微服务平台 TSF

实践教程

腾讯云

文档目录

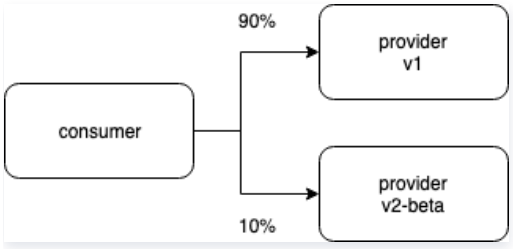
实践教程
灰度发布

最近更新时间：2022-06-10 11:31:37

当用户需要上线新的功能时，希望使用灰度发布的手段在小范围内进行新版本发布测试。
TSF 支持通过部署组和服务路由来实现灰度发布。

场景说明

consumer 调用 provider 时，provider 有两个版本 v1 和 v2-beta，其中 v2-beta 是测试版本。consumer 首先将90%的请求分配给 provider v1 版本，剩下的10%分配给 v2-beta 版本。如果发现 v2-beta 版本运行正常，则增加该版本的流量比例。



前提条件

- 已经下载基于 TSF Spring Cloud SDK 或者 Mesh 编写的代码程序包。
- 已经创建了集群和命名空间，集群中导入2个云服务器。
- 已经创建了 consumer 和 provider 应用（虚拟机部署方式），同时已经创建并部署了 consumer 部署组。

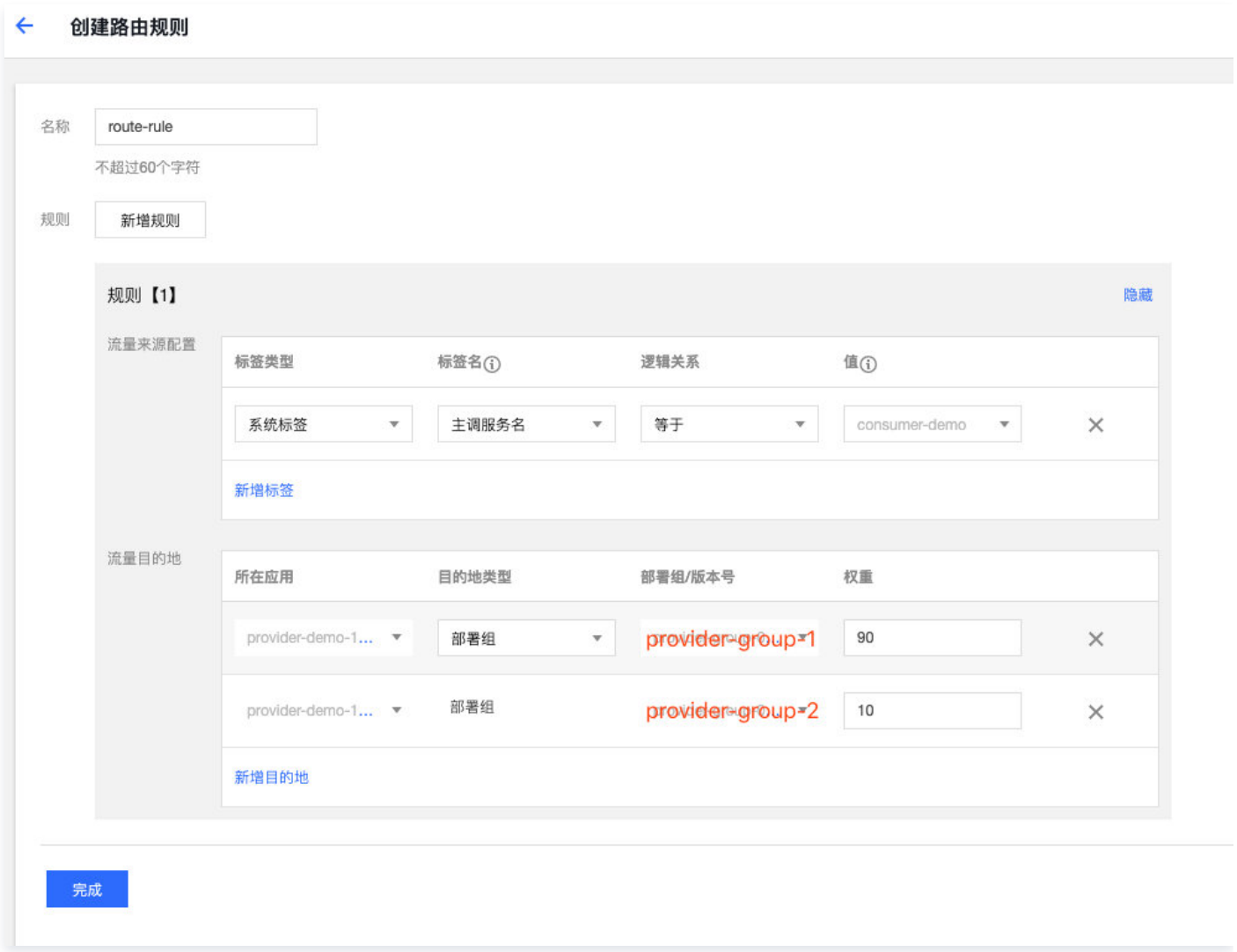
操作步骤

步骤1：provider 创建2个部署组

1. 登录 TSF 控制台 。
2. 在左侧导航栏中，单击 **应用管理**，进入应用列表页，选择 provider 应用，进入应用详情页。
3. 单击顶部**程序包管理**，上传 v1 和 v2-beta 的程序包。
4. 单击顶部**部署组**，在部署组列表页面创建两个部署组：
   - 部署组 provider-group-1，添加1个实例，实例规格为1核1GB。
   - 部署组 provider-group-2，添加1个实例，实例规格为1核1GB。
5. provider-group-1 部署程序包 v1。
6. provider-group-2 部署程序包 v2-beta。

步骤2：配置服务路由规则及初始路由权重 90:10

1. 在左侧导航栏中，单击 ** 服务治理 **，进入服务列表页，单击 provider 服务，进入服务详情页。
2. 单击顶部**服务路由**， 新建路由规则，流量来源为 主调服务名等于`consumer-demo` ，流量目的地如下图所示，设置 provider-group-1的权重为90，provider-group-2 的权重为10。
3. 在服务路由规则列表中，单击生效状态列的图标启动该规则，稍等几分钟，刷新页面观察列表下方流量比例变化情况，如果发现 provider-group-1 和 provider-group-2 的流量比例接近 90:10，说明路由规则生效。



步骤3：修改路由权重为 50:50

当 v2-beta 版本服务已经正常运行一段时间后，逐步增加 v2-beta 版本的流量比例，并减少 v1 版本的流量比例。

1. 在服务路由规则列表中，单击**编辑**。
2. 修改流量目的地中 provider-group-1 和 provider-group-2 的流量比例分别为50和50。
3. 稍等几分钟，刷新页面观察列表下方流量比例变化情况，如果 provider-group-1 和 provider-group-2 的流量比例接近 50:50，说明路由规则生效。

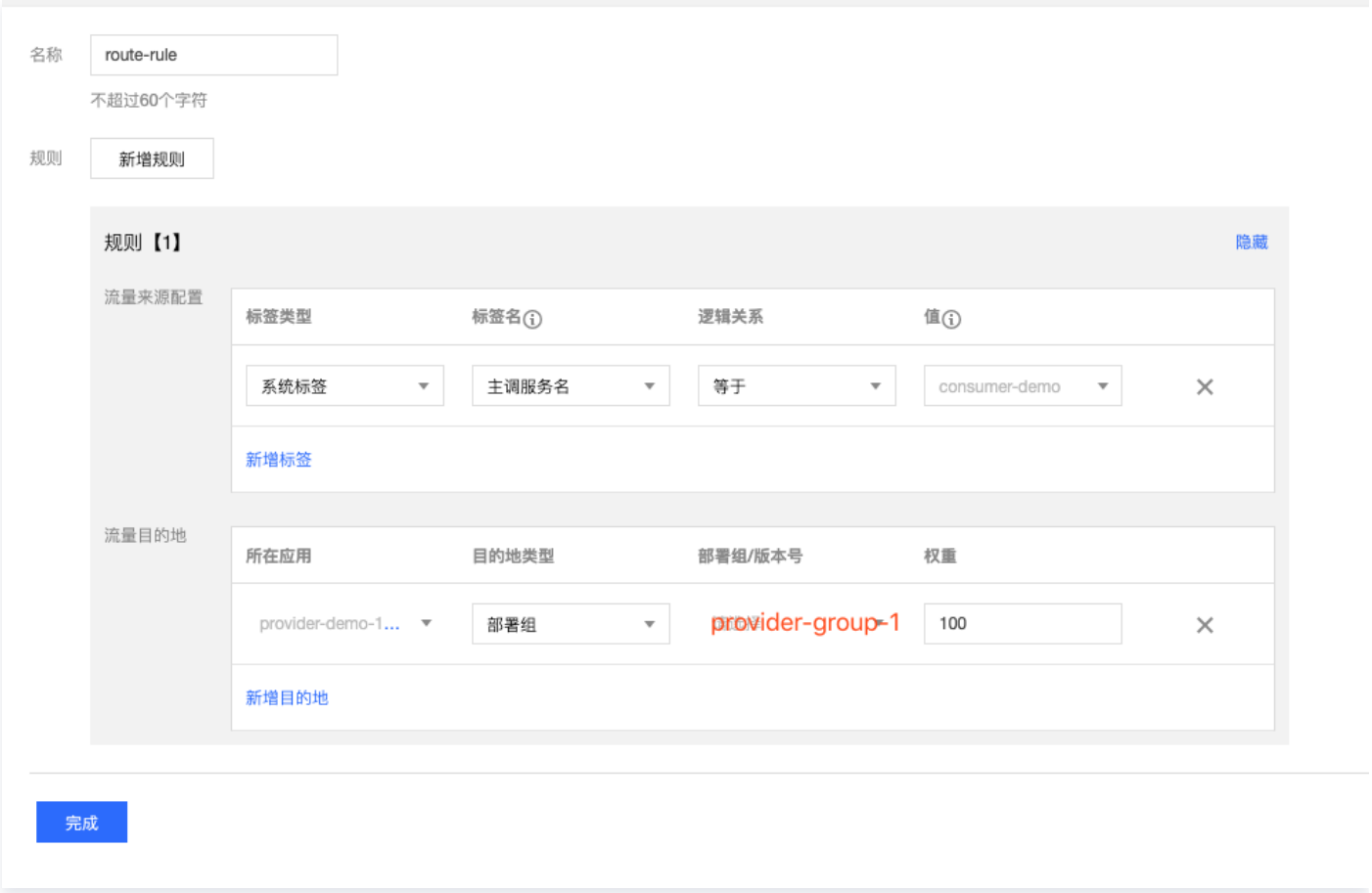
步骤4：修改路由权重为 10:90

逐步增加 v2-beta 版本的流量比例，并减少 v1 版本的流量比例。操作方法参考 步骤:3 。

步骤5：修改路由权重为 0:100

1. 修改路由规则，将 provider-group-1 的权重设置为100。
2. 稍等几分钟，刷新页面观察列表下方流量比例变化情况，如果只有 provider-group-1 有流量，说明路由规则生效。

3. 此时可以停止部署组 provider-group-2 。



使用说明

在真实使用场景中，有以下几点需要考虑：

- 评估单个 provider 服务实例可以处理多少请求，并根据流量分配来规划部署组的实例数量。
- 如果流量比例变化后，可能需要调整部署组的实例数量来满足新的流量比例。
- 可以通过设置**弹性伸缩规则**来支持动态调整实例数量。

就近路由和跨可用区容灾

最近更新时间：2024-06-14 18:16:21

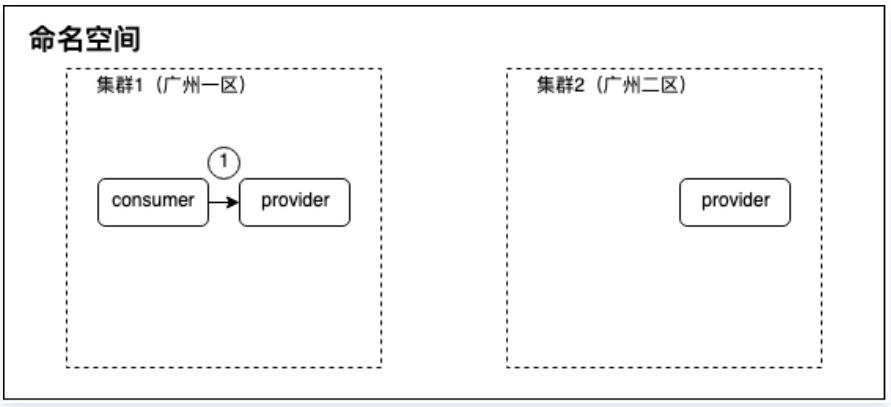
TSF 支持业务同城可用区就近路由和跨可用区容灾。该功能在1.12.0版本（参考 版本更新 ）发布之后生效，使用前提条件：

- 如果是 Spring Cloud 应用，必须使用1.12.0之后版本的 SDK。
- 如果是虚拟机应用，部署实例必须是1.12.0发布之后导入集群的云主机。
- 如果是容器应用，部署实例必须是1.12.0发布之后创建或者重新部署。

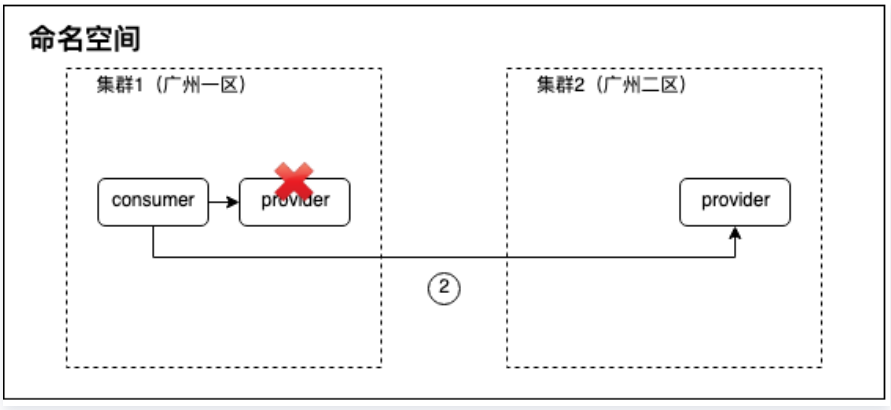
功能概述

命名空间、集群有如下特点：不同集群可关联同一命名空间，服务支持跨集群访问，详细说明请参考 命名空间 。
以 consumer 服务调用 provider 服务为例说明服务跨可用区访问的原理。在下方示意图中有两个集群，分别位于广州一区和广州二区。集群1分别部署了 consumer 和 provider 微服务，集群2部署了 provider 微服务。两个可用区的服务都会注册到同一个注册中心集群。

- 当**开启就近路由**时，广州一区的 consumer 会优先调用同一可用区的 provider。



- 当**开启就近路由**时，当广州一区的 provider 不可用时，consumer 会跨可用区调用广州二区的 provider。



- 当**关闭就近路由**时，广州一区的 consumer 会从两个可用区 provider 中随机选择实例进行调用。

就近路由原理

微服务的路由模块会将可用区作为一种系统标签，当**开启就近路由**时，服务消费者会将请求流量**全部**路由到相同可用区标签的服务提供者，只有当同一可用区服务提供者不可用时，才会进行跨可用区的服务调用；当关闭就近路由时，服务消费者会随机选择不同可用区的服务提供者实例进行调用。
关于服务路由的更多说明参考 服务路由基本原理 和 服务路由使用方法 。

实践教程

跨可用区容灾场景下的就近路由

场景：当**开启就近路由**时，广州一区的 consumer 会优先调用同一可用区的 provider ，只有当广州一区的 provider 不可用时，consumer 才会跨可用区调用广州二区的 provider。

步骤1：准备资源

1. 登录 TSF 控制台 。
2. 在左侧导航栏中，单击**集群**，进入集群列表页，单击**新建集群**。
3. 新建虚拟机集群 cluster1，**所在可用区**选择广州一区，设置其他属性。
4. 新建虚拟机集群 cluster2，**所在可用区**选择广州二区，设置其他属性。
5. 集群 cluster1 和集群 cluster2 分别导入所在可用区的云服务器。
6. 在集群 cluster1 的集群详情页 >**命名空间**标签页，单击**新建**，新建命名空间 dev-ns。
7. 在集群 cluster2 的集群详情页 >**命名空间**标签页，单击**新建**，新建命名空间 dev-ns。
8. 确认命名空间 dev-ns 的就近路由开关默认是开启的。

