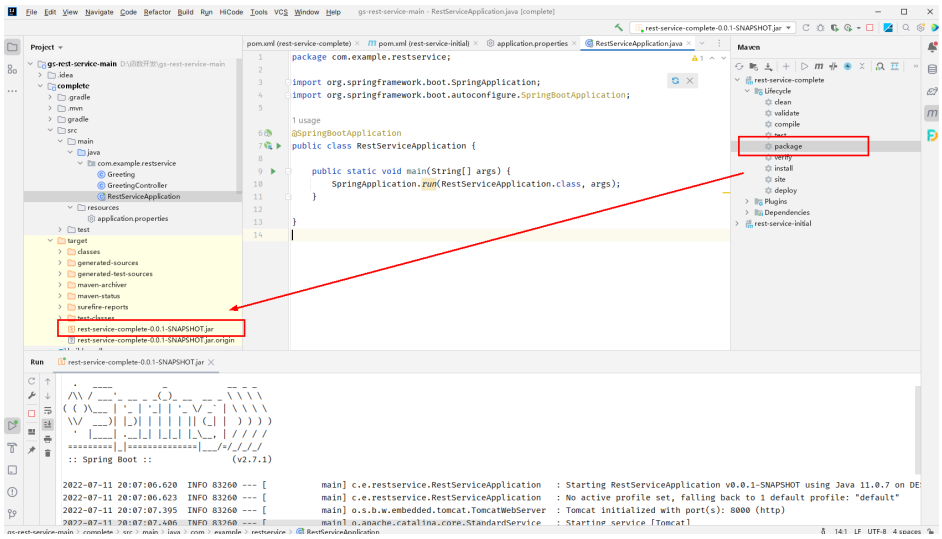
将既有项目部署到FunctionGraph通常只需要：修改项目监听端口号为8000，然后在jar包同目录创建bootstrap文件写入执行jar包的命令。

本案例使用IntelliJ IDEA，Maven项目。

构建代码包

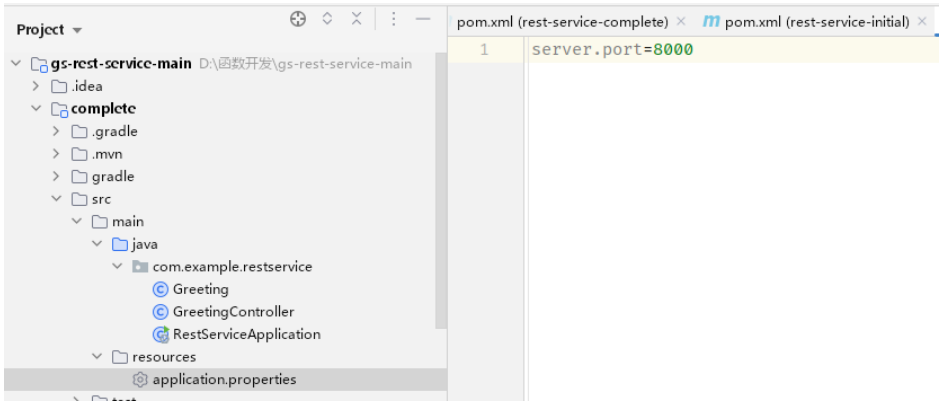
1. 打开Springboot项目，在maven插件处单击package，生成jar包。

