微服务平台 TSF

应用迁移

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

应用迁移
Spring Cloud TSF 应用迁移

最近更新时间：2023-11-01 17:50:42

操作场景

该文档帮助您将自有集群的应用服务平滑迁移至 TSF，并在迁移过程中确保服务正常的注册发现。
当您的 Spring Cloud 应用集群已经部署在云上的生产环境并承接业务流量，同时您想将 Spring Cloud 应用迁移至 TSF 享受一站式的微服务框架解决方案，在迁移过程中保障业务流量不中断，完全实现对业务用户的透明访问即为平滑迁移。

前提条件

您已经将 Spring Cloud 应用部署在腾讯云上。

迁移价值

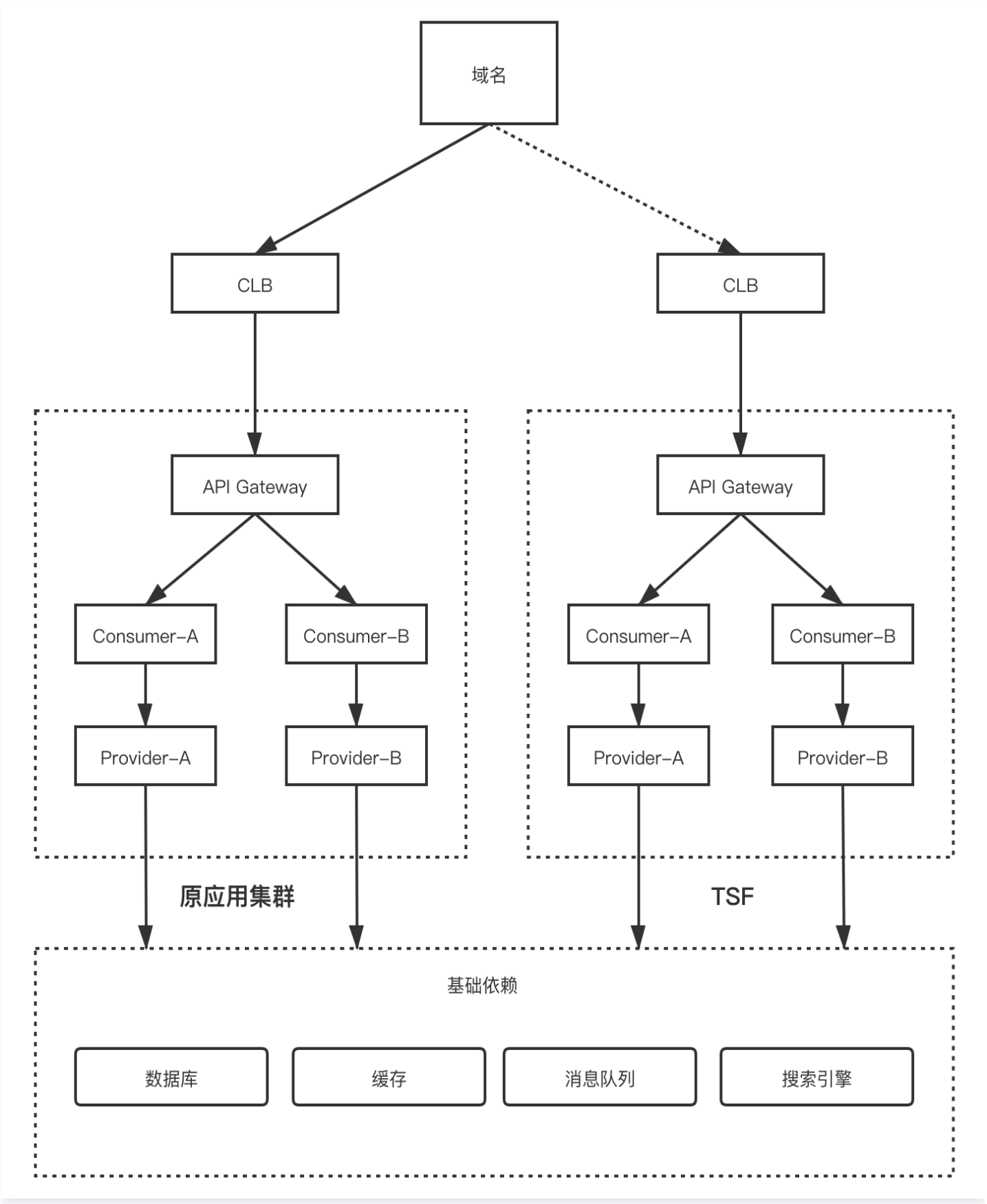
- TSF 为您提供**一站式应用生命周期管理服务**。提供从应用部署到应用运行的全流程管理，包括创建、删除、部署、回滚、扩容、下线、启动和停止应用并支持版本回溯能力。
- TSF 为您提供**高效的服务注册发现能力**。支持秒级的服务注册发现并提供本地注册信息缓存、服务实例注册发现异常告警、注册中心跨AZ区容灾等完善的高可用保障机制。
- TSF 为您提供**细粒度服务治理能力**。支持服务和 API 多级服务治理能力，通过配置标签形式进行细粒度的流量控制，实现灰度发布、就近路由、熔断限流、服务容错、访问鉴权等功能。
- TSF 为您提供**立体化应用数据运营**。提供完善应用性能指标监控和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