步骤2：部署应用

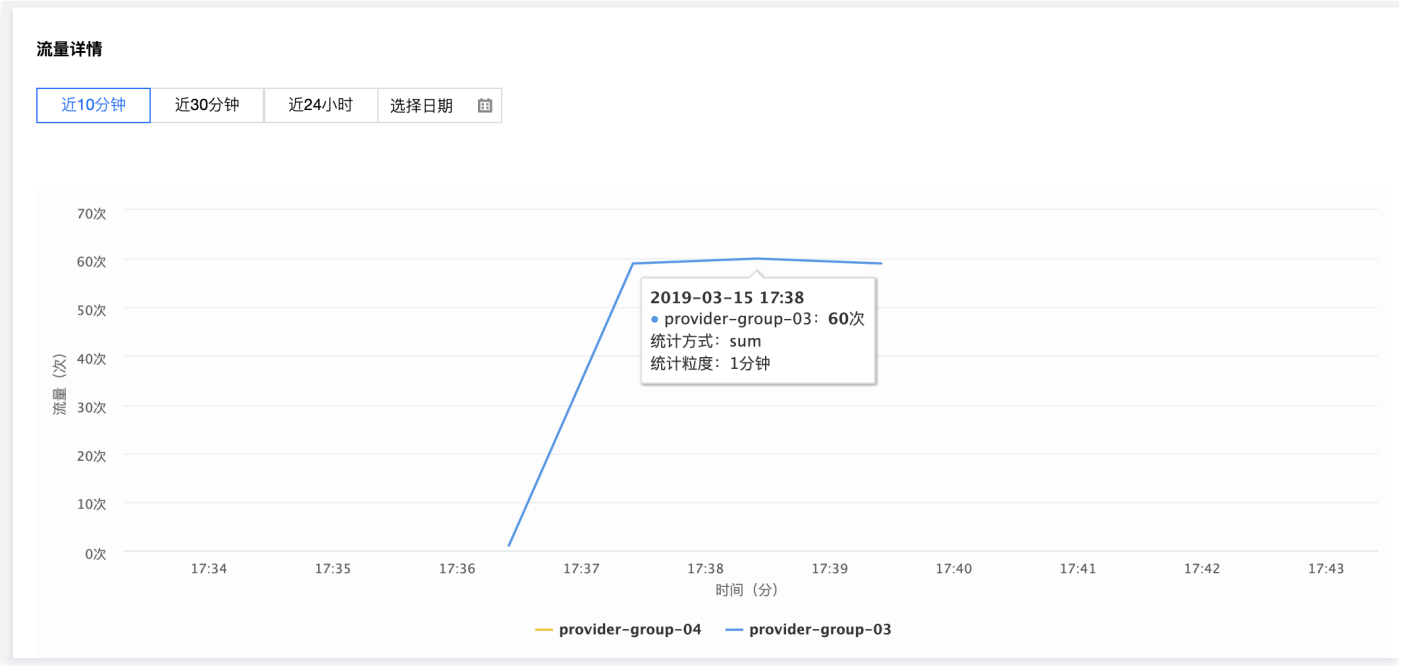
1. 在左侧导航栏中，单击 应用管理 ，**进入应用管理页，单击新建应用**。
2. 新建应用 consumer-app 和 provider-app。
3. 在应用详情页 > **程序包管理**页面上传 Demo 程序包，Demo 获取请参考 Demo 工程概述 。
4. 进入 conumser-app 应用详情页 >**部署组**标签页。
   新建部署组 consumer-group， 属于集群 cluster1，命名空间 dev-ns。添加实例，部署 consumer-demo 程序包。
5. 在 provider-app 应用详情页 > **部署组**标签页
   - 新建部署组 provider-group-03，属于集群 cluster1，命名空间 dev-ns，添加实例，部署 provider-demo 程序包。
   - 新建部署组 provider-group-04，属于集群 cluster2，命名空间 dev-ns，添加实例，部署 provider-demo 程序包。

关于部署的详细操作参考 虚拟机应用部署组 。

步骤3：验证就近路由

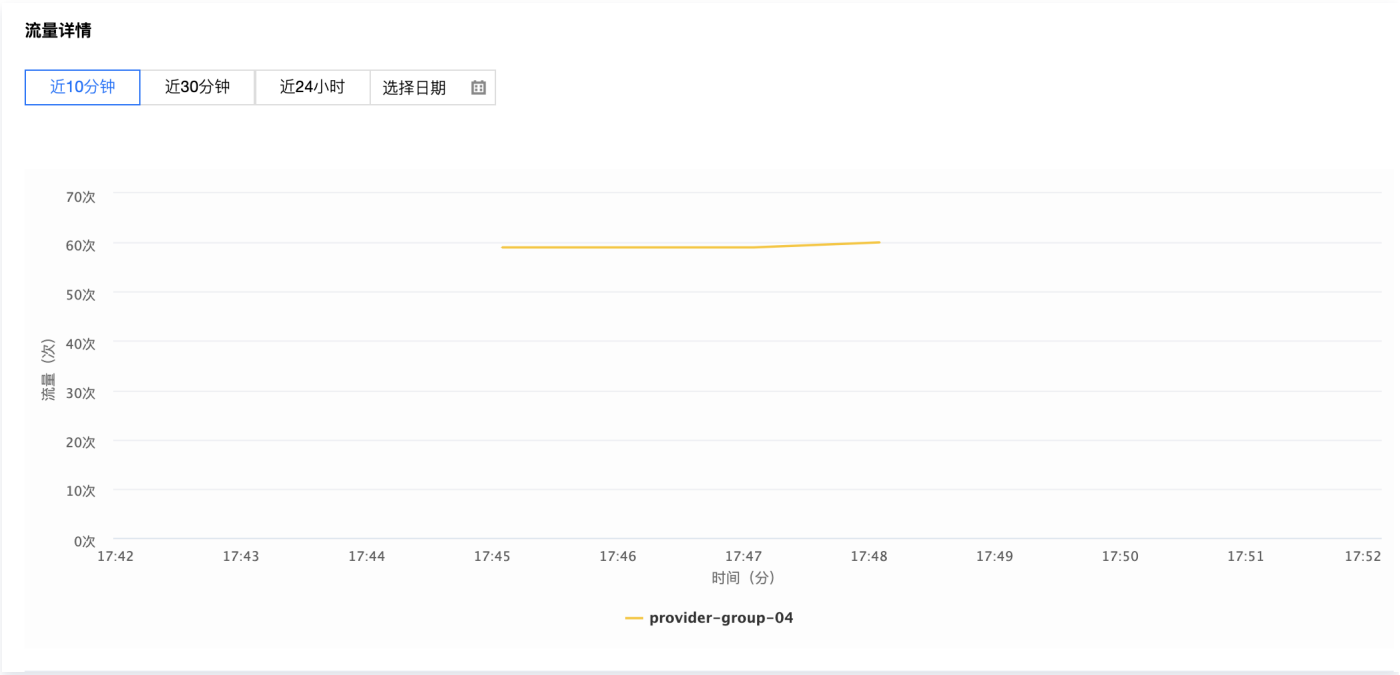
consumer 和 provider 部署成功后，观察服务依赖拓扑图是否出现，如果出现说明服务间调用正常。

1. 在控制台左侧导航栏，单击 ** 服务治理 **。
2. 单击 provider-demo 进入微服务详情页，单击**服务路由**标签页的流量详情中可以观察来自 consumer 的请求是否全部路由到部署组 provider-group-03 中，如果请求全部路由到部署组 provider-group-03 证明就近路由功能生效。



步骤4：验证容灾

1. 停止部署组 provider-group-03。
2. 观察 provider-demo 服务路由页面的流量详情中可以观察到流量分配到部署组 provider-group-04，证明容灾能力生效。



步骤5：关闭就近路由

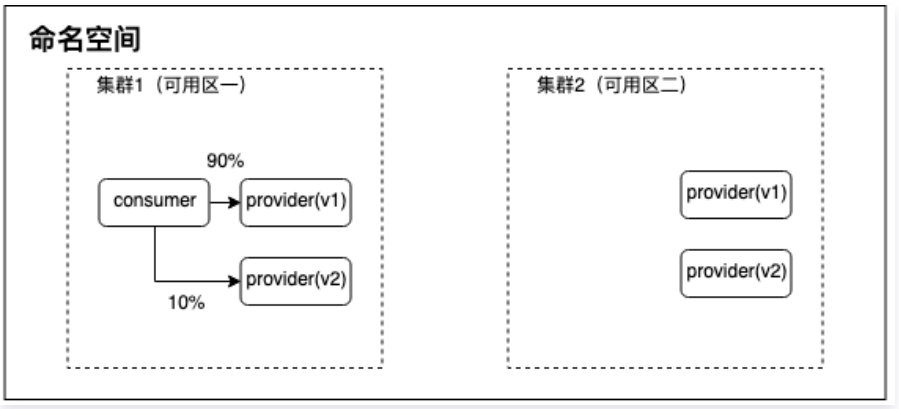
1. 启动部署组 provider-group-03，观察到流量分配回 provider-group-03。
2. 在命名空间页面关闭就近路由开关，观察到流量均匀分配到2个部署组。

就近路由与路由规则

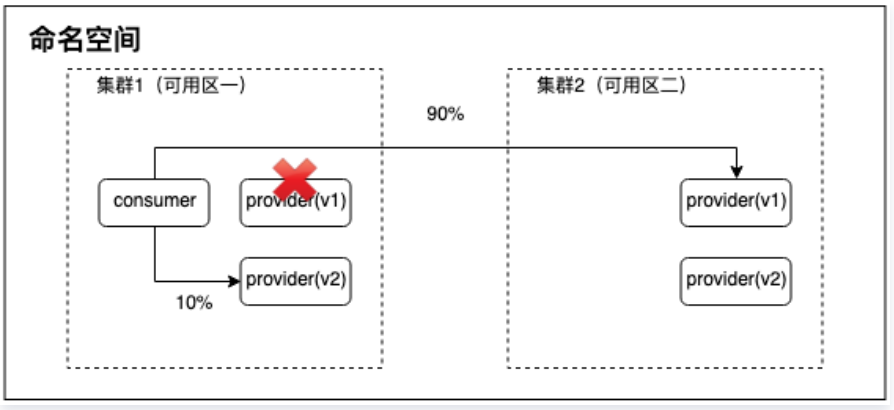
开启就近路由后，服务消费者会按照路由规则优先调用同一可用区的服务提供者，只有当同一可用区的服务提供者不可用时，会按照路由规则进行跨可用区调用。

场景：集群1和集群2都部署了 provider 的两个版本 v1 和 v2；路由规则是90%的流量分配到 v1，10%的流量分配到 v2；命名空间开启了就近路由。

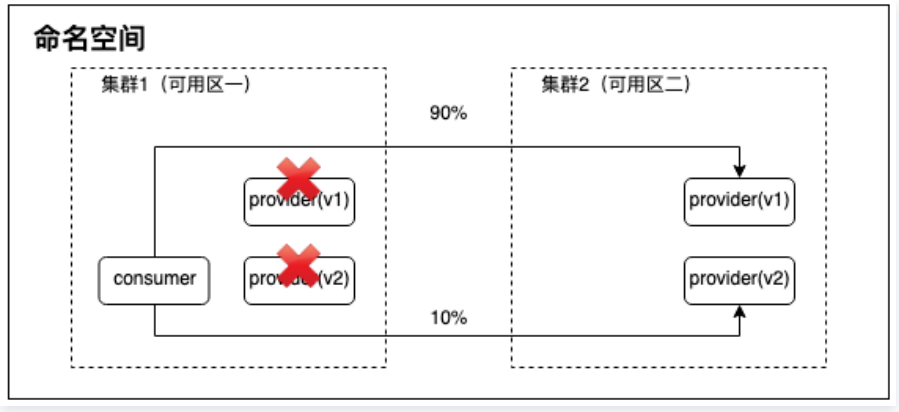
- **当集群1中 provider 的 v1 和 v2 版本实例都正常时，consumer 会按照路由规则调用同一可用区的 provider。**



- **当集群1中 provider 的 v1 实例不正常， v2 版本实例正常时，consumer 会按照路由规则将90%请求发送给可用区二的 v1 ，10%请求发送给可用区一的 v2。**



- **当集群1中 provider 的 v1 和 v2 实例都不正常，consumer 会按照路由规则将所有90%请求发送给可用区二的 v1，10%请求发送给可用区二的 v2。**

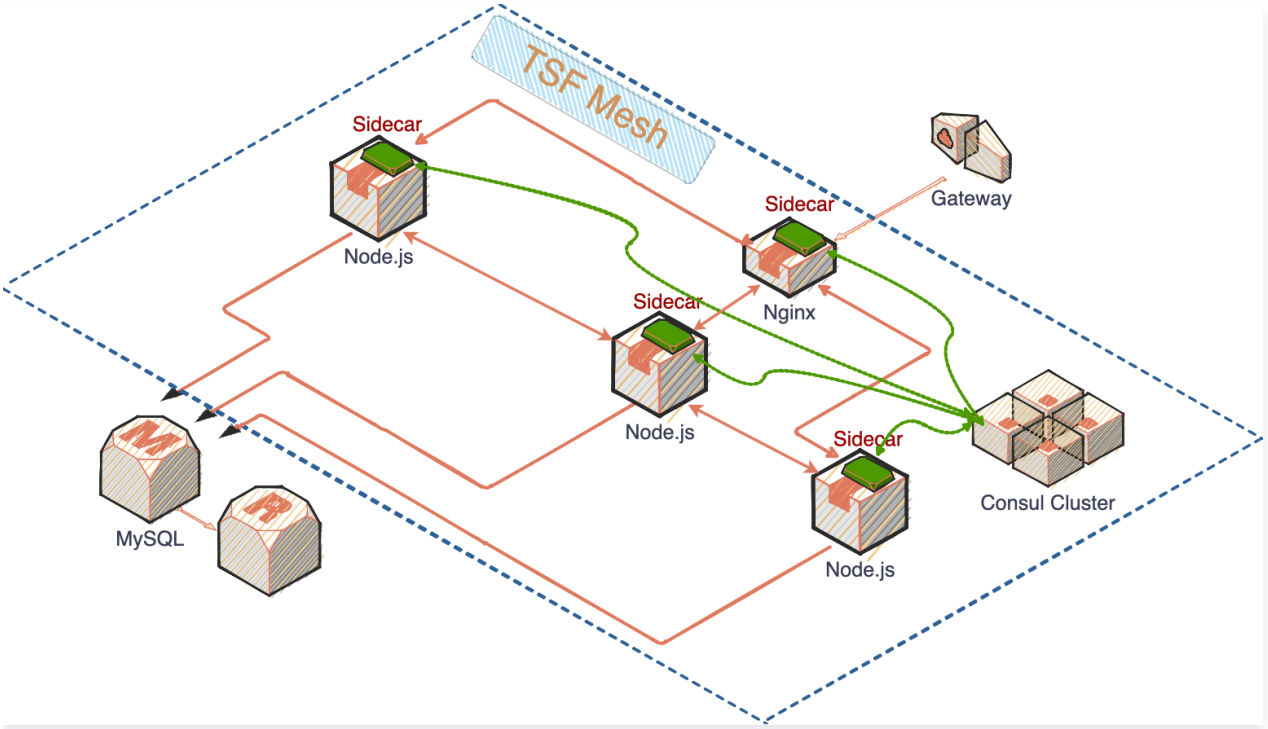


基于 TSF Mesh 的前端静态资源托管

最近更新时间：2024-11-05 16:13:52

操作场景

该任务指导用户依托 TSF Mesh 技术以服务形式托管 Node.js 和 Nginx，最终验证 Node.js 与 Nginx 可以以服务形式注册、发现及成功调用。概要架构图如下：



- Nginx：前端资源托管。
- Node.js：后端服务。
- Sidecar：和服务运行在同一个 Pod 中，与 Pod 共享网络，服务感受不到 Sidecar 的存在。
  - Sidecar 代理服务向注册中心注册服务相关信息，以便其他服务发现自身。
  - Sidecar 作为 Pod 内服务的 HTTP 代理，可以自动发现其他服务。
- Consul Cluster：管理和配置 Sidecar 来执行策略并收集遥测。