图 5-1 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 5-2 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
```

说明

函数中可直接调用Java运行环境，无需另外安装。

4. 选中jar包和bootstrap文件，打包成zip包。

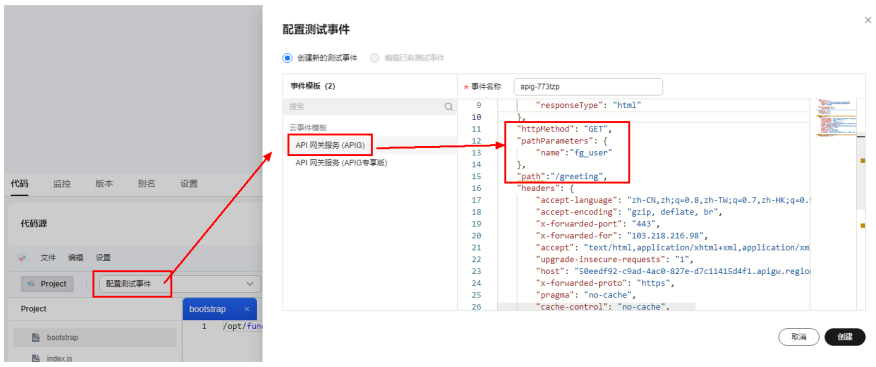
创建HTTP函数并上传代码

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，修改测试事件中的path、pathParameters参数，构建一个简单的Get请求。

图 5-3 配置测试事件

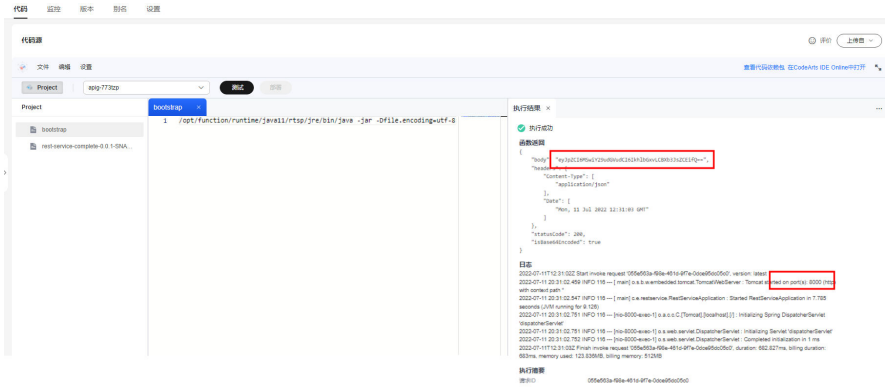


c. 单击“创建”，完成测试事件创建。

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。

图 5-4 查看函数返回结果

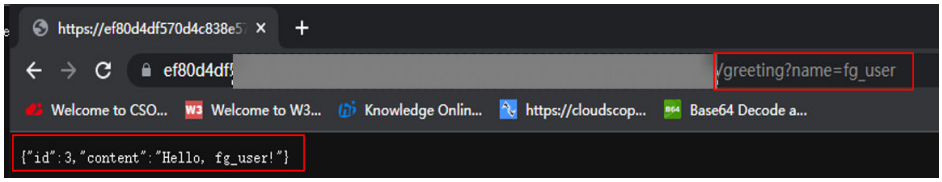


- 配置APIG触发器测试

  a. 请参见**使用APIG触发器**，创建APIG专享版触发器，“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 5-5 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 5-6 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。

   图 5-7 到 LTS 进行日志分析

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>
```

5.2 使用 Go 语言程序构建 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FunctionGraph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专享版触发器

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

图 5-8 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 5-9 请求结果

5.3 使用 FunctionGraph 函数访问 RDS for MySQL

5.3.1 使用 FunctionGraph 函数访问 RDS for MySQL 案例概述

场景介绍

在FunctionGraph中，不同函数实例间不共享状态，而数据库可实现结构化数据的持久化存储，进而实现状态共享。通过FunctionGraph访问云上数据库，可执行数据查询和数据插入等操作。

本文介绍如何在 FunctionGraph 中高可靠地访问RDS for MySQL并进行数据查询操作，同时提供**示例代码**供测试使用。示例代码中运用了**数据库连接池和重试机制**，可有效提升数据库操作的性能与可靠性，以此展示在 FunctionGraph 中安全和高RDS for MySQL 数据库的方法。

资源与成本规划

表1介绍使用FunctionGraph访问RDS for MySQL实践所需的资源和成本规划。

表 5-1 资源和成本规划

资源	资源说明	计费说明
函数工作流 FunctionGraph	• 函数类型：事件函数 • 示例区域：华东-上海一 • 创建量：1	• 计费模式：按需计费。 • 函数工作流提供免费试用，每月前100万次调用免费。具体计费项及说明请参考**函数工作流按需计费说明**。
云数据库 RDS	• 示例区域：华东-上海一 • 资源选配：MySQL • 购买量：1	• 计费模式：按需计费。 • 具体性能规格配置请根据业务需求选择，计费项及计费标准请参考**云数据库RDS for MySQL按需计费说明**。
虚拟私有云 VPC	• 示例区域：华东-上海一 • 子网数量：1 • 安全组数量：1 • 购买量：1	• 虚拟私有云：免费。 • 子网：免费。 • 安全组：免费。