迁移方案

为保障应用迁移过程中流量不中断，您可以通过**切流迁移**和迁移实例的**双注册发现迁移**两种方式实现平滑迁移。具体方案如下：

切流迁移方案

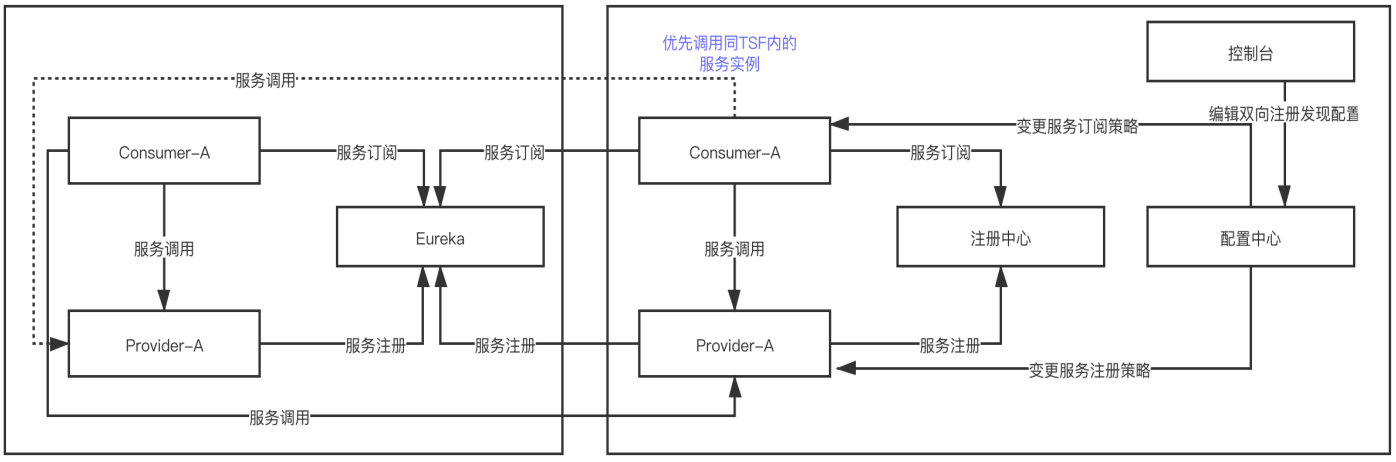
- **实现方法**：业务应用需要改造后在 TSF 集群全量部署，应用全部改造上线后通过切换域名配置将域名指向新集群的 CLB 服务均衡地址实现流量切换。
- **方案优点**：操作简单，原有应用集群和 TSF 集群不存在相互影响。
- **方案缺点**：需要保持两套集群同时运行，存在资源浪费；此外，由于是全量迁移，迁移过程中存在业务异常的概率较大。


域名
CLB
CLB
API Gateway
API Gateway
Consumer–A
Consumer–B
Consumer–A
Consumer–B
Provider–A
Provider–B
Provider–A
Provider–B
原应用集群
TSF
基础依赖
数据库
缓存
消息队列
搜索引擎

双注册发现方案

- **方案说明**：您仅需要将迁移的应用引入 TSF 组件依赖并进行双注册中心服务注册配置，将改造后的应用通过 TSF 进行部署，部署后的 Spring Cloud 应用即可被原应用集群和 TSF 集群中的服务发现，正常进行服务调用。
- **方案优点**：
  - 改造量小，仅需引入 TSF 组件依赖并进行完成配置操作，无需改动代码即可实现双注册发现能力。
  - 在迁移过程中实现已迁移应用与未迁移应用的互相发现，实现服务调用，有效保障业务连续性。
  - 整体服务迁移完成后，通过TSF提供配置管理能力可动态变更服务注册发现策略，无需重启应用即可完成迁移。
  - 迁移后的应用优先调用TSF内的服务实例，避免跨集群调用产生性能损耗。

> **说明**
> 目前 TSF 支持原有注册中心引擎为 Eureka、Zookeeper、Consul 与 TSF 注册中心引擎的双注册发现能力。



操作步骤

1. 在迁移的服务应用的 pom.xml 文件中引入 TSF 依赖：

```
<parent>
 <groupId>com.tencent.tsf</groupId>
 <artifactId>spring-cloud-tsf-dependencies</artifactId>
 <version><!-- 调整为 SDK 最新版本号 --></version>
</parent>
```

> **注意**

> 应用该功能 SDK 版本必须在1.24.0-Finchley-RELEASE、1.24.0-Greenwich-RELEASE 以上。

2. 在 `application.properties` 中添加双注册发现配置。
   exclude 字段代表不将服务注册到哪些服务注册中心引擎；subscribes 字段代表从哪些服务注册引擎发现服务信息。

```
tsf:
  migration:
    registry:
      // 不注册到ZOOKEEPER
      excludes: ZOOKEEPER
      // 从 EUREKA、TSF 进行服务发现
    subscribes:
      - EUREKA
      - TSF
```