操作步骤

步骤1：部署 Node.js 服务

1. 准备 Node.js 应用代码。
   可参考 Demo 中配置文件进行构建，其它语言的 TSF Mesh 的 Demo 示例可参考 TSF Mesh Demo 。
2. 制作 Node.js + express 应用镜像，详细操作请参考 制作容器镜像 。
3. 上传到腾讯云镜像仓库，详细操作请参考 镜像仓库 。
4. 部署 Node.js 应用：
   4.1 创建应用 。应用类型选择 Mesh 应用。
   4.2 在左侧导航栏选择**部署组** > **新建部署组**，创建 Node.js 的部署组。
   4.3 在部署组页面，选择部署组右侧操作列的**部署应用**，选择刚刚创建的镜像，设置部署相关信息。

步骤2：托管前端静态资源

1. 手动在云服务器上安装 Nginx，详细操作请参考 Nginx 官网的 安装说明 。

2. 准备程序包。程序包的目录为：
   - `www` 目录：该目录名可任意，用于存放 Web 静态资源文件。
   - `start.sh` 脚本：将 www 目录下文件移动到 Nginx 站点目录下，同时启动 Nginx 服务。
   - `stop.sh` 脚本：停止 Nginx 服务。
   - `cmdline` 文件：检查 Nginx 进程是否存在的 `grep` 命令关键字。

   用户可参考 nginx_demo 了解各文件的具体作用和写法。
3. 根据 Dockerfile 生成本地镜像并上传到腾讯云镜像仓库，可参考 制作容器镜像 。
4. 部署 Nginx 应用：

   4.1 创建应用 。应用类型选择 Mesh 应用。

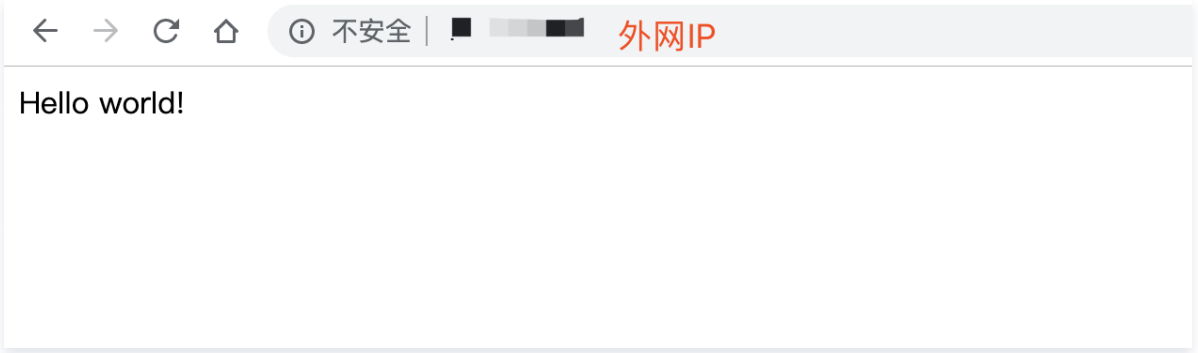
   4.2 在左侧导航栏选择**部署组** > **新建部署组**，创建 Node.js 的部署组。

   4.3 在部署组页面，选择部署组右侧操作列的**部署应用**，选择刚刚创建的镜像，设置部署相关信息。

步骤3：测试 Node.js 服务和 Nginx 服务之间的调用

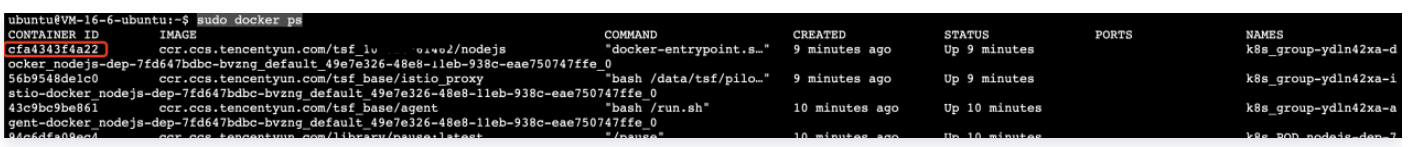
1. 通过外网 IP 访问测试 Node.js 应用。

2. 通过外网 IP 访问测试前端静态资源。



3. 在 Node.js 服务容器内 curl 访问 Nginx 服务。

```
sudo docker ps    #查找容器id
```



```
sudo docker exec -it cfa4343f4a22 /bin/bash  #进入容器内部
```

```
ubuntu@VM-16-6-ubuntu:~$ sudo docker exec -it cfa4343f4a22 /bin/bash
root@nodejs-dep-7fd647bdbc-bvzng:/usr/src/app#
```

在 Node.js 服务容器内 curl 访问 Nginx 服务成功。

```
root@nodejs-dep-58fcb66d5f-tggn7:/usr/src/app# curl http://nginx-service/www/hello.html
<html>
<header><title>This is title</title></header>
<body>
Hello world!
</body>
</html>
root@nodejs-dep-58fcb66d5f-tggn7:/usr/src/app#
```

微服务网关作为请求入口

最近更新时间：2024-08-19 17:56:21

操作场景

腾讯云 TSF 提供了微服务网关 SDK，并封装了限流、密钥对鉴权、第三方鉴权、设置访问标签等功能，方便您在界面上管理微服务 API。您需要自行准备计算资源（虚拟机或容器），将微服务网关部署、并导入微服务 API。

操作步骤

步骤1：部署微服务网关

1. 创建微服务网关应用
   登录 TSF 控制台，在左侧导航栏单击**应用管理**，并新建应用，业务类型选择微服务网关应用。

2. 部署微服务网关应用
   2.1 在微服务网关应用详情页，单击顶部**程序包管理** > **上传程序包**。
   我们使用不包含任何自定义过滤器的 微服务网关 Demo ，将文件中 msgw - demo 进行打包和上传。
   2.2 单击顶部**部署组** > **新建部署组**，将网关部署在希望对外暴露微服务 API 的微服务所在的命名空间中。部署组相关操作请参考 部署组管理 文档。

有关全局命名空间、非全局命名空间对微服务网关的影响，请参见 部署微服务网关 。

> **说明：**
> 提供的 Demo 端口默认为8080，可通过启动参数进行调整。

步骤2：通过微服务网关托管微服务 API

1. 在 TSF 控制台 的左侧导航栏中，单击组建中心下的**微服务网关 > 网关管理**。
2. 在**网关管理**页，点击目标网关的ID，进入到目标网关，单击**新建分组**，并填写分组信息。
   - 分组名称：最长为60个字符，只能包含小写字母、数字及分隔符（"_"、"-"），且不能以分隔符开头或结尾。
   - 访问 context：context 是访问网关管理的某一个 API 的路径的路径参数。以“/”开头，不能为空。在这个例子中，添加为 `/myfirsttest` 。

- 鉴权类型：密钥对鉴权或免鉴权。当选择密钥对鉴权时，请求参数中不带正确的 SecretId 和签名的访问会被拒绝。

3. 单击**保存&下一步**，进行网关部署组绑定。
4. 将分组与步骤1中创建的微服务网关应用的部署组进行绑定。

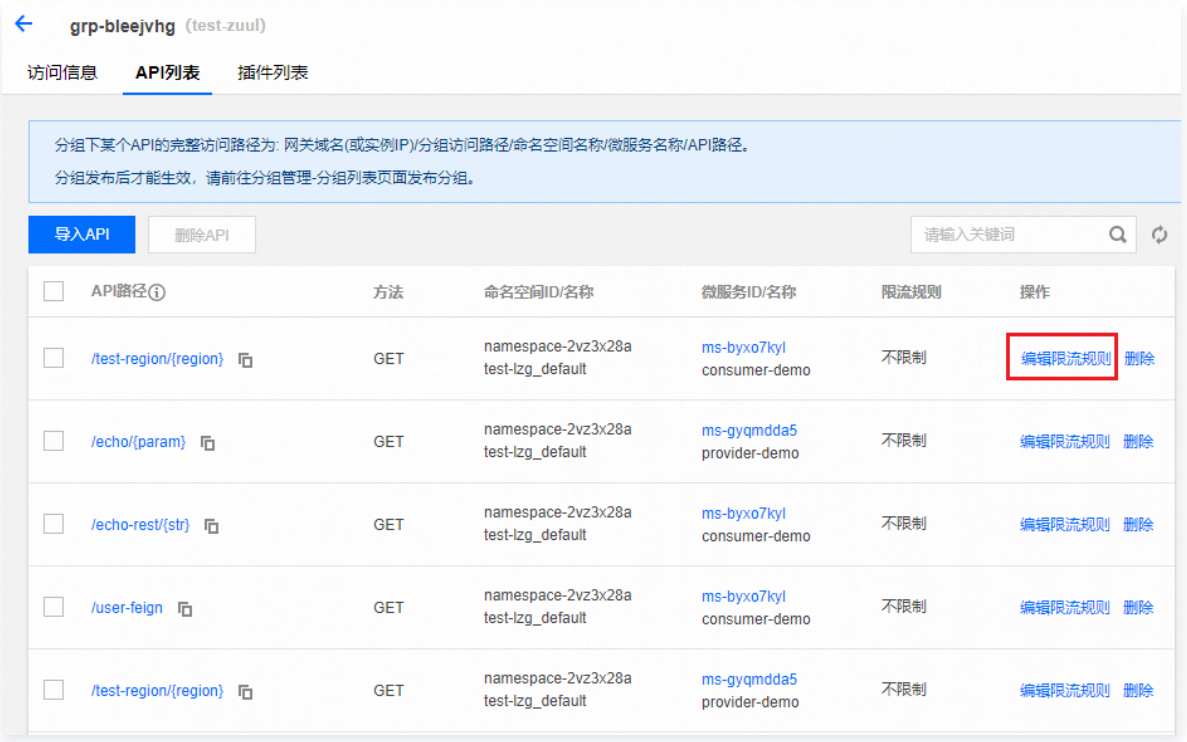
> **说明**
> 此处绑定的是创建的微服务网关应用的部署组，而不是需要对外暴露接口的微服务的部署组。

5. 在**API 列表**中，单击**导入 API**，选择对外暴露的微服务下的 API，完成后单击**提交**。

6. 返回分布管理列表，单击操作列的**发布分组**。发布之后，即可通过访问微服务网关的部署组 + 分组下的 API 访问路径进行访问。

步骤3：设置 API 限流策略

1. 在 TSF 控制台 的左侧导航栏中，单击组建中心下的**微服务网关** > 网关**管理**，
2. 在**网关管理**页，点击目标网关的ID，进入到目标网关，单击分组管理，查看分组详情中 API 列表页面。
3. 为已经导入的 API 设置限流。



说明与注意

访问路径相关说明：

微服务网关访问路径为： 微服务网关域名或 `IP` + `port` / 分组 `contxt` / 微服务所在命名空间 / 微服务名称/`API path`

示例：创建一个应用类型为微服务网关的应用，部署组中节点 IP 为 10.10.10.10，端口为8080，需要访问命名空间为 default 的微服务 provider，API 路径为 `/echo`，为此创建了一个访问路径为 test 的分组。此时应用的访问路径为 `10.10.10.10:8080/test/default/provider/echo` 。

其中，微服务网关的默认端口为8080，您可以通过修改启动参数对网关默认端口进行修改。

为微服务网关所在的部署组配置对外 VIP 的说明：

- 当微服务网关部署在虚拟机上时，您可以将微服务网关所在的部署组的机器挂载在腾讯云负载均衡实例后面。详情请参见 负载均衡快速入门 。
- 当微服务网关部署在容器上时，且需要将微服务 API 暴露在公网访问，则在新建容器部署组时，选择公网访问方式。详情请参见 容器应用部署组 > 网络访问方式 。

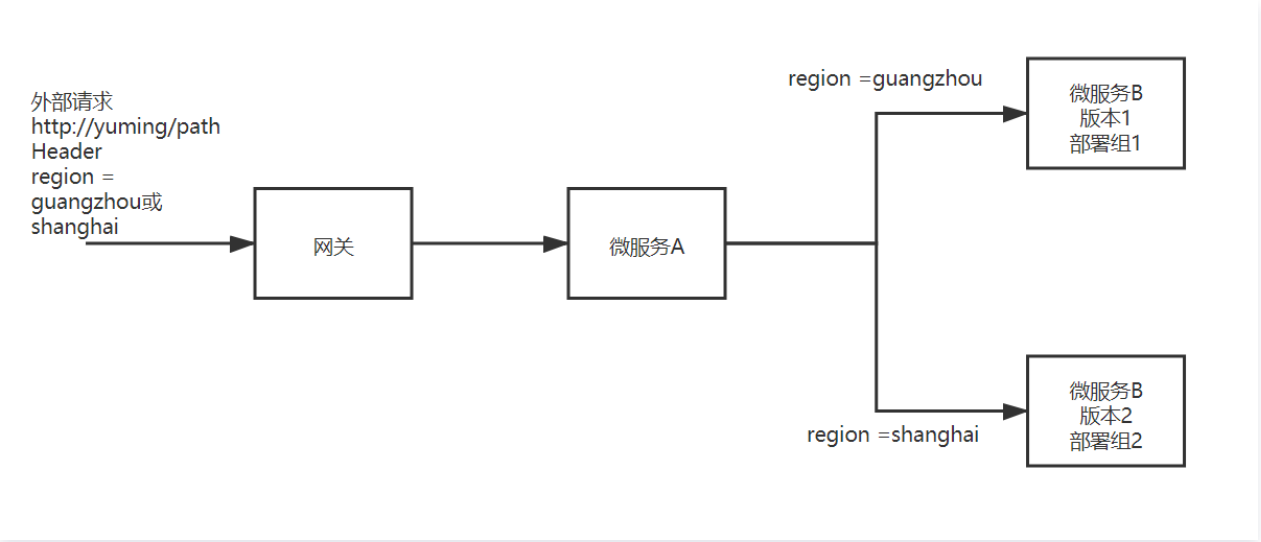
基于业务参数的服务治理

最近更新时间：2024-08-19 18:03:41

操作场景

本文档指导您通过微服务网关和服务路由能力，即可实现基于业务参数的服务治理。您无需修改代码，即可实现基于业务参数的路由、鉴权功能，并依据业务参数过滤请求调用链。
在实际的业务场景中，经常有依据某些请求参数值决定请求路由到某个服务的某个版本中的场景，或依据请求参数值对请求进行限流、鉴权等场景。
如下图中，当外网请求通过网关访问后端微服务时：

- 当请求参数 region = guangzhou 时，路由到微服务的版本1中。
- 当请求参数 region = shanghai 时，路由到微服务的版本2中。