操作流程

介绍使用FunctionGraph函数访问RDS for MySQL的整体操作流程，具体操作指导请参考**5.3.2 使用FunctionGraph函数访问RDS for MySQL操作步骤**。

表 5-2 函数访问 RDS for MySQL 操作流程

操作步骤	步骤说明
前提条件	进行本实践前，需已有可用的VPC网络环境、RDS for MySQL实例及其数据库和表；并已创建包含“VPC Administrator”权限的函数委托。
步骤一：创建函数依赖包	本实践使用Python示例代码实现数据库连接访问，代码依赖pymysql和DBUtils包，需将这些依赖包上传至函数工作流控制台，供后续函数调用。
步骤二：创建函数	在函数工作流控制台，创建用于访问RDS for MySQL的函数。
步骤三：配置函数	进入已创建函数的详情页中配置函数代码、依赖包及相关函数设置。
步骤四：测试函数	测试函数是否能成功访问RDS for MySQL实例中数据库表的记录。

5.3.2 使用 FunctionGraph 函数访问 RDS for MySQL 操作步骤

前提条件

- 已**创建虚拟私有云基本信息及默认子网**。
- 已**购买RDS for MySQL实例**，已**创建数据库**且数据库中有可用的表。RDS实例需与函数所在区域一致，本实践以“华东-上海一”区域为例。
- 已**创建函数委托**，委托中包含“VPC Administrator”权限。

步骤一：创建函数依赖包

表 5-3 依赖包文件下载

依赖包	依赖包概述	下载链接
pymysql	用Python编写的MySQL数据库连接器，其作用是让Python程序能够和MySQL数据库进行通信。	**pymysql_py36-1.zip**（**sha256校验文件**）
DBUtils	数据库连接池工具包，可以对数据库连接进行管理和复用。	**dbutils_py36-1.zip**（**sha256校验文件**）

1. 登录**函数工作流控制台**，区域选择“华东-上海一”。
2. 在左侧导航栏选择“函数 > 依赖包管理”，进入“依赖包管理”界面。
3. 单击“创建依赖包”，进入创建依赖包界面，参考**表5-4**分别创建“pymysql_py36”和“dbutils_py36”依赖包。

   如**图5-10**所示以创建pymysql依赖包为例，dbutils依赖包创建参数相同。

表 5-4 创建依赖包参数说明

参数	取值	参数说明
依赖包名称	pymysql_py36	自定义依赖包名称，便于识别不同的依赖包。 可包含字母、数字、下划线、点和中划线，长度不超过96个字符。以大/小写字母开头，以字母或数字结尾。
运行时	Python 3.6	选择依赖包的运行时语言。
代码上传方式	上传ZIP文件	分为上传ZIP文件和从OBS上传ZIP文件。 • 上传ZIP文件：单击“添加文件”，上传ZIP代码文件。**上传的文件大小限制为10MB。** • 从OBS上传文件：填写“OBS链接URL”。如何在OBS中上传文件对象请参见**上传对象**，OBS存储链接获取方法请参见**通过URL访问对象**。
文件上传	单击“添加文件”，上传**表5-3**下载的zip包。	选择上传ZIP文件后出现此参数，单击上传本地的ZIP文件。
描述	-	自定义填写依赖包的描述信息。

图 5-10 创建依赖包

步骤二：创建函数

1. 返回函数工作流控制台，在左侧导航栏选择“函数 > 函数列表”，单击“创建函数”。
2. 参考**表5-5**填写函数基本信息，基本信息填写完成如**图5-11**所示。

表 5-5 创建函数基本信息

参数	取值样例	参数说明
函数类型	事件函数	事件函数为可通过特定事件触发的函数，通常为JSON格式的请求事件。
区域	华东-上海一	选择函数所在的区域。本实践以“华东-上海一”区域为例。
函数名称	access-mysql-demo	输入自定义的函数名称，命名规则如下： ● 可包含字母、数字、下划线和中划线，长度不超过60个字符。 ● 以大/小写字母开头，以字母或数字结尾。
企业项目	default	选择函数所属的企业项目。企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 默认为“default”，支持选择已创建的企业项目。 **如果您没有开通企业管理服务，将无法看到企业项目选项。开通方法请参见如何开通企业项目。**
委托	VPC	选择函数的委托。通过委托来授权函数工作流来访问其他云服务。 本实践需配置函数访问VPC内资源，因此需选择包含“VPC Administrator”权限的委托。
运行时	Python 3.6	选择用于编写函数的运行时语言。本实践以Python 3.6为例。