3. 将应用打包。
   登录 TSF 控制台，参照 虚拟机托管应用 或者 容器托管应用 完成应用发布。
4. 验证结果。
   通过原有集群服务调用可发现服务是否正常，此外在**应用中心** > **服务治理**查看应用发布地域及命名空间查看是否存在服务信息，如果服务注册正常则具有该微服务名称、状态、运行实例、请求量等信息。
5. 在以上应用验证成功后，可逐步其他应用。
6. 清理迁移配置。
   迁移完成后，登录 TSF 控制台，参照 配置管理 动态变更服务注册发现方式，将原有集群的注册中心配置移除。

Spring Cloud 原生应用迁移

最近更新时间：2025-03-10 18:08:32

迁移价值

Spring Cloud 提供了微服务系统架构的一站式解决方案，并利用 Spring Boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控、分布式调用链等，通过 Spring Cloud 提供的一套简易的编程模型，我们可以在 Spring Boot 的基础上轻松地实现微服务项目的构建。
然而，这仅仅是帮助开发者开发微服务应用带来了便利，最终开发出的应用需要在生产环境运行起来，因此，我们还需要：

- **稳定的运行环境**，如虚机环境或者容器环境，并能对微服务应用进行生命周期管理。
- **高可用的注册中心**，Spring Cloud 虽然支持使用 Eureka、Zookeeper 或 Consul 实现服务注册发现，但需要我们自行搭建并保证其高可用。
- **统一的控制平面对服务进行治理**，Spring Cloud 整合了大量的服务治理组件，但没有一个统一的控制平面进行管理，这对大规模的微服务治理带来了不少的压力。
- **可视化的数据运营服务**，例如日志服务、监控服务、分布式调用链服务，例如 Spring Cloud 通过引入 Sleuth 实现分布式调用链，并和 Zipkin、HTrace、ELK兼容，但这些后端服务需要我们自行搭建并保证其稳定性。

也就是说，我们开发者开发完应用、构建好Jar包也只是第一步，要让应用稳定运行和持续运营，还需要一个微服务平台来满足上面的要求，这正是TSF——腾讯微服务平台（Tencent Service Framework）的价值所在：

- TSF 提供了**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 提供了**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 提供了**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 提供了**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。

目前TSF 支持原生 Spring Cloud 应用无侵入接入，无需改造即可直接接入 TSF，享受服务注册与发现、服务治理、应用监控和调用链跟踪等功能。

操作步骤

本文档以一个 开源商城系统 为示例，为您介绍将原生 Spring Cloud 应用迁移到 TSF 的方法。
该系统由以下几部分组成：

模块	说明
mall-admin	后台管理系统。
mall-auth	角色认证模块。

mall-gateway	网关模块，请求入口。
mall-portal	前台商城系统。
mall-search	商品检索模块。
mall-demo	用来测试 API 的样例工程。
mall-monitor	Spring 自带的监控模块，TSF 自带监控能力，因此可以略过。

环境准备

环境配置建议

> **说明：**
> 以下配置仅做建议，具体以您的实际业务需求为主。

- 开发环境：指含有 mall demo 程序源码的计算环境。
- 部署环境：指购买的腾讯云主机，并且运用 TSF 服务部署商城系统的环境。

环境	环境分类	配置
开发环境	–	• CPU: 4核 • 内存：8GB • 网络：50Mbps。
部署环境	中间件部署服务器	• 配置：1台云主机 • CPU：4核 • 内存：8GB • 网络：50Mbps。
	微服务部署服务器	• 配置：每个服务1台云主机 • CPU：1核 • 内存：2GB • 网络：20Mbps，按量计费 • 磁盘：50GB。

中间件部署服务器准备

1. 参见环境配置建议 购买云服务器 。
2. 安装 Docker 和 Docker Compose。
3. 下载 mall-demo 程序包 ，并将其上传到云服务器中。
4. 进入 `tsf-demo-public/document/docker` 目录，执行如下命令，等待下载和容器拉起完成。

```
docker-compose -f docker-compose-env.yml up -d
```

> **说明：**
> 下载时间根据实际网络带宽可能需等待几分钟到几十分钟不等。

5. 执行下面的命令创建 RabbitMQ 的 virtual_host、用户和权限，需要等 RabbitMQ 启动完成，如果下面的命令报错，再次执行。

```
docker exec -it rabbitmq /init.sh
正常情况下屏幕会显示如下：
Adding user "mall" ...
Setting tags for user "mall" to [administrator] ...
Adding vhost "/mall" ...
Setting permissions for user "mall" in vhost "/mall" ...
```

迁移上云

步骤1：准备应用程序包

前提条件

安装 Maven

操作步骤

1. 下载 mall-demo程序包 到本地。
2. 在 `tsf-demo-public` 根目录下执行如下命令，进行依赖初始化，耗时根据网速可能不同。

```
mvn clean
```