前提条件

在开始本文的实践前，您需要先了解 TSF 的以下功能：

- 微服务网关的部署 、通过微服务网关管理 API
- 微服务网关插件（tag 类型插件）
- 服务路由
- 标签

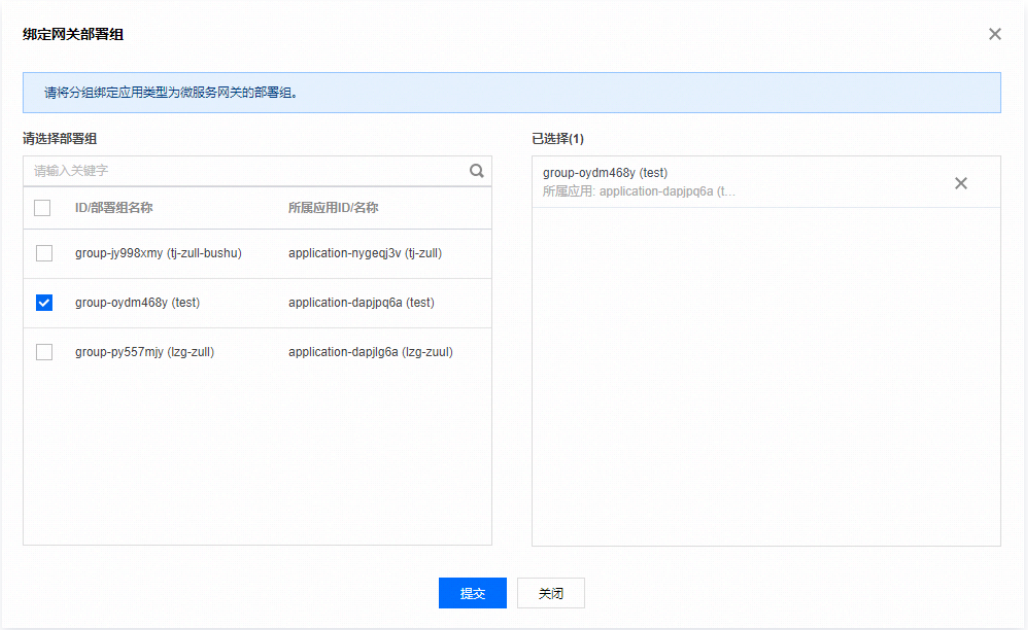
操作步骤

步骤1：新建微服务网关分组

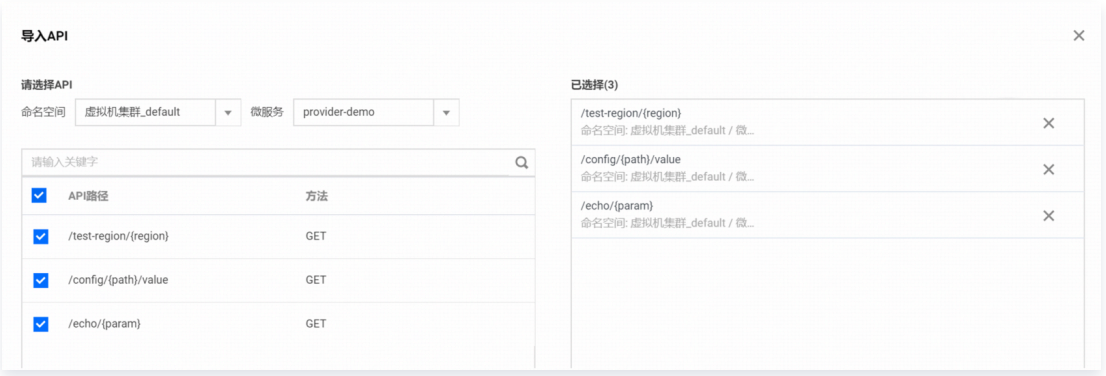
1. 准备 consumer -demo 和 provider - demo 两个微服务。要求如下：
   - provider - demo 存在两个版本的包：provider-demo-guangzhou.jar 和 provider-demo-shanghai.jar。
   - provider - demo 两个版本在访问请求中携带 region 参数，且访问 /test-region 接口时，会分别返回 shanghai 和 guangzhou；访问 provider - demo 时，可以携带两个 Header 参数、region 和 usertype。
   - consumer - demo 部署在一个部署组上，provider - demo 部署在两个部署组上，每个部署组部署一个版本的 jar 包。创建部署组的相关操作请参见 部署组管理 文档。
2. 部署一个微服务网关（使用 官网 demo ），部署后服务名为 msgw - demo，部署组名称为 test，默认端口为8080（给对应机器开放8080端口）。在本 demo 中，将直接通过网关访问微服务 provider。
   具体操作请参考 部署微服务网关 。
3. 新建微服务网关分组，并将微服务网关分组绑定在创建好的网关应用部署组上。具体操作请参见 微服务网关分组管理 。


新建分组
1 填写基本信息
2 绑定网关部署组
分组名称
请输入分组名称
最长为60个字符，只能包含小写字母、数字及分隔符("_"、"-")，且仅能以小写字母开头，不能以分隔符结尾
访问context
请输入访问context
请填写path参数，以"/"开头，不能为空，不支持多个"/"符号。
托管API类型
微服务API（推荐）
外部API
鉴权类型
密钥鉴权
无鉴权
访问参数配置
命名空间
path
微服务名称
path
备注
选填，200字符内
保存 & 下一步
关闭

4. 绑定网关部署组：


绑定网关部署组
请将分组绑定应用类型为微服务网关的部署组。
请选择部署组
请输入关键字
ID/部署组名称
所属应用ID/名称
group-jy998xmy (tj-zull-bushu)
application-nygeqj3v (tj-zull)
group-oydm468y (test)
application-dapjpq6a (test)
group-py557mjy (lzg-zull)
application-dapjlg6a (lzg-zuul)
已选择(1)
group-oydm468y (test)
所属应用: application-dapjpq6a (t...
提交
关闭

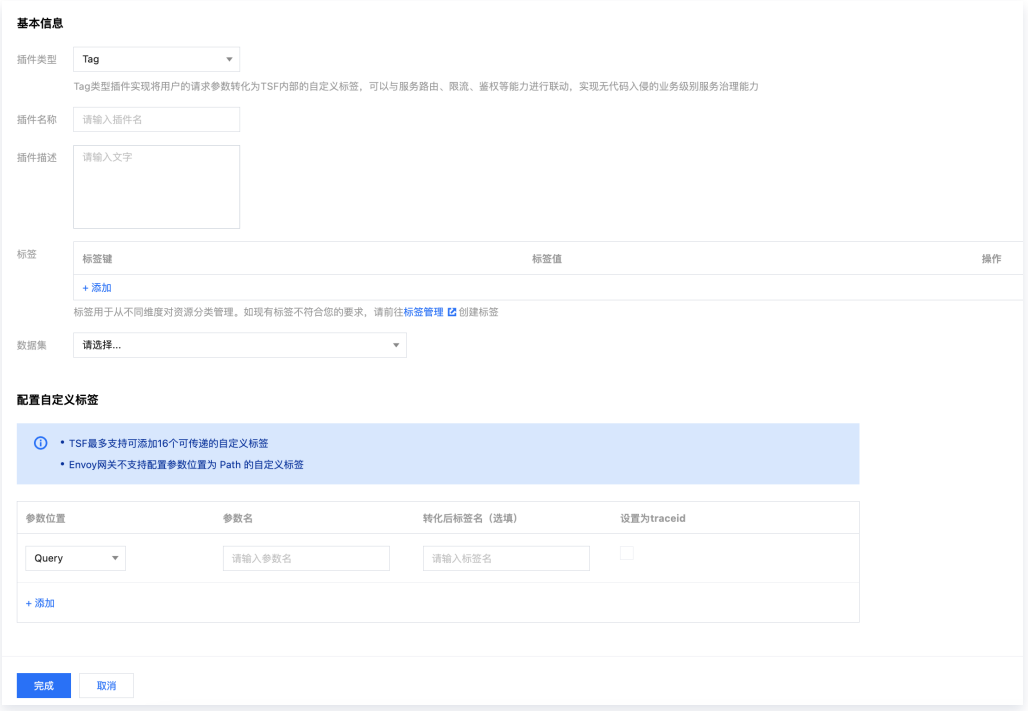
5. 将微服务 API 导入到分组中，并将分组进行发布。



步骤2：配置微服务网关插件

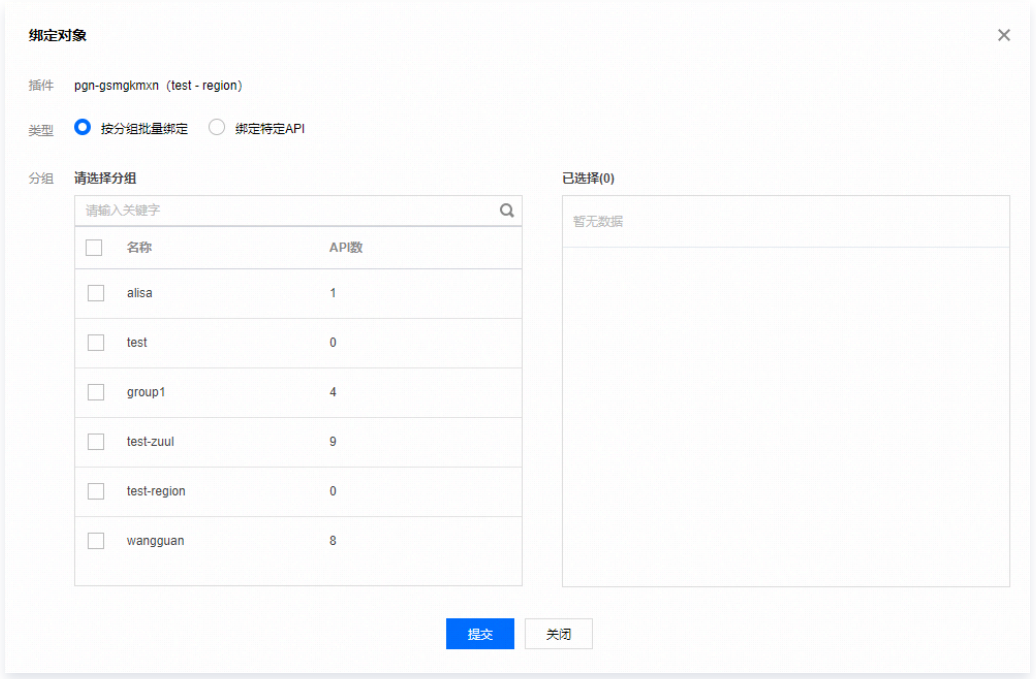
在这一步中，我们在网关配置插件，将请求参数转化为 TSF 中的标签信息。

1. 在 TSF 控制台 左侧导航栏中，单击**微服务网关 > 插件管理**。
2. 在插件管理页左上角，单击**新建插件**，填写以下信息：
   - 插件类型：选择 Tag。
   - 插件名称：最长60个字符。



3. 单击**完成**，跳转至插件管理列表页。

4. 选择目标插件，单击操作列的**绑定对象**，将创建好的插件与步骤1中创建的分组进行绑定。



5. 单击**提交**，完成配置。

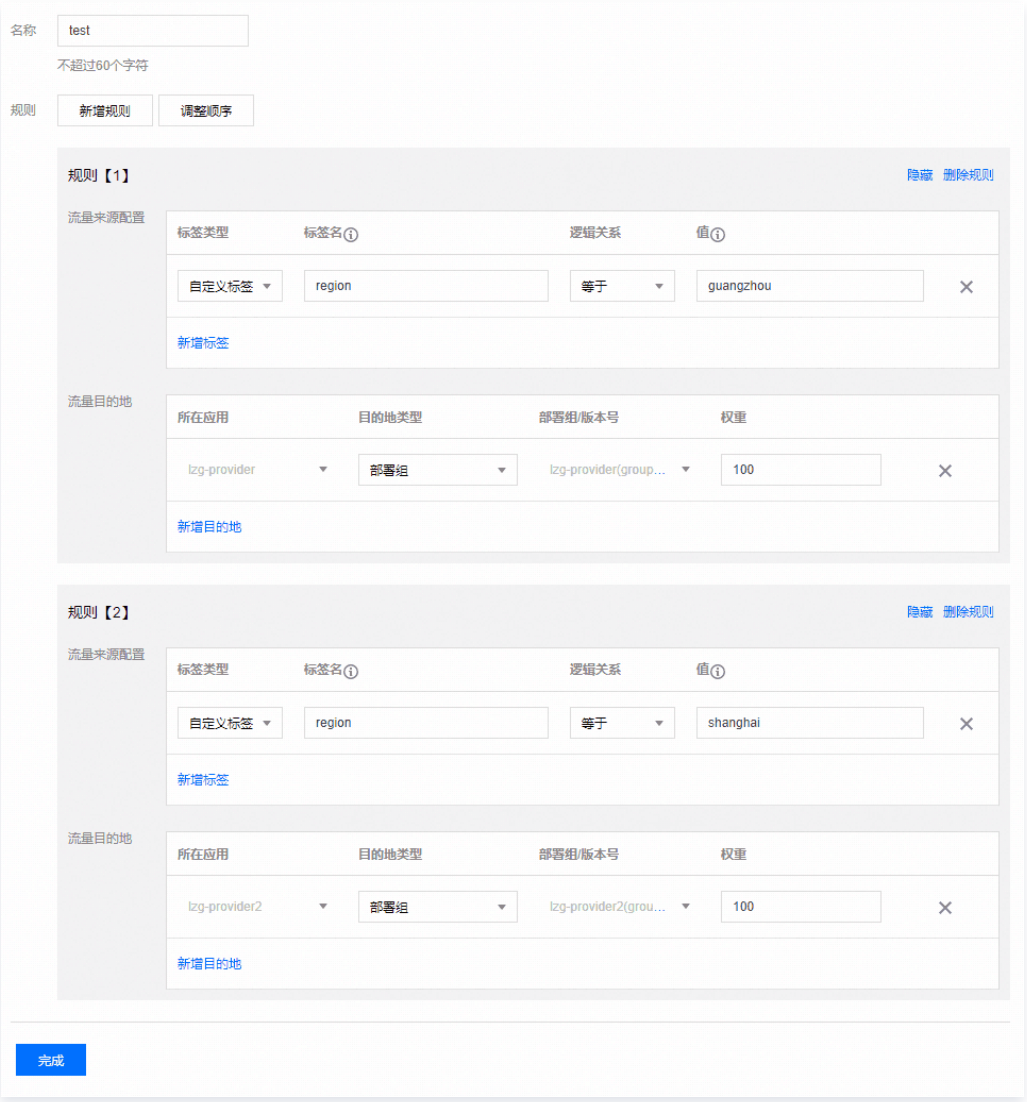
步骤3：配置服务治理规则

在这一步中，我们配置依据步骤2已经转化的标签，配置服务治理规则。当前 TSF 标签可以与服务路由、服务鉴权、服务限流、调用链进行联动。此处将为您介绍路由、鉴权、以及调用链联动功能。

服务路由

1. 在 TSF 控制台 左侧导航栏中，单击**服务治理**，进入服务列表页。
2. 找到创建好的 provider - demo 微服务，单击微服务名称，进入服务详情页。
3. 在服务详情页顶部，单击**服务路由**，进入服务路由页面。
4. 在服务路由页左上角，单击**新建路由规则**，创建两条规则：
   - 当自定义标签 region 值为 guangzhou 时，100%的流量指向部署了 provider-demo-guangzhou.jar 的部署组。

- 当自定义标签 region 值为 shanghai 时，100%的流量指向部署了provider-demo-shanghai.jar 的部署组。规则配置可参考下图：



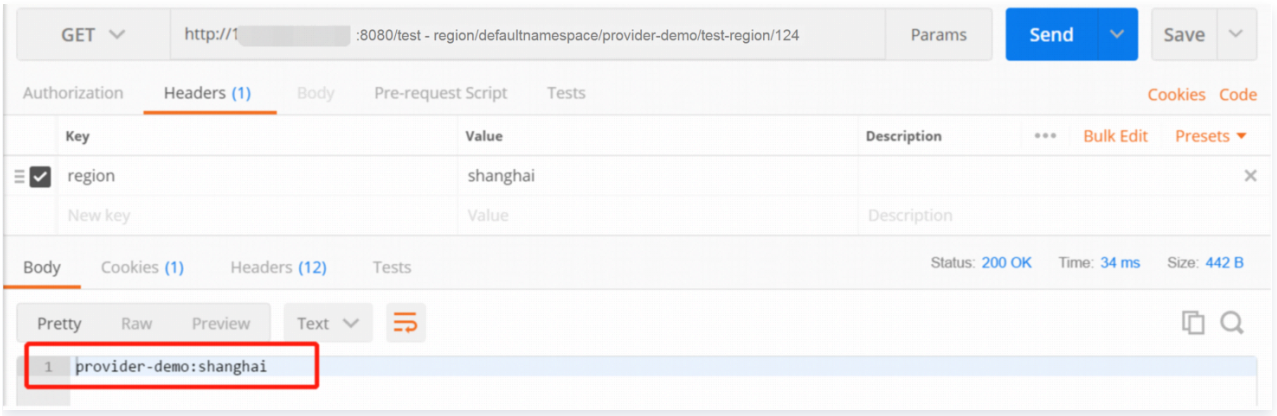
> **说明**
> 路由规则始终是在被调用方进行配置的。

5. 单击**完成**，该规则即可生效。

结果验证：

公网发送请求： `http://129.XX.XX.XX:8080/test-region/tj-test-1_default/provider-demo/test-region/12` **并携带** Header 参数 region = shanghai

若配置的路由生效，则请求会路由到返回值为 shanghai 的部署组中。



同理，当请求 Header 参数 = guangzhou 时，则请求会路由到返回值为 guangzhou 的部署组中。

服务鉴权

1. 在 TSF 控制台 左侧导航栏中，单击**服务治理**，进入服务列表页。
2. 找到创建好的 provider - demo 微服务，单击微服务名称，进入服务详情页。
3. 在服务详情页顶部，单击**服务鉴权**，进入服务鉴权页面。
4. 选择服务鉴权方式为黑名单鉴权，并单击页面左下角的**新建鉴权规则**。
5. 配置鉴权规则。当请求参数 Header 中携带了参数 usertype = user 时，请求不通过，且永久生效。配置方式如下：
   - 标签类型：自定义标签
   - 标签名：usertype
   - 逻辑关系：等于
   - 值：user
   - 生效状态：永久生效
6. 单击**完成**，完成服务鉴权，自动跳转至服务鉴权页面。