图 5-11 创建函数基本信息

3. 参考**表5-6**填写函数高级设置信息，填写完成如**图5-12**所示，单击“立即创建”。

表 5-6 创建函数高级设置

参数	取值样例	参数说明
函数访问公网	未开启	开启时，函数可以通过默认网卡访问公网上的服务，其公网访问带宽为用户间共享，仅适用于测试场景。
函数访问VPC内资源	开启。 • VPC：vpc-fg（192.168.x.x/x） • 子网：subnet-fg（192.168.x.x/x）	开启后可选择函数需访问的VPC及其子网。**需选择与所创建RDS实例相同的VPC环境。** 开启时，函数将使用配置的VPC所绑定的网卡进行网络访问，同时禁用函数工作流的默认网卡，即开关“函数访问公网”参数将不生效。
日志记录	未开启	启用日志功能后，函数运行过程中产生的日志会上报到云日志服务（LTS）。 LTS将按需收取日志管理费用，详情请参见**云日志服务价格详情**。

图 5-12 创建函数高级设置

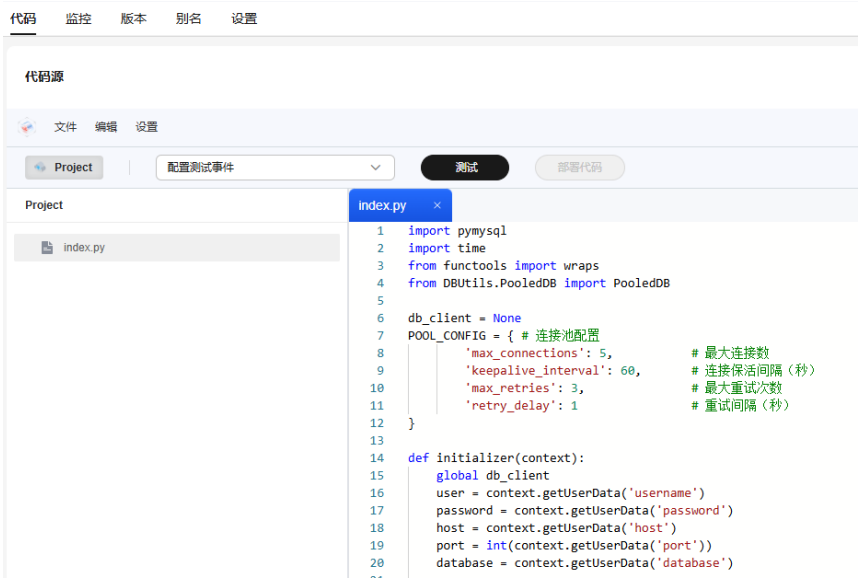
步骤三：配置函数

1. 成功创建函数后进入函数详情页，复制**5.3.3 函数访问RDS for MySQL示例代码**，如**图5-13**所示粘贴在代码在线编辑器中覆盖“index.py”文件中的代码，单击“部署代码”。

须知

本实践的示例代码实现了从RDS for MySQL实例数据库的user表中查询前10条记录的功能，请根据您数据库中的实际表名进行代码内容修改。

图 5-13 部署代码



2. 在“代码”页签下，滑至页面最底部的“代码依赖包”板块，单击“添加依赖包”。
3. 在“选择依赖包”弹窗，如**图5-14**所示，“依赖包源”选择“私有依赖包”，分别添加**步骤一：创建函数依赖包**创建的“pymysql_py36”和“dbutils_py36”依赖包。

图 5-14 选择依赖包

4. 添加完成后如图5-15所示。

图 5-15 代码依赖包

5. 单击“设置”页签，选择“常规设置”，调整“执行超时时间”为“30”，“内存”选择“256”，单击“保存”。

图 5-16 函数常规设置

6. 在“设置”页签下选择“环境变量”，单击“编辑环境变量”，在弹窗中单击添加环境变量，参考表5-7的说明列添加环境变量，添加完成后如图5-17所示，单击“确定”。