3. 进入每个项目的 `src/main/resource` 目录，根据已经部署的容器所有的云服务器地址，修改 application.yml 文件中的连接信息。

> **说明：**
> 若在本地安装调试可以忽略本步骤，即在本地安装 docker 和所有基础组件，在本地启动 Spring Cloud 调试。

```
    # mysql中替换localhost为内网IP
    url: jdbc:mysql://localhost:3306/mall?
useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai
    # redis中替换localhost为内网IP
    host: localhost
```

```
    # rabbitmq中替换localhost为内网IP
    host: localhost
    # mongo中替换localhost为内网IP
    host: localhost
    # ES中替换127.0.0.1为内网IP
    uris: 127.0.0.1:9200
```

4. 进入 mall-mbg 项目的 `src/main/resource` 目录，修改 generator.properties 文件中 MySQL 的连接信息，修改localhost为指定主机名/IP。
5. 在 `tsf-demo-public` 根目录下，执行如下命令将项目进行打包。

```
mvn clean package -DskipTests
```

6. 在 target 目录下，可看到生成的 jar 程序包。

```
需上传jar包本地路径
mall-admin/target/mall-admin-1.0-SNAPSHOT            # 后台管理系统
mall-auth/target/mall-auth-1.0-SNAPSHOT              # 角色认证模块
mall-gateway/target/mall-gateway-1.0-SNAPSHOT        # 网关模块，请求入口
mall-portal/target/mall-portal-1.0-SNAPSHOT          # 前台商城系统
mall-search/target/mall-search-1.0-SNAPSHOT          # 商品检索模块
两个可选部署的服务
mall-demo/target/mall-demo-1.0-SNAPSHOT        # 用来测试API的样例工程
mall-monitor/target/mall-monitor-1.0-SNAPSHOT    # Spring自带的监控模块，TSF自
带监控能力，故可以略过
```

步骤2：部署应用到 TSF

以部署 mall-search 服务为例，介绍在 TSF 上部署一个应用的流程。

> **注意：**
> 服务部署顺序建议：服务网关 mall-gateway -> mall-auth -> mall-admin -> mall-portal -> mall-search -> mall-demon。

新建集群

1. 登录 腾讯微服务平台控制台，左侧导航栏选择**资源管理 > 虚拟机集群或容器集群**，单击新建，创建一个名为 mall-demo 的集群。
2. 单击集群操作栏的**导入云主机**，将购买的云服务器全部导入到集群中。

创建并部署应用

1. 在左侧导航栏选择**应用管理 > 业务应用**，单击**新建应用**，创建一个名为 mall-search 的应用。

2. 单击**下一步**，开启注册配置治理。
   - **注册配置治理**：开启。
   - **实例类别**：共享实例（TSF-Consul）。
   - **接入方式**：Mesh接入。
   - **实现方式**：原生应用。
3. 单击**完成**后，进入应用详情页。
4. 在**制品**卡片，单击**上传程序包**，将 步骤1 中下载的 mall-search-1.0-SNAPSHOT.jar 程序包上传到 TSF 平台。
5. 单击**应用部署**，填写部署信息。

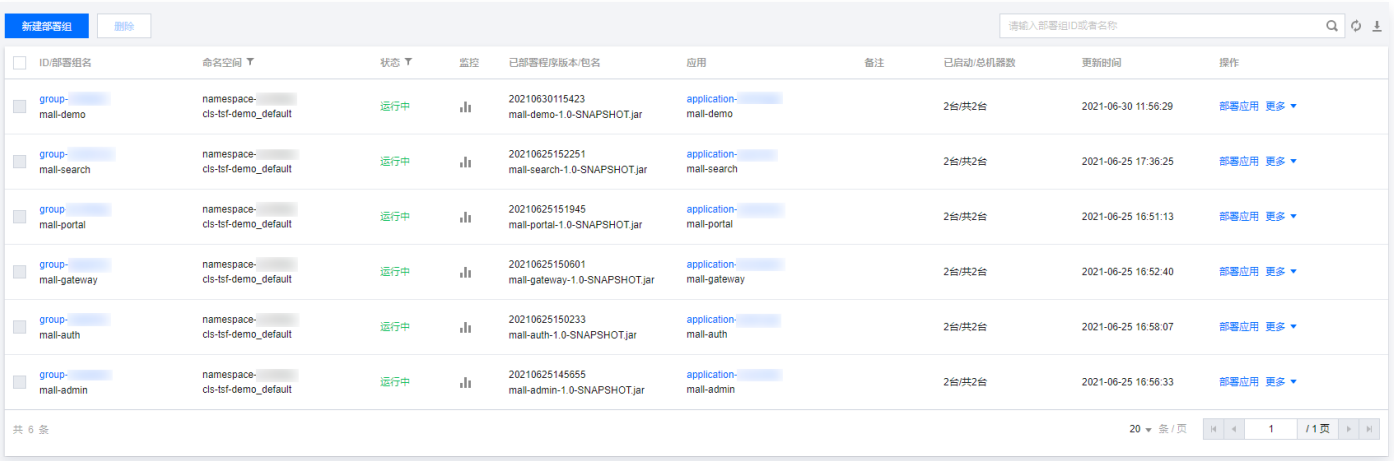
6. 单击**下一步**，进行部署配置。
7. 在部署配置页面，选择刚刚上传的程序包版本.