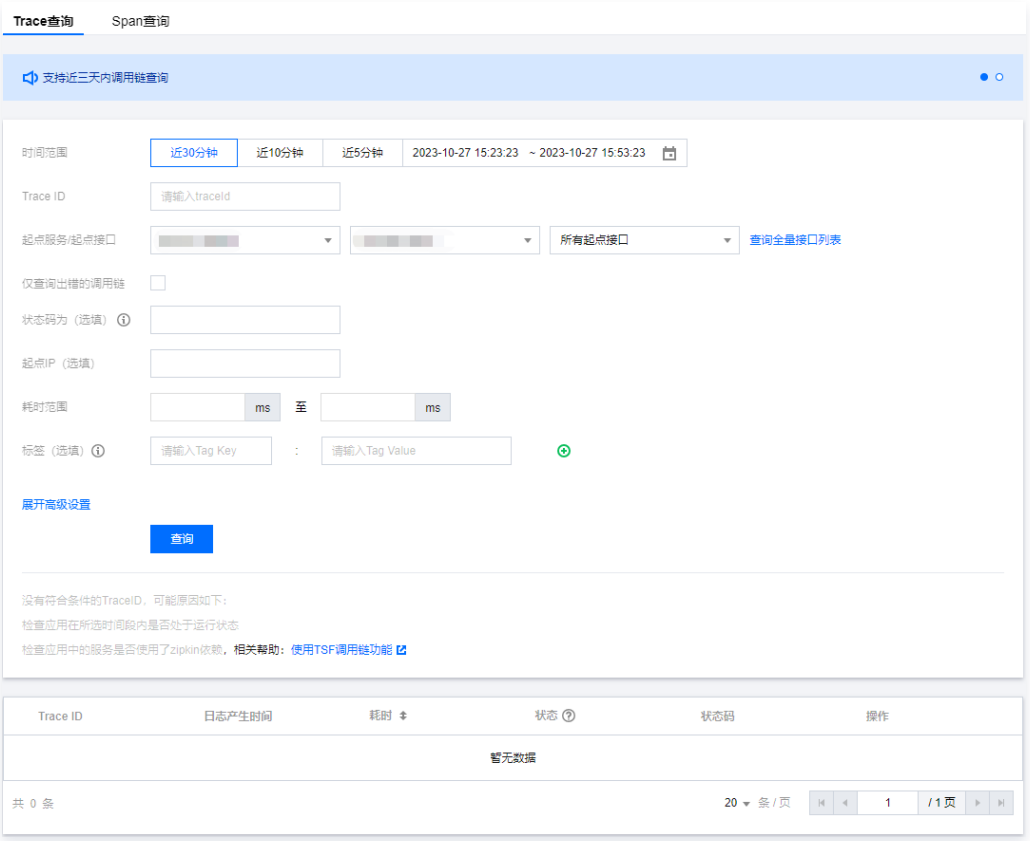
结果验证：
同理发送请求，携带 Header 参数 usertype = user，region = shanghai。发现返回请求失败。

调用链查询

在 TSF 中，我们提供了基于请求标签过滤调用链的能力，您可以依据业务数据过滤对应请求的调用链。最为常见的场景是查询某个用户 ID 的请求调用成功失败情况以及层级耗时。
使用方法：

1. 在 TSF 控制台 左侧导航栏中，单击**依赖分析** > **调用链查询**，进入调用链查询页面。
2. 选择对应的命名空间和微服务，单击 “展开高级设置” 。
3. 输入查询标签。
   我们输入 userid：1000001，过滤 userid 为1000001的请求数据。

4. 单击**查询**，即可查看携带对应标签的请求 Trace 数据列表。



容器服务实例优雅下线

最近更新时间：2024-01-19 11:48:11

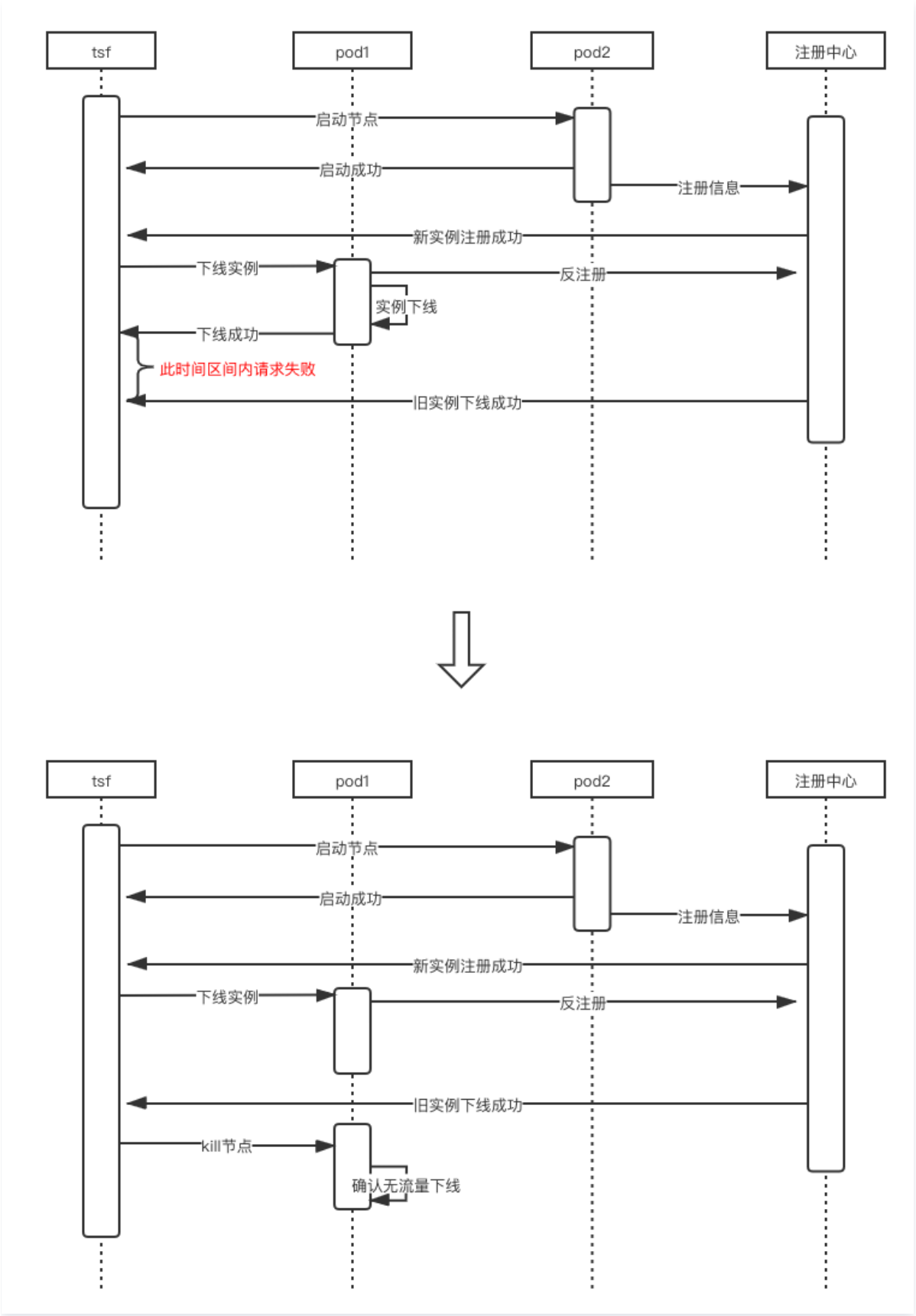
操作场景

当应用在滚动发布时，应用实例下线后向注册中心注销实例信息，服务调用方在监听到实例下线更新本地服务实例列表期间仍会将流量分发到已经下线的服务实例，造成业务异常。因此在应用滚动发布过程中，可以通过调整部署组滚动发布更新配置达到业务无中断优雅下线的目的。

> **说明**
> 目前容器集群与虚拟机集群对应的部署组应用的服务都支持优雅下线操作。

实现原理

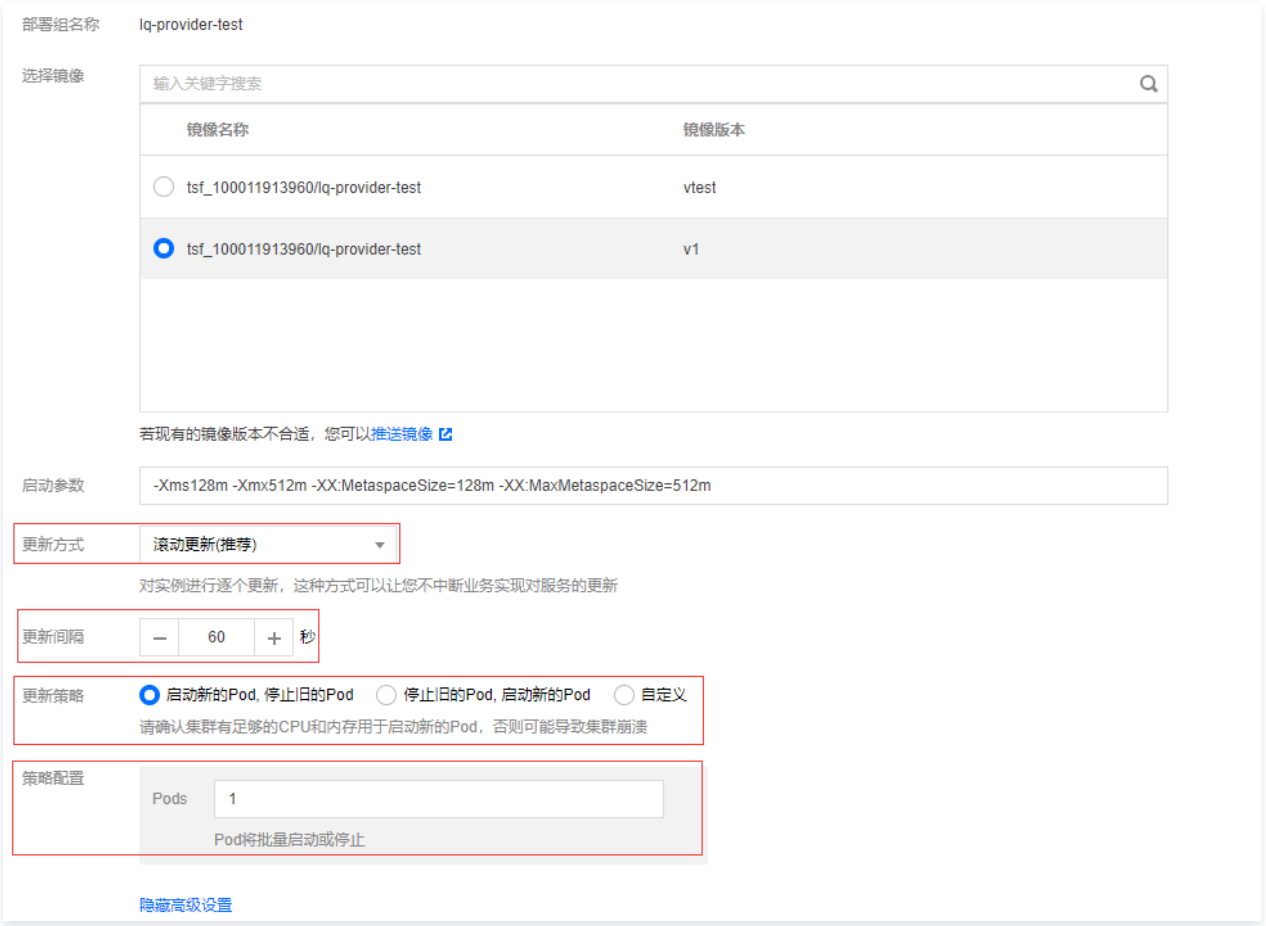
当容器应用进行滚动更新时，provider 实例接收到实例下线通知，向注册中心注销实例信息后实例下线此时实例下线的变更并未推送至 consumer，consumer 会向已经下线的实例正常发起业务调用，造成业务异常。
当启动容器服务滚动发布优雅下线配置后，TSF 会先启动一个 provider 实例向服务注册中心注册实例，而后向下线实例发送下线通知注销实例，待 consumer 将流量路由到新上线实例后在进行原有实例的服务下线，保障滚动发布升级时服务无中断。



操作步骤

1. 登录 TSF 控制台 。
2. 在左侧导航栏中，找到**应用中心**分类，单击**应用管理**，进入应用列表页。

3. 在应用列表页，单击应用 ID（或筛选应用），进入应用详情页 。
4. 单击顶部**部署组**，再选择指定的部署组，单击操作列的**部署应用**。
5. 在**资源配置**字段勾选**部署 agent 容器**并设置 agent 容器的 CPU 和内存 request 和 limit。
6. 单击**展开高级设置**，配置滚动发布策略按如下推荐配置启动服务优雅下线能力。
   - **配置更新方式**：滚动更新。滚动更新会在应用发布时按照更新策略启动新实例销毁旧实例，在整个发布过程中保证实例数一致。
   - **更新间隔**：60秒。等待pod就绪的最小时间设置为60s，保证 pod 实例正常启动并注册到 consul 后，被确认为就绪状态并继续滚动更新。
   - **更新策略**：启动新的 pod，停止旧的 pod。选择该策略则先启动新实例完成流量导入后再删除旧实例。
   - **策略配置**：pods 数为1，选择 pods 数为1则每次更新则先启动一个新实例，再下线一个旧实例。



7. 单击**提交**，即可完成对容器服务实例服务的优雅下线。

> **注意**
> 已经运行的部署组，需要通过配置上述滚动更新发布策略并进行重新部署后 pod 实例才具备优雅下线能力。

通过接口获取 TSF 监控数据

最近更新时间：2023-07-18 17:30:51

操作场景

TSF 记录了业务丰富的监控数据，控制台提供了多种监控数据展示入口。TSF 提供了部分监控数据接口的 API，客户可基于这些接口对数据进行二次加工、打造更符合业务需求的监控展示平台。
本教程将介绍已开放的监控数据接口。

前提条件

在开始本文的实践前，您需要先了解 TSF 的以下功能：

- Spring Cloud 应用开发 - 服务监控
- Mesh 应用开发 - 查询 SiderCar 监控信息
- 在 TSF 控制台的监控
  - 服务监控
  - 部署组监控
  - 实例监控
  - 接口监控
- JVM 监控相关

操作步骤

服务调用监控统计概览

本接口为 DescribeOverviewInvocation，API 各项参数请参见 接口文档。
本接口在 TSF 控制台的入口是：概览页的服务监控部分，包括了请求量、请求错误率、平均响应耗时三个指标。
本接口可获取指定命名空间下所有服务在一段时间内的请求量、请求错误率、平均响应耗时。注意是指定命名空间下的所有服务监控信息的聚合值。

服务监控统计

本接口为 DescribeStatistics，API 各项参数请参见 接口文档。
本接口用于多种服务监控数据的统计，具体在 TSF 控制台的入口如下所示：

- 服务监控（ `Type` 传参为 `Service` ）
  - 首页 > 服务指标统计TOP 10 > 服务
  - 服务监控 > 维度选择 "服务"
  - 服务监控详情 > 服务概览
- 接口监控（ `Type` 传参为 `Interface` ）(注意仅 23 以上的 SDK 版本支持)
  - 首页 > 服务指标统计TOP 10 > 接口
  - 服务监控 > 维度选择 "接口"
  - 服务监控详情 > 接口监控
  - 服务监控详情 > 接口监控 > 服务上游（需要传 `BucketKey` 为 `UpstreamApi` ）
- 实例监控（ `Type` 传参为 `Instance` ）
  - 服务监控详情 > 实例监控
- 部署组监控（ `Type` 传参为 `Group` ）
  - 服务监控详情 > 部署组监控
- 组件监控（ `Type` 传参为 `SQL` 或 `NoSQL` ）
  - 服务监控详情 > 组件监控

使用本接口时请注意传参，尽量避免因传参导致的接口错误。

监控指标曲线

本接口为 DescribeInvocationMetricDataCurve，API 各项参数请参见 接口文档。
本接口用于查询监控指标的曲线统计图，在 TSF 控制台的接口包括了：