表 5-7 环境变量

键	值（示例）	说明
host	192.168.x.x	数据库实例的访问地址。 • 如果您选择**同VPC内的RDS数据库**的场景，请将此环境变量值设置为数据库的**内网地址**。 • 如果您选择**跨VPC或跨地域访问RDS数据库**的场景，请将此环境变量值设置为数据库的**外网地址**。 RDS控制台操作步骤：进入目标RDS实例详情页，在左侧导航栏选择“连接管理”，在连接信息处获取数据库的内网地址或外网地址。
database	db_test	RDS实例中创建的数据库名称。
password	******	RDS实例中设置的数据库密码。 密码为敏感信息，建议开启加密参数。
port	3306	RDS实例中设置的数据库端口。
username	fg_user	RDS实例中创建的账号名称。

图 5-17 编辑环境变量

7. 在“设置”页签下选择“生命周期”，如图5-18所示开启初始化配置，设置“初始化超时时间”为“60”，“函数初始化入口”为“index.initializer”，单击“保存”。

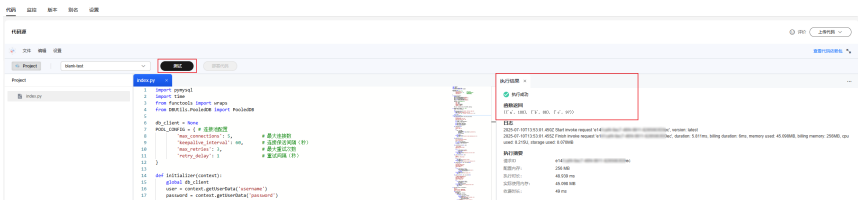
图 5-18 初始化配置

步骤四：测试函数

1. 回到“代码”页签，单击“测试”弹出配置测试事件弹窗，选择“空白模板”，无需修改单击“创建”。
2. 如图5-19所示单击“测试”，查看函数执行结果。

   根据示例代码，函数返回连接的RDS for MySQL实例数据库的user表中的前10条记录。

图 5-19 测试函数



5.3.3 函数访问 RDS for MySQL 示例代码

函数访问 RDS for MySQL 示例代码

这段示例代码实现了**从RDS for MySQL实例数据库的user表中查询前10条记录的功能**。通过使用数据库连接池和重试机制，代码能够高效且可靠地执行数据库操作。