健康检查建议勾选“存活检查”和“就绪检查”，因为项目已经集成 actuator，如图填写请求路径即可，端口号根据 application.yml 中定义填写。

8. 单击**下一步**，进行发布策略配置。
9. 单击**下一步**，进行服务注册配置。
10. 单击**完成**，进行应用部署。

查看部署结果

重复步骤 1 到步骤 3，依次将所有服务部署到 TSF 上，服务部署顺序建议：**服务网关 mall-gateway -> mall-auth -> mall-admin -> mall-portal -> mall-search -> mall-demon。**
当完成所有的服务部署，部署结果如下。



部署结果验证

步骤1：验证服务依赖功能

通过部署前端页面，验证服务依赖功能。

1. 登录中间件部署服务器，在服务器上安装 node.js 。
2. 下载前端代码，地址 mall-admin-web 。
3. 在项目根目录下执行如下命令，安装前端项目所需的第三方依赖。

```
npm install
```

4. 修改 dev.env.js 文件中的 BASE_API 配置为网关服务的端口，示例如下：IP 为gateway 服务机器内网 IP，port 为服务的端口号。

```
http://IP:PORT/mall-admin
```

5. 执行如下命令运行前端项目。

```
npm run dev
```

6. 访问前端页面，地址：`http://中间件服务器的外网 IP: 8090` ，体验服务。

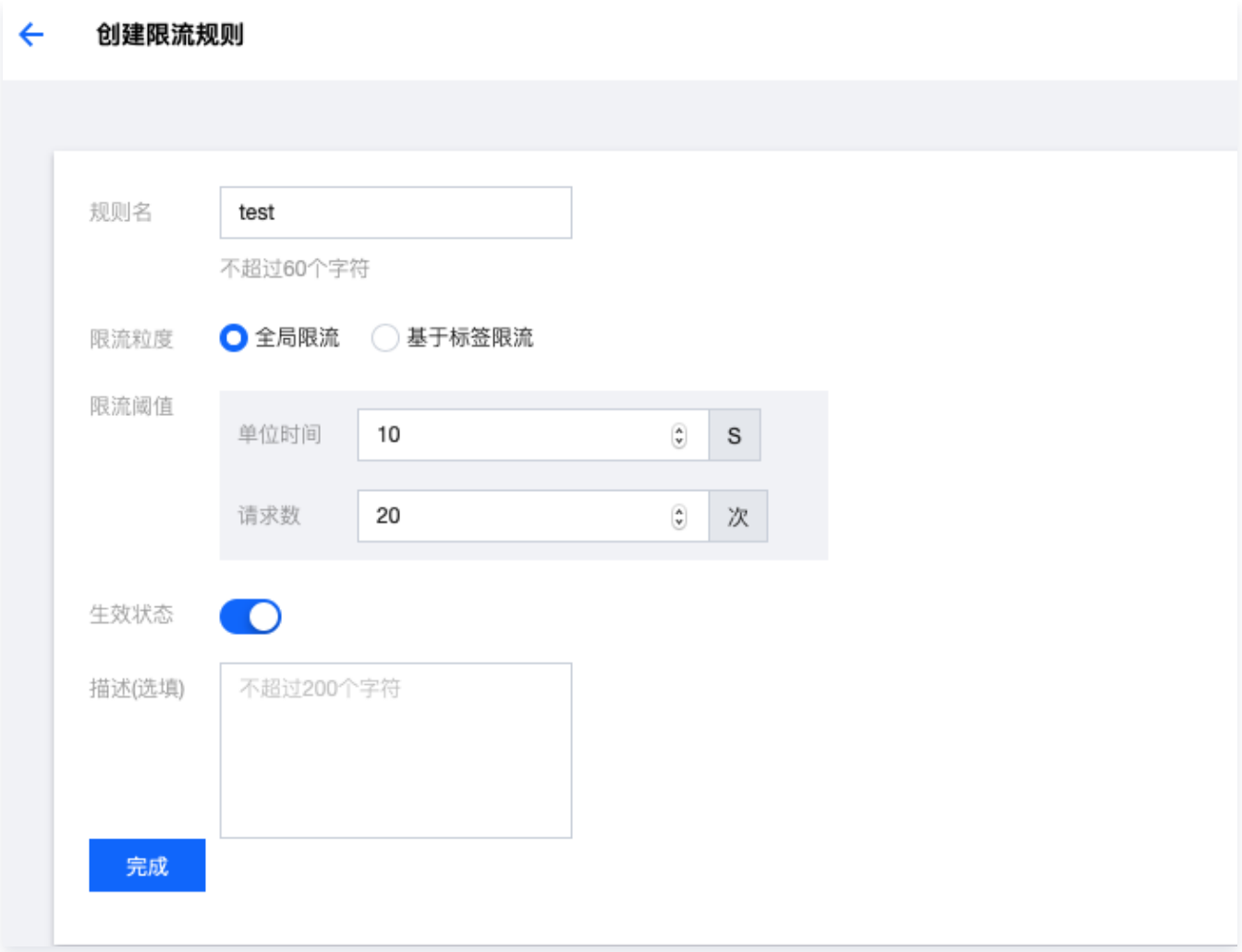
7. 登录 腾讯微服务平台控制台，在**依赖分析 > 服务依赖拓扑**页面，选择命名空间和时间后，可看到如下图的依赖关系。