- 服务治理详情
- 服务监控详情

本接口支持的 Metric 和 Function 如下：

- REQUEST_COUNT（请求数量，支持 SUM）。
- FAILURE_COUNT（请求失败数量，支持 SUM）。
- FAILURE_RATE（请求失败率，支持 NONE）。
- HTTP_STATUS_INFORMATIONAL（支持 SUM）。
- HTTP_STATUS_SUCCESSFUL（支持 SUM）。
- HTTP_STATUS_REDIRECTION（支持 SUM）。
- HTTP_STATUS_CLIENT_ERROR（支持 SUM）。
- HTTP_STATUS_SERVER_ERROR（支持 SUM）。
- HTTP_STATUS_OTHER（支持 SUM）。
- RESPONSE_TIME（支持 AVG, PERCENTAGE_50, PERCENTAGE_75, PERCENTAGE_95, PERCENTAGE_99）。
- QPS（qps，支持 NONE）。
- TOP_INTERFACE（最大的五个接口占比，支持 BUCKET_5）。
- APDEX（apdex，支持 NONE）。
- DURATION_MAX（最大调用延时，支持 MAX）。
- ALL_SERVICE_COUNT（总服务数，支持 SUM）。
- ONLINE_SERVICE_COUNT（在线服务数，支持 SUM）。
- ONLINE_INSTANCE_COUNT（在线节点数，支持 SUM）。

本接口支持的 MetricDimension 如下：ServiceName, OperationName, OperationMethod, PeerServiceName, PeerOperationName, DbName, Script, Period。
本接口较复杂，建议尽量以控制台前端的传参为参考。

监控指标值

本接口为 DescribeInvocationMetricDataPoint，API 各项参数请参见 接口文档。
本接口用于查询监控指标的单个值，是一个聚合后的值，在 TSF 控制台的接口是：

- 服务监控详情 > 服务概览

本接口的参数取值可参考 DescribeInvocationMetricDataCurve 接口。

监控数据维度

本接口为 DescribeInvocationMetricDataDimension，API 各项参数请参见 接口文档。
本接口在 TSF 控制台的入口是：

- 服务监控详情 > 服务概览 > 请求概览 > 切换客户端与服务端

耗时分布散点图

本接口为 DescribeInvocationMetricScatterPlot，API 各项参数请参见 接口文档。
本接口用于查询耗时分布的散点图，在 TSF 控制台的接口是：

- 服务监控详情 > 服务概览 > 请求概览 > 响应耗时分布

本接口支持的 MetricDimension 包括了：NamespaceId, GroupId, InstanceId, OperationName, ServiceName, PeerServiceName, PeerOperationName。
本接口支持的 Metric 和 Function 如下：

- RANGE_COUNT_DURATION（响应耗时分布统计，支持 SUM）。

服务监控指标

本接口为 DescribeInovcationIndicators，API 各项参数请参见 接口文档。
本接口在 TSF 控制台的入口是服务治理页面，主要配合其他服务相关接口获取服务的监控信息。
本接口的 Dimensions 传参选择 `Service`。

Java 实例 JVM 监控数据

本接口为 DescribeJvmMonitor，API 各项参数请参见 接口文档。
本接口用于查询 Java 实例的 JVM 的监控信息，由 JVM Monitor 采集，若无 JVM 监控数据，请参阅前提条件中的 JVM 监控相关文档。注意 JVM 监控数据保留时间较短，若使用此接口，请尽量获取最近的数据。
本接口的 `RequiredPictures` 参数是以返回值属性名作为入参，具体请参阅接口文档。

说明与注意

- 由于监控接口较为复杂，建议根据 TSF 控制台前端的传参，确定 API 调用的传参。TSF 控制台前端接口可能有变化，以当前页面为准。
- TSF 会对接口调用做限频，请注意调用的频率。
- 本文涉及的各个监控数据接口在查询时间跨度较大时，需要较多的数据聚合操作。若出现接口超时，请减少查询时间跨度以保证数据正常返回。
- 若业务正常但无法查询到监控数据，请优先排查是否有监控数据落盘，路径为 `/data/tsf_apm/monitor/logs/`。若无落盘数据，请根据本文前提条件中的开发指南确认是否正确配置依赖。
- TSF 控制台上未开放的部分接口将在未来陆续开放。

Dubbo 应用迁移 TSF Mesh

最近更新时间：2024-01-19 11:48:11

操作场景

本文档指导您将现有 Dubbo 应用迁移至 TSF 平台，并采用虚拟机 Mesh 的部署方式，通过 Service Mesh 的方式实现 Dubbo 应用透明的服务注册发现和无侵入的服务治理，享受 TSF 平台一站式的微服务框架解决方案。
下文以 Dubbo Mesh Demo 为例介绍具体的迁移方式和操作步骤。Dubbo Mesh Demo 提供了 client、greet 和 hello 3个应用，3个应用之间的调用关系是：client -> greet -> hello，client 应用每隔5秒会自动发起请求。

迁移价值

- TSF 为您提供**一站式应用生命周期管理服务**。提供从应用部署到应用运行的全流程管理，包括创建、删除、部署、回滚、扩容、下线、启动和停止应用并支持版本回溯能力。
- TSF 为您提供**高效的服务注册发现能力**。支持秒级的服务注册发现并提供本地注册信息缓存、服务实例注册发现异常告警、注册中心跨 AZ 区容灾等完善的高可用保障机制。
- TSF 为您提供**细粒度服务治理能力**。支持服务和 API 多级服务治理能力，通过配置标签形式进行细粒度的流量控制，实现灰度发布、就近路由、熔断限流、服务容错、访问鉴权等功能。
- TSF 为您提供**立体化应用数据运营**。提供完善应用性能指标监控和分布式调用链分析、服务依赖拓扑、日志服务工具，帮助您高效分析应用性能瓶颈及故障问题排查。

前提条件

- 已经下载 Dubbo Mesh Demo 的代码程序包。
- 已经创建了集群和命名空间，集群中导入3个云服务器。
- 已经创建了 client、greet 和 hello 应用（虚拟机部署方式）。

操作步骤

步骤1：修改配置通过直连服务名方式访问服务提供者

配置直连服务提供者的方式，详情请参考 Dubbo 官方文档 直连提供者 。
在 Dubbo Mesh Demo 中，greet 服务需要访问 hello 服务，采用 XML 配置方式，可以配置为：

```
<dubbo:reference id="helloService" check="false"
interface="org.apache.dubbo.samples.api.HelloService"
url="dubbo://org.apache.dubbo.samples.api.HelloService:20880">
```

此处的直连地址为 `dubbo://org.apache.dubbo.samples.api.HelloService:20880` ，
`org.apache.dubbo.samples.api.HelloService` 为步骤2中 hello 服务注册的服务名。

步骤2：按照 TSF Mesh 方式构建程序包

要将 Dubbo 应用以 Mesh 的方式部署到 TSF 平台上，需要遵循 Mesh 的方式构建程序包，主要包含以下几个文件：

1. 编译好的 Dubbo 应用 jar 包

建议使用 FATJAR 包，可直接运行，如 Demo 中 greet 服务的 jar 包为 `dubbo-samples-greetservice-1.0.jar` 。

2. spec.yaml 文件

此 yaml 文件用于描述 Dubbo 应用的服务注册信息，至少需要配置服务名、服务监听端口并指定 Dubbo 协议，如下所示：

```
apiVersion: v1
```

```
kind: Application
spec:
  services:
    - name: org.apache.dubbo.samples.api.GreetingService
      ports:
        - targetPort: 20881
          protocol: dubbo
      healthCheck:
        path:
```

> **注意**
> 此处的服务名如上面的 `org.apache.dubbo.samples.api.GreetingService` 将被注册到注册中心，其它 Dubbo 应用将通过该服务名进行调用，同时，服务名需要跟该 Dubbo 应用提供的 interface 名保持一致。

3. start.sh 启动脚本

此 shell 脚本用于启动部署的 Dubbo 应用，编写时保证下面两点即可：

- 幂等执行，已经启动的 Dubbo 应用不会因再次执行该脚本而被停止。
- 后台执行，将日志输出到本地文件。

编写完后可手工执行验证是否启动成功。
以下是 Demo 中 greet 服务的启动脚本：

```
#!/bin/bash

already_run=`ps -ef|grep "dubbo-samples-greetservice-1.0.jar"|grep -v grep|wc -l`
if [ ${already_run} -ne 0 ];then
        echo "dubbo-greet already Running!!!! Stop it first"
        exit -1
fi
nohup java -jar dubbo-samples-greetservice-1.0.jar 1>stdout.log 2>&1 &
```

4. stop.sh 停止脚本

此 shell 脚本用于停止部署的 Dubbo 应用，编写完后可手工执行验证是否停止成功，以下是Demo中 greet 服务的停止脚本：

```
#!/bin/bash

pid=`ps -ef|grep "dubbo-samples-greetservice-1.0.jar"|grep -v grep|awk '{print $2}'`
kill -SIGTERM $pid
echo "process ${pid} killed"
```

5. cmdline 进程执行命令文件

TSF 平台部署在主机上的 tsf-agent 会通过检查系统运行进程中是否有 cmdline 文件中的执行命令，来验证 Dubbo 应用进程是否存活，一般设置为进程的启动命令即可，如 Demo 中 greet 服务的 cmdline：

```
java -jar dubbo-samples-greetservice-1.0.jar
```

> **注意**
> 这里不需要包括上面启动脚本中的 `nohup` 和 `1>stdout.log 2>&1 &` 。

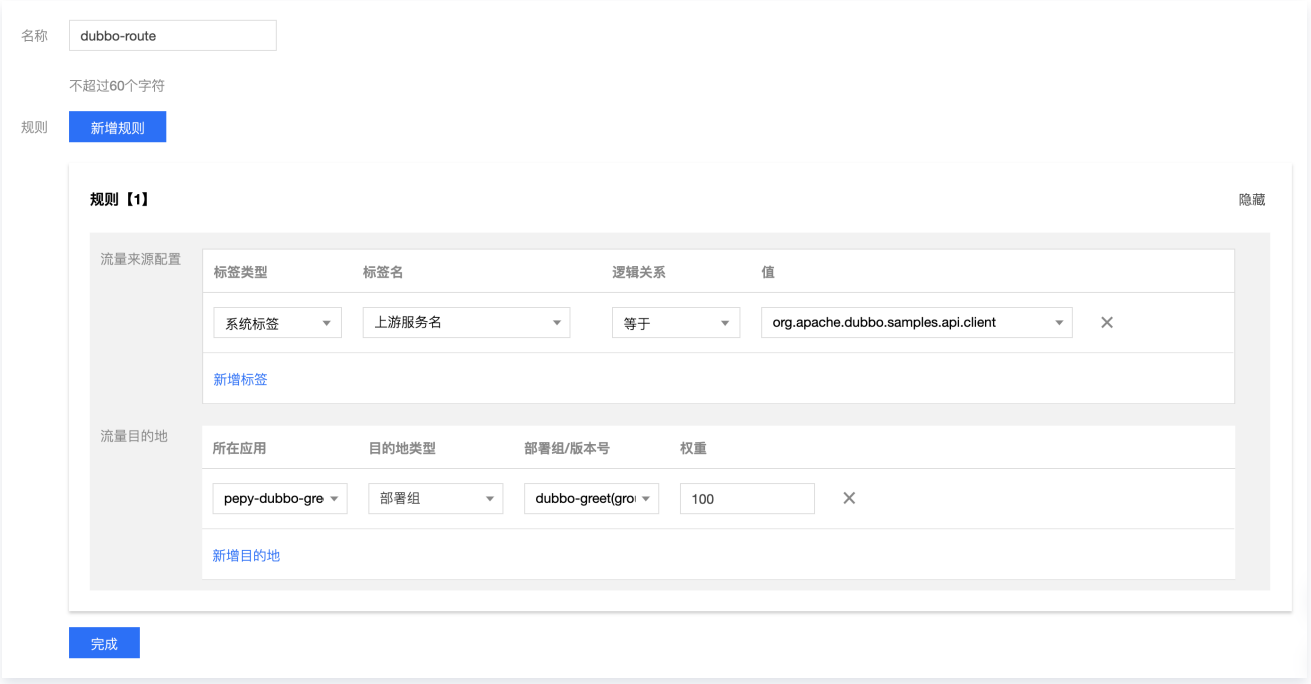
步骤3：部署 Dubbo 应用到 TSF 平台

1. 登录 TSF 控制台 。
2. 在左侧导航栏中，单击 应用管理，进入应用列表页，选择已创建的 Dubbo 应用，进入应用详情页。
3. 单击顶部**程序包管理**，上传步骤2中构建好的程序包。
4. 单击顶部**部署组**，在部署组列表页面创建部署组，选择集群和命名空间，添加实例，选择相应版本的程序包，完成部署，具体操作方式参考 虚拟机应用部署组 。
5. 部署完成后，在应用的**部署组**页查看对应部署组的状态，如果是 `运行中` 表示部署成功。
6. 在左侧导航栏中，单击 服务治理，进入服务列表页，如果可以看到步骤2中 spec.yaml 文件指定的服务名且是 `在线` 状态，表示该 Dubbo 服务注册成功，其它 Dubbo 服务可通过此服务名进行调用。

步骤4：配置服务治理规则

Dubbo 服务路由

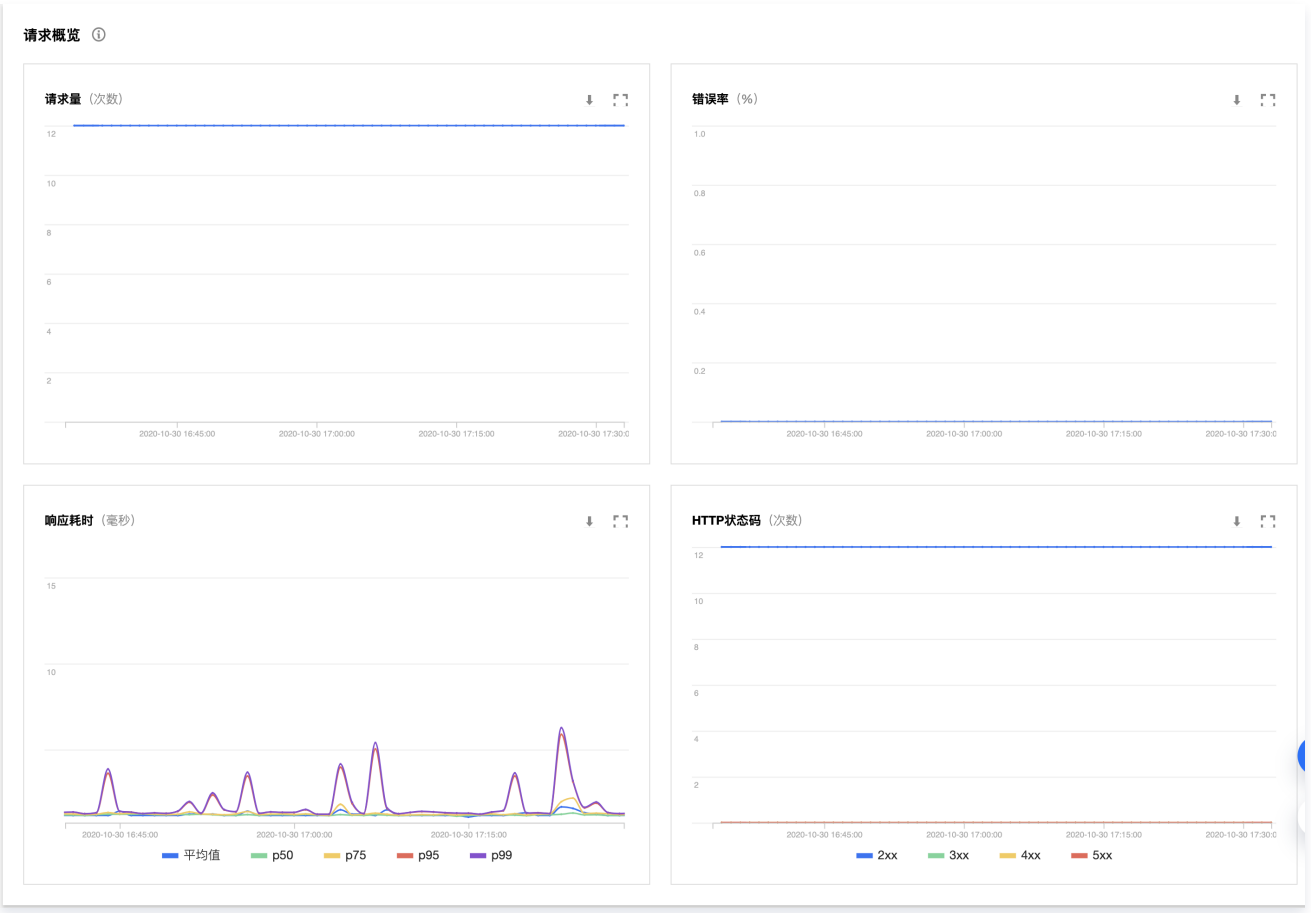
1. 在左侧导航栏中，单击 服务治理，进入服务列表页，单击需要设置路由的目的 Dubbo 服务名，进入服务详情页。
2. 单击顶部**服务路由**， 新建路由规则，具体操作方式参考 服务路由使用方法 ，如下图所示：