以下为完整的函数示例代码。其中关于连接池和重试部分的代码解读请参考**5.3.4 示例代码解读**。

```
import pymysql
import time
from functools import wraps
from DBUtils.PooledDB import PooledDB

db_client = None
POOL_CONFIG = { # 连接池配置
    'max_connections': 5,          # 最大连接数
    'keepalive_interval': 60,      # 连接保活间隔（秒）
    'max_retries': 3,              # 最大重试次数
```

```
    'retry_delay': 1                # 重试间隔（秒）
}

def initializer(context):
    global db_client
    user = context.getUserData('username')
    password = context.getUserData('password')
    host = context.getUserData('host')
    port = int(context.getUserData('port'))
    database = context.getUserData('database')

    dbConfig = { # mysql数据库配置
        'host': host,
        'port': port,
        'user': user,
        'password': password,
        'database': database,
        'charset': 'utf8',
    }
    db_client = Database(context, POOL_CONFIG, dbConfig)

def handler(event, context):  # 执行入口
    logger = context.getLogger()

    try:
        result= db_client.query("SELECT * FROM user LIMIT 10")
    except Exception as e:
        logger.info("query database error:%s" % e)
        return {"code": 400, "errorMsg": "internal error %s" % e}

    return result

class MySQLConnectionPool:
    def __init__(self, context, pool_config, db_config):
        """
        初始化数据库连接池
        :param db_config: 数据库配置
        :param pool_config: 连接池配置
        """
        self.context = context
        self.logger = context.getLogger();

        self.db_config = db_config
        self.pool_config = pool_config
        self.pool = self._create_pool()
        self.last_keepalive_time = 0

    def _create_pool(self):
        """
        创建数据库连接池
        :return: 连接池对象
        """
        try:
            pool = PooledDB(
                creator=pymysql,
                maxconnections=self.pool_config['max_connections'],
                mincached=1,
                **self.db_config
            )
            return pool
        except Exception as e:
            self.logger.error(f"Failed to create connection pool: {e}")
            raise

    def _get_connection(self):
        """
        从连接池获取连接，并确保连接有效
        :return: 数据库连接对象
        """
```



```
        conn = self.pool.connection()
        if not self._is_connection_alive(conn):
            conn = self.pool.connection()
        return conn

    def _is_connection_alive(self, conn):
        """
        检查连接是否存活
        :param conn: 数据库连接对象
        :return: bool
        """
        try:
            with conn.cursor() as cursor:
                cursor.execute("SELECT 1")
                return True
        except Exception as e:
            self.logger.warning(f"Connection is not alive: {e}")
            return False

    def _close_connection(self, conn):
        """
        关闭连接
        :param conn: 数据库连接对象
        """
        try:
            conn.close()
            self.logger.info("Connection closed")
        except Exception as e:
            self.logger.error(f"Failed to close connection: {e}")

    def _execute_query(self, conn, sql, params=None):
        """
        执行数据库查询
        :param conn: 数据库连接对象
        :param sql: SQL 语句
        :param params: SQL 参数
        :return: 查询结果
        """
        try:
            with conn.cursor() as cursor:
                cursor.execute(sql, params)
                if sql.strip().lower().startswith('select'):
                    return cursor.fetchall()
                return None
        except Exception as e:
            self.logger.error(f"Query failed: {e}")
            raise

    def _execute_write(self, conn, sql, params=None):
        """
        执行写操作（插入、更新、删除）
        :param conn: 数据库连接对象
        :param sql: SQL 语句
        :param params: SQL 参数
        :return: 受影响的行数
        """
        try:
            with conn.cursor() as cursor:
                cursor.execute(sql, params)
                conn.commit()
                return cursor.rowcount
        except Exception as e:
            self.logger.error(f"Write operation failed: {e}")
            conn.rollback()
            raise

def retry(max_retries=3, retry_delay=1):
    """
    重试装饰器
```

```
    :param max_retries: 最大重试次数
    :param retry_delay: 重试间隔（秒）
    """
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            retries = 0
            while retries < max_retries:
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    print(f"Attempt {retries + 1} failed: {e}")
                    if retries < max_retries - 1:
                        time.sleep(retry_delay)
                    retries += 1
            print(f"Failed after {max_retries} attempts")
            raise
        return wrapper
    return decorator

class Database:
    def __init__(self, context, pool_config, db_config):
        self.pool_config = pool_config
        self.db_config = db_config
        self.pool = MySQLConnectionPool(context, pool_config, db_config)

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def query(self, sql, params=None):
        """
        执行查询操作
        :param sql: SQL 语句
        :param params: SQL 参数
        :return: 查询结果
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_query(conn, sql, params)
        return result

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def execute(self, sql, params=None):
        """
        执行写操作（插入、更新、删除）
        :param sql: SQL 语句
        :param params: SQL 参数
        :return: 受影响的行数
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_write(conn, sql, params)
        return result
```

5.3.4 示例代码解读

连接池

在示例代码中基于DBUtils.PooledDB构建MySQL连接池（MySQLConnectionPool），并提供内置的连接探活机制，配置了最大连接数（max_connections）和连接探活间隔（keepalive_interval），代码片段如下：

```
POOL_CONFIG = { # 连接池配置
    'max_connections': 5,           # 最大连接数
    'keepalive_interval': 60,       # 连接保活间隔（秒）
    'max_retries': 3,               # 最大重试次数
    'retry_delay': 1                # 重试间隔（秒）
}

class MySQLConnectionPool:
    def __init__(self, context, pool_config, db_config):
```

```
    """
    初始化数据库连接池
    :param db_config: 数据库配置
    :param pool_config: 连接池配置
    """
    self.context = context
    self.logger = context.getLogger();

    self.db_config = db_config
    self.pool_config = pool_config
    self.pool = self._create_pool()
    self.last_keepalive_time = 0

def _create_pool(self):
    """
    创建数据库连接池
    :return: 连接池对象
    """
    try:
        pool = PooledDB(
            creator=pymysql,
            maxconnections=self.pool_config['max_connections'],
            mincached=1,
            **self.db_config
        )
        return pool
    except Exception as e:
        self.logger.error(f"Failed to create connection pool: {e}")
        raise

def _get_connection(self):
    """
    从连接池获取连接，并确保连接有效
    :return: 数据库连接对象
    """
    conn = self.pool.connection()
    if not self._is_connection_alive(conn):
        conn = self.pool.connection()
    return conn

def _is_connection_alive(self, conn):
    """
    检查连接是否存活
    :param conn: 数据库连接对象
    :return: bool
    """
    try:
        with conn.cursor() as cursor:
            cursor.execute("SELECT 1")
            return True
    except Exception as e:
        self.logger.warning(f"Connection is not alive: {e}")
        return False

def _close_connection(self, conn):
    """
    关闭连接
    :param conn: 数据库连接对象
    """
    try:
        conn.close()
        self.logger.info("Connection closed")
    except Exception as e:
        self.logger.error(f"Failed to close connection: {e}")

def _execute_query(self, conn, sql, params=None):
    """
    执行数据库查询
    :param conn: 数据库连接对象
```



```
    :param sql: SQL 语句
    :param params: SQL 参数
    :return: 查询结果
    """
    try:
        with conn.cursor() as cursor:
            cursor.execute(sql, params)
            if sql.strip().lower().startswith('select'):
                return cursor.fetchall()
            return None
    except Exception as e:
        self.logger.error(f"Query failed: {e}")
        raise

def _execute_write(self, conn, sql, params=None):
    """
    执行写操作（插入、更新、删除）
    :param conn: 数据库连接对象
    :param sql: SQL 语句
    :param params: SQL 参数
    :return: 受影响的行数
    """
    try:
        with conn.cursor() as cursor:
            cursor.execute(sql, params)
            conn.commit()
            return cursor.rowcount
    except Exception as e:
        self.logger.error(f"Write operation failed: {e}")
        conn.rollback()
        raise
```