步骤2：验证服务治理功能

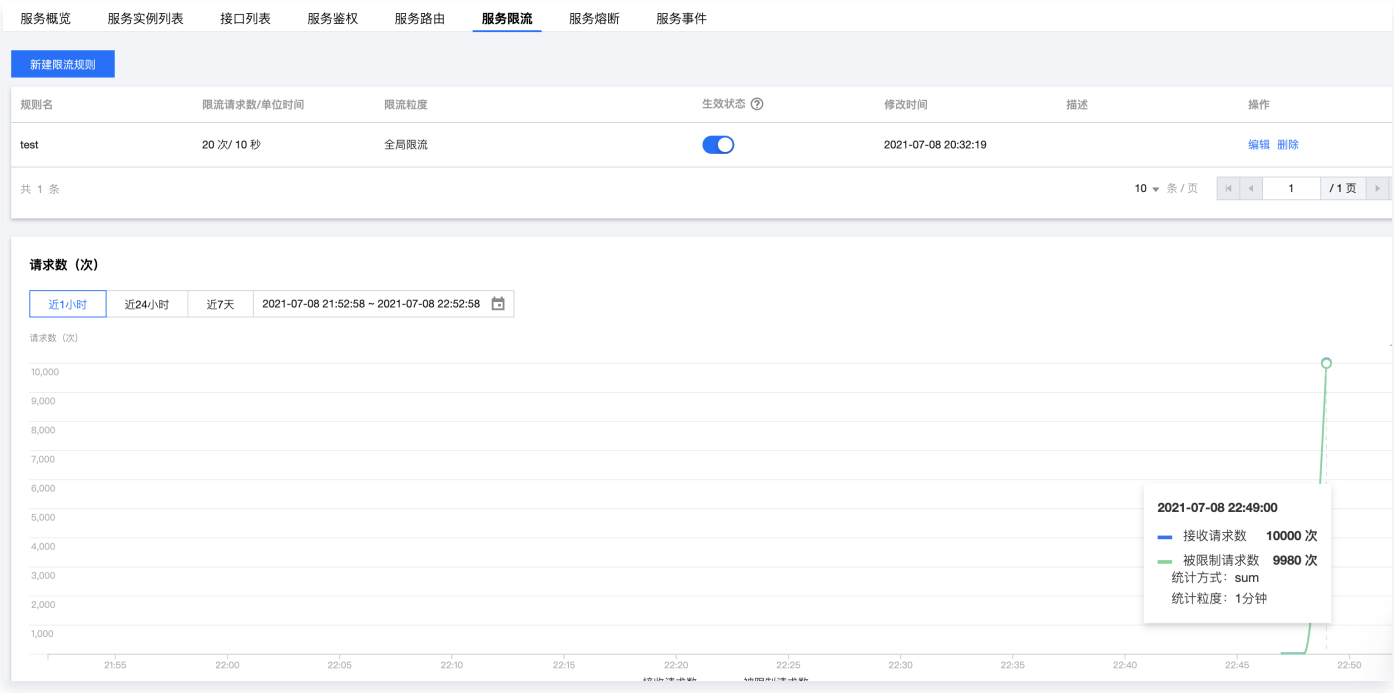
验证服务限流功能

服务限流详细介绍请参见 服务限流 。

- 典型业务问题：后端业务被高频恶意访问，导致核心业务链路阻塞，系统瘫痪。
- 场景：用户频繁访问拉取商品列表接口。
- 需求：保证核心服务 mall－admin 被每秒中最多被请求20次。
- 规则配置：在 腾讯微服务平台控制台 **> TSF Consul > 服务治理**页面找到 mall-admin 服务，进入服务详情页面，配置服务限流规则。



- **效果验证：**



验证服务鉴权功能

服务鉴权详细介绍请参见 服务鉴权 。

- 电商典型场景：后端敏感业务需要对访问权限进行控制。
- 场景：对于后台商品管理模块，仅支持有权限的服务对它进行访问。例如，在这个场景中，我们限制 gateway 微服务可以不访问 mall admin 微服务，所有从 gateway 发起的请求都会被拒绝。
- 配置方式：在 腾讯微服务平台控制台 > TSF Consul > 服务治理页面找到 mall-admin 服务，进入服务详情页面，配置服务鉴权规则。