Dubbo 服务监控

1. 在左侧导航栏中，单击 服务治理，进入服务列表页，单击需要监控的 Dubbo 服务名，进入服务详情页。

2. 单击顶部**服务概览**，可查看该服务依赖拓扑、请求概览等信息，如下图所示：



Dubbo 调用链追踪

1. 在左侧导航栏中，单击 服务依赖拓扑，进入服务依赖拓扑页面。
2. 在顶部选择对应 Dubbo 服务所在的命名空间，再选择对应时间段，页面将自动显示该命名空间下所有服务的调用关系图，如下图所示：

泳道标流经 Kafka 后持续传递能力

最近更新时间：2022-06-13 17:03:29

操作场景

创建好泳道后，当有组件为 Kafka 时，我们希望染色流量能在流经 Kafka 后被对应泳道中的部署组消费，而未染色的消息被不在任何泳道中的部署组消费。并且希望泳道标（即 `LaneId`）能在消息流经 Kafka 后继续传递给下游服务。

名词解释

- 基线部署组：不在任何泳道中的部署组
- 泳道部署组：在任意泳道的部署组

前提条件

- 当前仅 1.23.17-Greenwich 及其之后的版本的 SDK 支持该能力。
- 需要使用 `KafkaTemplate` 发送消息，使用 `KafkaListener` 接收消息。

操作步骤

开启泳道标流经 Kafka 后持续传递能力

配置 `tsf.lane.kafka.laneOn` 为 `true`。该能力默认关闭，可通过本地配置或应用配置修改。

开启后，使用 `KafkaTemplate` 发送消息时，若消息生产者所在部署组为泳道部署组，发送到 Kafka 的消息将会带有当前泳道部署组所在泳道的泳道标（即 `LaneId`）。

开启后，使用 `KafkaListener` 接收消息时，基线部署组将消费不带泳道标的消息，即消费未染色消息。当服务**没有泳道部署组**时，即该服务仅有基线部署组，此时基线部署组将默认消费带泳道标的消息，即消费染色消息。消息消费者所在部署组的泳道标也会被替换为消息携带的泳道标。

支持基线部署组消费带泳道标的消息

当服务**有泳道部署组**，但是泳道部署组**不在线或手动下线**时，支持通过配置 `tsf.lane.kafka.mainConsumeLane = true` 使得基线部署组消费带泳道标的消息。该配置默认为 `false`。

支持泳道部署组消费基线消息

支持通过配置 `tsf.lane.kafka.laneConsumeMain = true` 使得泳道部署组可消费基线消息。该配置默认为 `false`。

跨线程能力支持

我们实验性地提供了支持泳道标跨线程传递的能力。

`CrossThreadLocal` 提供了跨线程场景下的线程池实现，`CrossCallable` 与 `CrossRunnable` 实现了对原生 `Callable` 和 `Runable` 的增强以提供跨线程能力。以 `CrossThreadLocal` 和 `CrossRunnable` 为例，使用方法如下所示。

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class CrossRunnableTest {

    private static ExecutorService executorService = Executors.newCachedThreadPool();

    private static ThreadLocal<Integer> threadLocal = new CrossThreadLocal<>();

    public static void main(String[] args) {

        for (int i = 0; i < 10; i++) {
            Integer val1 = (int) (Math.random() * 100);
            threadLocal.set(var1);
```

```
            executorService.submit(CrossRunnable.get(() -> {
                Integer val2 = threadLocal.get();
                System.out.println(val2.equals(var1));  // true
            }));
        }

        executorService.shutdown();
    }
}
```

当开启基线部署组消费带泳道标的消息，此时基线部署组将消费带泳道标消息。为了使得下游服务知道当前消息是染色流量，我们将在 `KafkaListener` 处理该消息时候临时将消费者所在部署组的泳道标修改，并在 `KafkaListener` 处理完消息后将泳道标清除。因此开发者可利用跨线程能力将泳道标传递到下游的服务中。

下面是一个简单的 demo 展示对跨线程能力的支持。当基线服务消费泳道消息时，注释 `1`、`2`、`3` 处将打印泳道标，而注释 `4` 处不会打印泳道标，因为 `3` 使用了跨线程功能 `CrossRunnable`，使得泳道标可跨线程传递。当基线服务在消费泳道消息后，泳道标将会从线程中清理，此处再次消费基线消息，注释 `1`、`2`、`3`、`4` 都不会打印泳道标。从而实现了基线服务消费泳道消息后能将泳道标传递到下游服务，并且不污染基线服务。

```
@Component
public class KafkaReceiver {
    private static ExecutorService executorService = Executors.newCachedThreadPool();

    private static ExecutorService executorService1 = Executors.newCachedThreadPool();

    @KafkaListener(topics = "test")
    public void listen(ConsumerRecord<?, ?> record) {
        Optional<?> kafkaMessage = Optional.ofNullable(record.value());

        if (kafkaMessage.isPresent()) {
            Object message = kafkaMessage.get();

            logger.info("before lane id: {}", TsfLaneIdHolder.getLaneId());
// 1
            logger.info("before cross lane id: {}", TsfLaneIdHolder.getCrossLaneId());      // 2

            executorService.submit(CrossRunnable.get(() -> {
                String laneId = TsfLaneIdHolder.getCrossLaneId();
                logger.info("cross lane id: {}", laneId);
// 3
            }));

            executorService1.submit(() -> {
                String laneId = TsfLaneIdHolder.getLaneId();
                logger.info("no cross lane id: {}", laneId);
// 4
            });

        }

    }
}
```

TSF 基于 CODING 自动化部署插件

最近更新时间：2025-04-29 16:23:22

操作场景

TSF 应用可以使用 CODING 流水线进行部署，通过流水线及插件一键创建应用、上传文件、创建部署组、导入服务器、部署应用。
基于 CODING 在 TSF 平台做自动化部署主要分为两部分内容，第一部分是代码仓库接入 CODING 并结合自身代码的编译流程实现自动化编译（CI），第二部分是将编译得出的代码包或者镜像通过 TSF 插件部署到TSF平台中（CD）。

前置条件

参见 CODING 快速开始，使用 CODING 进行自动化编译（CI）。

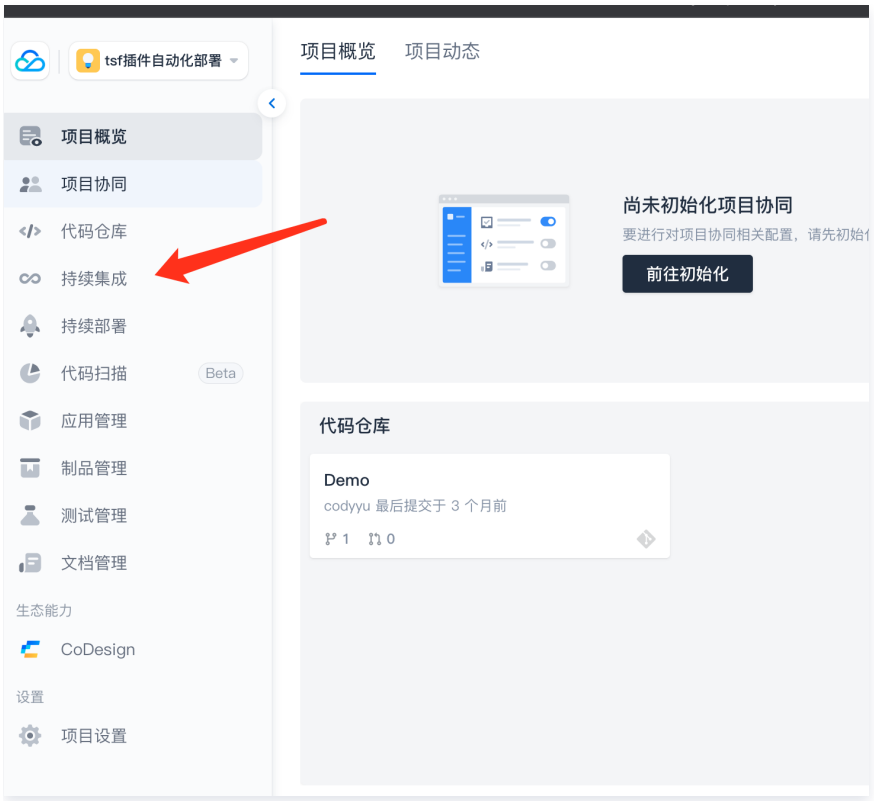
CD-基于 TSF 插件自动化部署

1. 插件概览

> **注意：**
> 新版本插件名后面有（new）标识。

插件名称	插件功能	版本	插件 ID
TSF 创建应用	创建 TSF 应用	1.0	tsf_general_create_application
TSF 虚机应用程序包上传	上传程序包到 TSF 应用中	1.0	tsf_public_upload_package
TSF 创建配置项	创建 TSF 配置项	1.0	tsf_general_create_config
TSF 创建文件配置项	创建 TSF 文件配置项	1.0	tsf_general_create_file_config
TSF 创建公共配置项	插件 TSF 公共配置项	1.0	tsf_general_create_public_config
TSF 发布配置项	发布 TSF 配置项	1.0	tsf_general_release_config
TSF 发布文件配置项	发布 TSF 文件配置项	1.0	tsf_general_release_file_config
TSF 发布公共配置项	发布 TSF 公共配置项	1.0	tsf_general_release_public_config
TSF 创建容器部署组	创建 TSF 容器部署组	1.0	tsf_general_create_container_group
TSF 部署容器应用	部署 TSF 容器应用	1.0	tsf_general_deploy_container_group
TSF 创建部署组	创建 TSF 虚机部署组	1.0	tsf_general_create_group
TSF 部署虚机应用	部署 TSF 虚机应用	1.0	tsf_general_deploy_vm_group
TSF 添加实例	向 TSF 虚机部署组添加实例	1.0	tsf_general_vm_group_add_instance
TSF 创建泳道	创建 TSF 泳道	1.0	tsf_general_create_lane
TSF 创建泳道规则	创建 TSF 泳道规则	1.0	tsf_general_create_lane_rule
TSF 启用泳道规则	开启 TSF 泳道规则	1.0	tsf_general_enable_lane_rule

2. 插件编排(流程配置)



选择项目，单击持续集成，进入流程编排。针对不同部署方式，配置不同流水线，建议的编排顺序如下：

虚机方式部署

首次创建 TSF 资源，插件可以参考以下顺序在 CODING 中编排好。1-1步骤属于 CI 流程，通过该步骤构建出可用于部署的物料包，其余各 TSF 插件可按需求选用。

容器方式部署

首次创建 TSF 资源，插件可以参考以下顺序在 CODING 中编排好。2-1步骤属于 CI 流程，通过该步骤构建出可用于部署的镜像，制作容器镜像参考该文档：制作容器镜像 。其余各TSF插件可按需求选用。

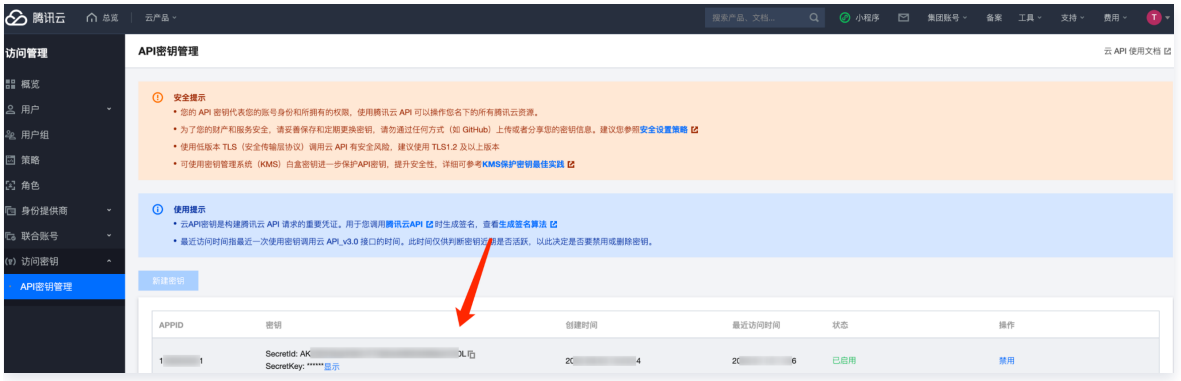
3. 公共环境变量配置

通过批量添加环境变量导入以下公共变量

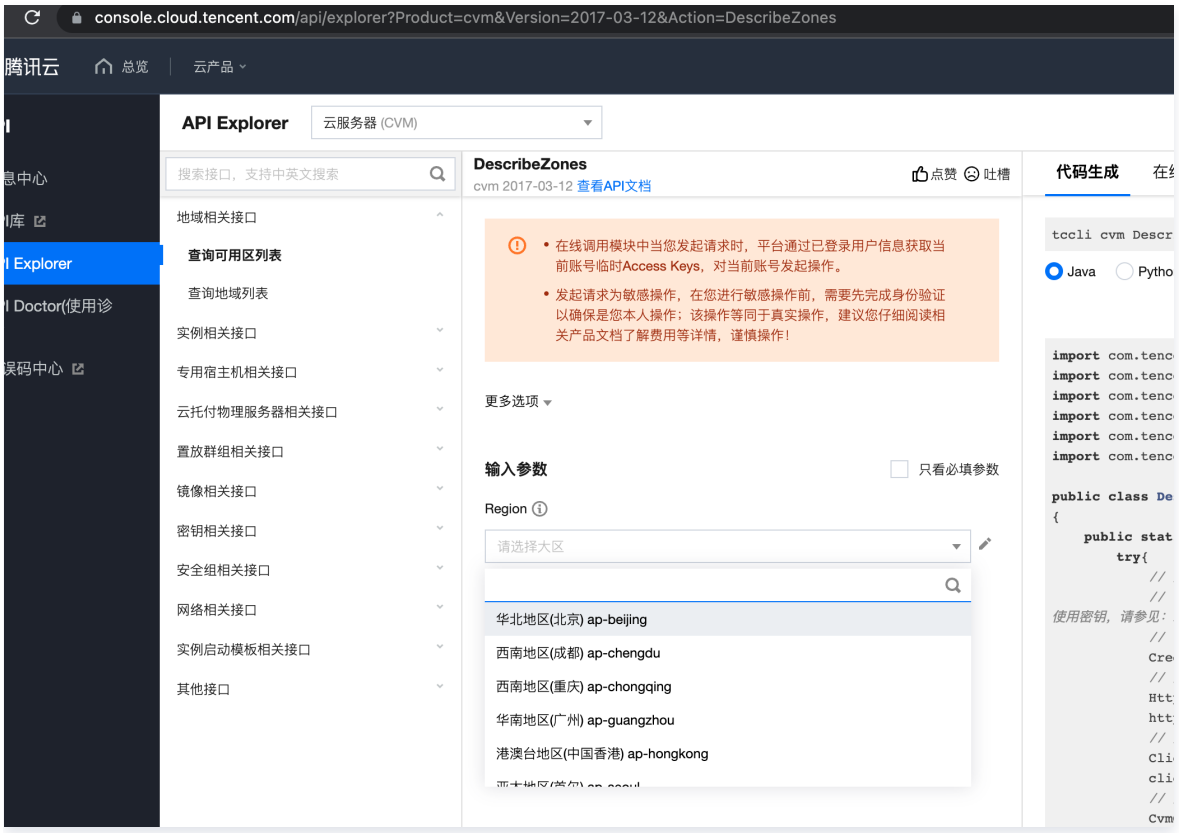