使用MySQL连接池复用已创建的连接，能有效提升程序性能。最大连接数配置确保连接资源的使用保持在可控范围内，同时确保线程安全。

相关概念说明：

表 5-8 连接相关概念说明

概念	说明
最大连接数配置区间	在FunctionGraph函数配置Mysql最大连接数建议在如下区间选取一个值： ● **最大连接数下限** =（**函数单实例并发度**）*（函数单次执行访问MySQL并发度） ● **最大连接数上限** =（MySQL实例连接数上限）/（函数最大实例数） 例如：某个访问MySQL函数单实例并发度配置为5，每次执行函数访问MySQL并发度为2，函数最大实例数默认400，访问的MySQL实例连接数上限为30000，则计算如下： ● 最大连接数下限 = 5*2 = 10 ● 最大连接数上限 = 30000/400 = 75 按上述结果，建议将最大连接数配置为50。
连接探活间隔	不要超过**函数执行超时时间**，避免因连接断开导致的问题。

重试

通过装饰器构建自动重试机制，确保在执行MySQL操作失败后重试特定次数，这样可以显著降低暂时性故障的影响。例如，在瞬时网络抖动、磁盘问题导致服务暂时不可用或调用超时的情况下，提高MySQL操作的成功率。

代码片段如下：

```
POOL_CONFIG = { # 连接池配置
        'max_connections': 5,           # 最大连接数
        'keepalive_interval': 60,       # 连接保活间隔（秒）
        'max_retries': 3,               # 最大重试次数
        'retry_delay': 1                # 重试间隔（秒）
}

class Database:
    def __init__(self, context, pool_config, db_config):
        self.pool_config = pool_config
        self.db_config = db_config
        self.pool = MySQLConnectionPool(context, pool_config, db_config)

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def query(self, sql, params=None):
        """
        执行查询操作
        :param sql: SQL 语句
        :param params: SQL 参数
        :return: 查询结果
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_query(conn, sql, params)
        return result

    @retry(max_retries=POOL_CONFIG['max_retries'], retry_delay=POOL_CONFIG['retry_delay'])
    def execute(self, sql, params=None):
        """
        执行写操作（插入、更新、删除）
        :param sql: SQL 语句
        :param params: SQL 参数
        :return: 受影响的行数
        """
        conn = self.pool._get_connection()
        result = self.pool._execute_write(conn, sql, params)
        return result

def retry(max_retries=3, retry_delay=1):
    """
    重试装饰器
    :param max_retries: 最大重试次数
    :param retry_delay: 重试间隔（秒）
    """
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            retries = 0
            while retries < max_retries:
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    print(f"Attempt {retries + 1} failed: {e}")
                    if retries < max_retries - 1:
                        time.sleep(retry_delay)
                    retries += 1
            print(f"Failed after {max_retries} attempts")
            raise
        return wrapper
    return decorator
```