- 效果验证：

自动化部署

当应用非常多，不希望使用控制台逐个部署怎么办呢？ 或者已经使用了 jenkins、travis 等工具，如何对接到 TSF 平台上呢？我们可以参见下面的操作来进行实践。

mall-demo 程序包 中的 deploy.py 脚本支持自动上传和部署一个新的应用到现有的集群中，默认选择集群中可用实例中的第一个实例机器部署应用。

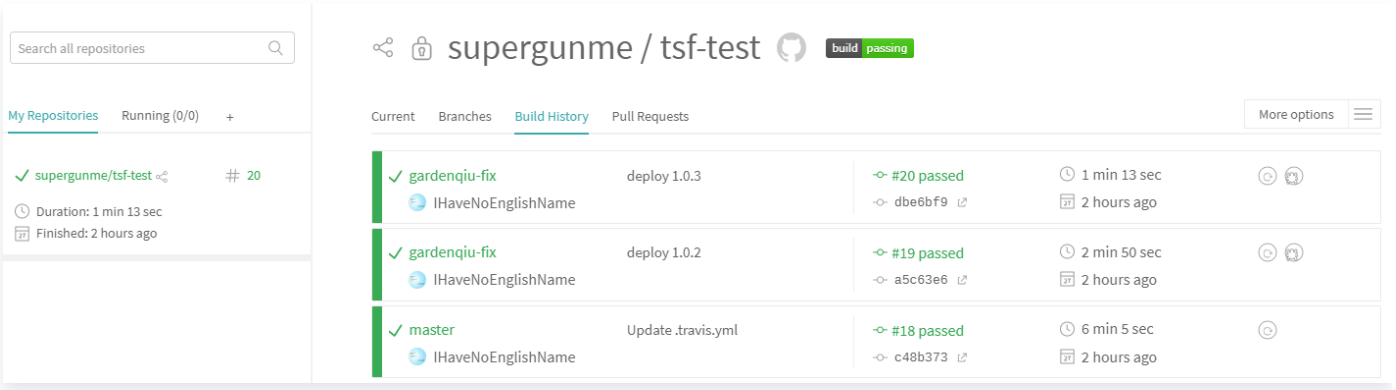
1. 在 deploy 目录下的 deploy.py 文件中配置 secret_id、secret_key，clusterId 和 namespace 等参数。

参数	是否必选	说明
path	必选	程序包路径。
applicationName	必选	应用名称。
appId	必选	账号 AppID。
groupName	可选	默认采用和应用名称同名，不可重复。
microserviceType	可选	默认“NATIVE” 云原生应用。否，填写 “N”。
applicationType	可选	默认“V”表示虚拟机部署。
pkgVersion	可选	上传的程序包版本号，默认当前时间戳，时间戳格式：“YYYYmmddHHMMSS”。

2. 在 travis.yml 中添加脚本任务和任务所需的执行参数。依次是：程序包路径、应用名和 APPID。

```
- ./scripts/deploy.py mall-demo/target/mall-demo-1.0-SNAPSHOT.jar "test"
"1234567890"
```

3. 提交 commit，并且推送到远程分支，自动触发 Travis CI 流程。Travis 流程执行成功。



Spring Cloud Alibaba 应用迁移

最近更新时间：2023-10-17 15:01:31

操作场景

应用迁移到 TSF 平台后即可享受平台一站式微服务解决方案：管理资源和应用更加便利、丰富的服务治理能力、多维度的监控和跨可用区高可用方案等。
本文介绍从 Spring Cloud Alibaba 迁移到 TSF 平台所需的改造工作，经过改造后，服务可以在 TSF 平台上成功注册和互相调用。现在就目前 Spring Cloud Alibaba 常见的 Restful 和 Dubbo 两类服务框架，分别说明改造方法。

Restful 服务改造

这部分改造主要包括删除 Nacos 依赖和配置、添加 TSF 的依赖和启动类的注解。

步骤1：依赖调整

1. 以下 Nacos 配置中心、注册中心、Spring Boot、Spring Cloud 的依赖，在迁移的时候，需要删除。

```
    <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
                    <groupId>com.alibaba.cloud</groupId>
                    <artifactId>spring-cloud-starter-alibaba-nacos-
config</artifactId>
    </dependency>
    <dependency>
                    <groupId>com.alibaba.cloud</groupId>
                    <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
    </dependency>
    <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- Spring Cloud Open Feign -->
    <dependency>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-starter-
openfeign</artifactId>
    </dependency>
```

2. 添加 TSF 依赖，TSF SDK 已经包含 Spring Boot、Spring Cloud 的依赖。

```
<!-- TSF 启动器 -->
<dependency>
                <groupId>com.tencent.tsf</groupId>
                <artifactId>spring-cloud-tsf-starter</artifactId>
                <version><!-- 调整为 SDK 版本号 --></version>
</dependency>
```

> **说明**
> SDK 版本请参见 Spring Cloud 概述-SDK版本使用说明 。

步骤2：配置调整

以下是 Nacos 服务注册中心和配置中心的配置项示例，在迁移时，要删除配置文件中这部分内容。

Nacos 服务注册中心

```
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
#spring.cloud.nacos.discovery.instance-enabled=true



spring.cloud.nacos.username=nacos
spring.cloud.nacos.password=nacos



management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

Nacos 配置中心