```
SECRET_ID:xxx
SECRET_KEY:xxx
REGION:ap-guangzhou
TSF_ENV:tsf_public
API_URL:tsf.tencentcloudapi.com
API_HOST:tsf.tencentcloudapi.com
ACCOUNT_APPID:xxx
```

公共环境变量详情

1. 插件使用环境：TSF_ENV: tsf_public。
2. 密钥：SECRET_ID、SECRET_KEY：在访问管理中 > 访问密钥 > API 密钥管理中获取。



3. TSF 地域：REGION: ap-guangzhou。各地域具体对应的值可以在 API Explorer 上查看。



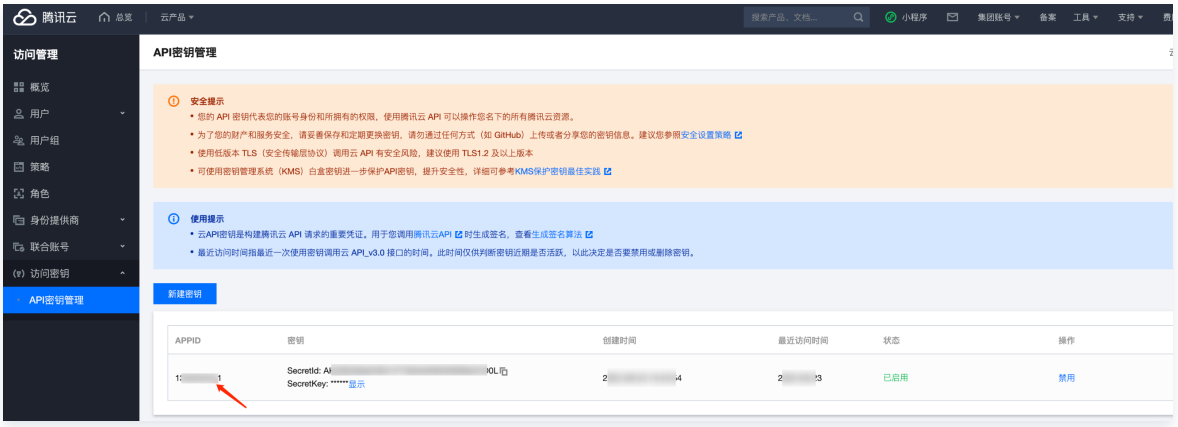
4. 云 API 地址

   API_URL：`tsf.tencentcloudapi.com`

   API_HOST：`tsf.tencentcloudapi.com`

5. 账号 ID

   APPID：API 密钥管理的 APPID



TSF插件入参说明

1. TSF 创建应用

参数名	是否必填	描述	样例
TSF应用名称	是	最长60个字符，只能包含小写字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
TSF应用类型	是	V：虚拟机应用；C：容器应用	V

TSF应用微服务类型	是	M：service mesh应用；N：普通应用；G：网关应用；	N
TSF应用需要绑定的数据集ID	否	需要绑定的数据集id	program-a22ozxja
数据集ID列表	否	需要绑定的数据集id列表	["program-a22ozxja","program-a22ozxja"]
忽略创建镜像仓库	否	true/false	false

2. TSF 虚机应用程序包上传

参数名	是否必填	描述	样例
TSF应用名称	是	最长60个字符，只能包含小写字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
应用程序包版本号	是	最长32个字符，支持 a-z，A-Z，0-9，横杠(-)、下划线(_)、点(.)	1.0.0
待上传的程序包本地路径	是	程序包所在的构建机相对路径	demo/task-schedule-example-1.2.0.jar

3. TSF 创建配置项

参数名	是否必填	描述	样例
配置项名称	是	最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
配置项版本	是	只能包含小写字母、数字及分隔符("-","."),且必须以小写字母或数字开头、以小写字母或数字结尾,中间不能有连续的"-"或"."	1.0.0
配置项值	是	配置项值	server: 127.0.0.1
应用名称	是	最长60个字符，只能包含小写字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
配置是否Base64编码	否	配置值是否使用Base64编码，使用Base64时，该参数必填	true
配置项版本描述	否	配置描述	tsf-config1
配置项值类型	否	废弃参数	废弃参数
数据集ID列表	否	需要绑定的数据集id列表	["program-a22ozxja","program-a22ozxja"]

4. TSF 创建文件配置项

参数名	是否必填	描述	样例

配置项名称	是	最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
配置项版本	是	只能包含小写字母、数字及分隔符("-",".")，且必须以小写字母或数字开头、以小写字母或数字结尾，中间不能有连续的"-"或"."	1.0.0
应用名称	是	最长60个字符，只能包含小写字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
配置项文件名	是	TSF配置项文件名	demo
配置项文件发布路径	是	TSF配置项文件发布路径	/etc/config/
配置项文件内容	是	配置项值	server: 127.0.0.1
配置项文件编码	是	utf-8 或 gbk	utf-8
后置命令	否	后置脚本命令，创建完成后，不支持修改后置命令脚本	sh /etc/nginx/conf.d/start.sh
配置是否Base64编码	否	配置值是否使用Base64编码，使用Base64时，该参数必填	true
配置项版本描述	否	配置描述	tsf-config1
数据集ID列表	否	需要绑定的数据集id列表	["program-a22ozxja","program-a22ozxja"]

5. TSF 创建公共配置项

参数名	是否必填	描述	样例
配置项名称	是	最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
配置项版本	是	只能包含小写字母、数字及分隔符("-",".")，且必须以小写字母或数字开头、以小写字母或数字结尾，中间不能有连续的"-"或"."	1.0.0
配置项值	是	配置项值	server: 127.0.0.1
配置是否Base64编码	否	配置值是否使用Base64编码，使用Base64时，该参数必填	true
配置项版本描述	否	配置描述	tsf-config1
配置项值类型	否	废弃参数	废弃参数
数据集ID列表	否	需要绑定的数据集id列表	["program-a22ozxja","program-a22ozxja"]

6. TSF 发布配置项

参数名	是否必填	描述	样例
配置项名称	是	最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo

配置项版本	是	只能包含小写字母、数字及分隔符("-","."),且必须以小写字母或数字开头、以小写字母或数字结尾,中间不能有连续的"-"或"."	1.0.0
部署组ID	是	需要发布的部署组id	group-a25e62ra
配置项发布描述	否	TSF应用配置项发布描述	config1

7. TSF 发布文件配置项

参数名	是否必填	描述	样例
配置项名称	是	最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
配置项版本	是	只能包含小写字母、数字及分隔符("-","."),且必须以小写字母或数字开头、以小写字母或数字结尾,中间不能有连续的"-"或"."	1.0.0
部署组ID	是	需要发布的部署组id	group-a25e62ra
配置项发布描述	否	TSF应用配置项发布描述	config1

8. TSF发布公共配置项

参数名	是否必填	描述	样例
TSF公共配置项名称	是	最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
TSF公共配置项版本	是	只能包含小写字母、数字及分隔符("-","."),且必须以小写字母或数字开头、以小写字母或数字结尾,中间不能有连续的"-"或"."	1.0.0
TSF命名空间名称	是	发布配置的TSF命名空间名称	tsf-demo
发布描述	否	TSF应用配置项发布描述	config1

9. TSF 创建容器部署组

参数名	是否必填	描述	样例
TSF集群名称	是	部署组所属集群名称	tsf-demo
部署组所属命名空间名称	是	部署组所属命名空间名称	tsf-demo
部署组关联的应用名称	是	部署组关联应用名称	tsf-demo
部署组名称	是	部署组名称，最长60个字符，只能包含字母、数字及分隔符（“-”），且必须以字母开头，数字或字母结尾	tsf-demo
部署组备注	否	部署组备注	group1
实例数量	是	实例数量	2
网络访问类型	是	0:公网 1:集群内访问 2：NodePort	0
端口映射，Array Of ProtocolPort	是	端口映射，Array Of ProtocolPort	[{"Protocol":"TCP","Port":18081,"Target

			Port":18081}]
初始分配的 CPU 核数	否	初始分配的 CPU 核数，对应 K8S request，request和limit两者至少填一个	0.5
最大分配 CPU 核数	否	最大分配 CPU 核数，对应 K8S limit，request和limit两者至少填一个	0.5
初始分配的内存 MiB 数	否	初始分配的内存 MiB 数，对应 K8S request，request和limit两者至少填一个	1024
最大分配内存 MiB 数	否	最大分配内存 MiB 数，对应 K8S limit，request和limit两者至少填一个	1024

10. TSF 部署容器应用

参数名	是否必填	描述	样例
部署组名称	是	TSF容器部署组名称	tsf-demo
部署组所属集群名称	是	部署组所属集群名称	tsf-demo
镜像版本名称	是	镜像版本名称,如v1	v1
镜像名	是	镜像名称，如tsf_123456789/nginx	tsf_123456789/nginx
实例数量	是	部署组实例数量, 正整数	1
部署组jvm启动参数	否	jvm参数，如"-Xms128m -Xmx512m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=512m"	-Xms128m -Xmx512m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=512m
初始分配的 CPU 核数	否	初始分配的 CPU 核数，对应 K8S request	0.5
最大分配 CPU 核数	否	最大分配 CPU 核数，对应 K8S limit	0.5
初始分配的内存 MiB 数	否	初始分配的内存 MiB 数，对应 K8S request	1024
最大分配内存 MiB 数	否	最大分配内存 MiB 数，对应 K8S limit	1024
部署组应用运行的环境变量	否	[{"Value":" -Xloggc:/data/tsf_apm/monitor/jvm-metrics/gclog.log ","Name":"JAVA_TOOL_OPTIONS"}]	[{"Value":" -Xloggc:/data/tsf_apm/monitor/jvm-metrics/gclog.log ","Name":"JAVA_TOOL_OPTIONS"}]
部署方式	否	部署方式，0表示快速更新，1表示滚动更新。默认为 0	0
是否不立即启动	否	是否不立即启动	false
kubernetes滚动更新策略的MaxSurge参数	否	和期望ready的副本数比，超过期望副本数最大比例（或最大值），这个值调的越大，副本更新速度越快。	25%
kubernetes滚动更新策略的	否	和期望ready的副本数比，不可用副本数最大比例（或最大值），这个值越小，越能保证服务稳定，更新越平滑；	25%

MaxUnavailable参数			
滚动更新必填，更新间隔（秒）	否	滚动更新必填，更新间隔（秒）	1
不创建 k8s service	否	不创建 k8s service	false
网络访问类型	是	0:公网 1:集群内访问 2:NodePort 3:VPC内网访问	0
端口映射	是	端口映射	[{"Protocol":"TCP","Port":18081,"TargetPort":18081}]
子网ID	否	VPC子网ID	subnet-ad960efo
service 是否为 headless 类型	否	Headless Service 仅支持在创建时可选，创建后不可变更访问方式	false
是否删除之前创建的 Service	否	当为 true 且 DisableService 也为 true 时，会删除之前创建的 service，请小心使用	false
是否开启 SessionAffinity	否	是否基于来源IP做会话保持	false
是否基于来源IP做会话保持时间	否	是否基于来源IP做会话保持时间	2
节点调度策略	否	NONE表示不使用调度策略；CROSS_AZ表示尽可能跨可用区部署	CROSS_AZ
存活健康检查	否	json string(参考 https://cloud.tencent.com/document/api/649/36099#HealthCheckSettings)	
就绪健康检查	否	json string(参考 https://cloud.tencent.com/document/api/649/36099#HealthCheckSettings)	
镜像server	否	镜像仓库地址，如ccr.ccs.tencentyun.com	ccr.ccs.tencentyun.com
镜像仓库类型	否	镜像仓库类型，tcr或者personal	tcr
数据卷信息	否	数据卷信息，json string, 解析后为Array of VolumeInfo(https://cloud.tencent.com/document/api/649/36099#VolumeInfo)	
数据卷挂载点信息	否	json string, 解析后为Array of VolumeMountInfo(https://cloud.tencent.com/document/api/649/36099#VolumeMountInfo)	
是否清除数据卷信息	否	是否清除数据卷信息，默认false	false
是否部署 agent 容器	否	是否部署 agent 容器	true
agent 容器分配的 CPU 核数	否	初始分配的 CPU 核数，对应 K8S request	0.1
agent 容器最大的 CPU 核数	否	最大分配 CPU 核数，对应 K8S limit	0.1

agent 容器分配的内存 MiB 数	否	初始分配的内存 MiB 数，对应 K8S request	125
agent 容器最大的内存 MiB 数	否	最大分配内存 MiB 数，对应 K8S limit	400
istio proxy 容器分配的 CPU 核数	否	初始分配的 CPU 核数，对应 K8S request	0.1
最大分配 CPU 核数，对应 K8S limit	否	最大分配 CPU 核数，对应 K8S limit	0.1
istio proxy 容器分配的内存 MiB 数	否	初始分配的内存 MiB 数，对应 K8S request	125
istio proxy 容器最大的内存 MiB 数	否	最大分配内存 MiB 数，对应 K8S limit	400

11. TSF创建部署组

参数名	是否必填	描述	样例
TSF集群名称	是	部署组所属集群名称	tsf-demo
部署组所属命名空间名称	是	部署组所属命名空间名称	tsf-demo
部署组关联的应用名称	是	部署组关联应用名称	tsf-demo
部署组名称	是	部署组名称，最长60个字符，只能包含字母、数字及分隔符（“-”），且必须以字母开头，数字或字母结尾	tsf-demo
部署组描述	否	部署组备注	group1

12. TSF 部署虚机应用

参数名	是否必填	描述	样例
部署组名称	是	TSF虚机部署组名称	tsf-demo
程序包名称	是	需要部署的程序包名称	task-schedule-example-1.2.0.jar
程序包版本	是	需要部署的程序包	1.0.0
集群名称	是	部署组所属集群名称	tsf-demo
部署应用描述信息	否	部署应用描述信息	deploy1
JDK名称	否	konaJDK或openJDK	openJDK
JDK版本	否	8或11 (openJDK只支持8)	8
部署组启动参数	否	TSF部署组启动参数，如"-Xms128m -Xmx512m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=512m"	-Xms128m -Xmx512m -XX:MetaspaceSize=128m -

			XX:MaxMetaspaceSize=512m
部署方式	是	部署方式，0表示快速更新，1表示滚动更新	0
是否启用beta批次	否	beta 批次：是否首次用一个实例来部署新版本，如果部署成功，才会部署后面的批次实例	false
滚动发布每个批次的时间间隔	否	滚动发布每个批次的时间间隔,单位：分钟	1
是否开启健康检查	否	是否开启健康检查	false
存活健康检查	否	json string(参考 https://cloud.tencent.com/document/api/649/36099#HealthCheckSettings)	
就绪健康检查	否	json string(参考 https://cloud.tencent.com/document/api/649/36099#HealthCheckSettings)	
是否允许强制启动	否	开启强制启动则实例忽视consul服务注册报错信息正常启动	false
启动脚本 base64编码	否	启动脚本 base64编码	
停止脚本 base64编码	否	停止脚本 base64编码	
是否进行增量部署	否	是否进行增量部署，默认为false，全量更新	false

13. TSF 添加实例

参数名	是否必填	描述	样例
TSF部署组名称	是	TSF虚拟机集群部署组名称	tsf-demo
扩容的机器实例ID列表	是	部署组要添加的虚拟机实例ID列表，json字符串，如["ins-abc","ins-def"]	["ins-abc","ins-def"]
集群名称	是	集群名称	tsf-demo

14. TSF 创建泳道

参数名	是否必填	描述	样例
泳道名称	是	最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾	tsf-demo
泳道部署组信息	是	格式为Array of LaneGroup，例如[{"GroupId":"group-xxxxx","Entrance":true}]	[{"GroupId":"group-xxxxx","Entrance":true}]
泳道备注	否	泳道备注	lane1

15. TSF 创建泳道规则

参数名	是否必填	描述	样例

泳道ID	是	泳道ID	lane-all4popx
泳道规则名称	是	不能为空。最长60个字符	tsf-demo
泳道规则标签列表	是	格式为Array of LaneRuleTag，例如[{"TagName":"test","TagOperator":"EQUAL","TagValue":"1"}]	[{"TagName":"test","TagOperator":"EQUAL","TagValue":"1"}]
泳道规则标签关系	是	泳道规则标签关系，与：RELEATION_AND，或：RELEATION_OR	RELEATION_AND
泳道规则备注	否	泳道规则备注	laneRule1

16. TSF 启用泳道规则

参数名	是否必填	描述	样例
TSF泳道规则ID	是	TSF泳道规则ID	lane-r-yd9om5mx