```
spring.cloud.nacos.config.server-addr=127.0.0.1:8848



#nacos certification information
spring.cloud.nacos.username=nacos
spring.cloud.nacos.password=nacos



#spring.cloud.nacos.config.refreshable-dataids=common.properties
#spring.cloud.nacos.config.shared-data-ids=common.properties,base-
common.properties
```

```
spring.cloud.nacos.config.shared-configs[0]= common333.properties
spring.cloud.nacos.config.shared-configs[1].data-id= common111.properties
spring.cloud.nacos.config.shared-configs[1].group= GROUP_APP1
spring.cloud.nacos.config.shared-configs[1].refresh= true
spring.cloud.nacos.config.shared-configs[2]= common222.properties



#spring.cloud.nacos.config.ext-config[0]=ext.properties
spring.cloud.nacos.config.extension-configs[0].data-
id= extension1.properties
spring.cloud.nacos.config.extension-configs[0].refresh= true
spring.cloud.nacos.config.extension-configs[1]= extension2.properties
spring.cloud.nacos.config.extension-configs[2].data-id= extension3.json
```

步骤3：代码调整

向应用启动类中添加 `@EnableTsf` 注解。

步骤4：应用部署

实现以上三步操作并且成功编译打包后，即完成改造，进入应用部署环节。TSF 平台支持多种应用类型，Restful 应用可以采用普通应用类型进行部署，更多信息请参见 虚拟机托管应用，按照文档指引操作即可。

Dubbo 服务改造

TSF 平台纳管 Spring Cloud Alibaba Dubbo 应用采用的是 TSF Mesh 技术，TSF Mesh 支持 Dubbo、HTTP 等协议，通过 Sidecar 注册到服务注册中心、处理服务间通信。
改造主要涉及两方面内容：

- 一个是 Mesh 化，引入 Sidecar。
- 另一个是移除原有注册中心的依赖和配置。TSF 兼容双注册，在不移除原有注册中心的情况下，经过少量代码调整，也可以成功部署应用，但是为了降低程序包的大小，加快应用部署和启动速度，建议要移除原有注册中心的依赖和配置。

步骤1：Mesh 改造

服务定义和注册（必选）
创建 spec.yaml 文件，定义服务名、暴露的端口和协议，服务名和接口名相同。由于 Dubbo 2 采用的是接口级服务发现，所以需要每个接口都要进行定义，在最新版的 Dubbo 3 中，采用了应用级服务发现，操作会简化很多。

```
apiVersion: v1
kind: Application
spec1:
  services:
  - name: com.dubbo.service.UserService
  ports:
```

```
    - targetPort: 20881
      protocol: dubbo
  healthCheck:
    path:
```

API 定义和上报（可选）

创建 apis 目录，该目录放置服务的 API 定义。一个服务对应一个 yaml 文件，文件名即服务名，API 遵循 OPENAPI 3.0 规范。

```
openapi: 3.0.0
info:
version: "1.0.0"
title: user service
paths:
/api/v6/user/create:
  get:
    responses:
      '200':
        description: OK
      '401':
        description: Unauthorized
      '402':
        description: Payment Required
      '403':
        description: Forbidden
/api/v6/user/account/query:
  get:
    responses:
      '200':
        description: OK
      '401':
        description: Unauthorized
      '402':
        description: Payment Required
      '403':
        description: Forbidden
/health:
  get:
    responses:
      '200':
        description: OK
      '401':
        description: Unauthorized
      '402':
        description: Payment Required
```

```
      '403':
        description: Forbidden
```

调用链 Header 传递（可选）

要实现 Mesh 应用调用链和服务依赖拓扑功能，需要在请求中带上9个相关 header。

```
// 9个调用链相关的头，具体说明
(https://www.envoyproxy.io/docs/envoy/v1.8.0/configuration/http_conn_man/hea
ders.html?highlight=tracing)
traceHeaders = ['x-request-id',
                    'x-trace-service',
                    'x-ot-span-context',
                    'x-client-trace-id',
                    'x-b3-traceid',
                      'x-b3-spanid',
                    'x-b3-parentspanid',
                    'x-b3-sampled',
                      'x-b3-flags']
```

参见文档： 开发使用指引

步骤2：依赖调整

删除 Nacos 服务注册和配置的依赖，如下例：

```
<dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
config</artifactId>
</dependency>
<dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
</dependency>
```

步骤3：配置调整

关闭 Dubbo 服务注册中心配置，如下例：

```
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
#spring.cloud.nacos.discovery.instance-enabled=true
```

```
spring.cloud.nacos.username=nacos
spring.cloud.nacos.password=nacos



management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

删除 Nacos 配置中心的配置，如下例：

```
spring.cloud.nacos.config.server-addr=127.0.0.1:8848



#nacos certification information
spring.cloud.nacos.username=nacos
spring.cloud.nacos.password=nacos



#spring.cloud.nacos.config.refreshable-dataids=common.properties
#spring.cloud.nacos.config.shared-data-ids=common.properties,base-
common.properties
spring.cloud.nacos.config.shared-configs[0]= common333.properties
spring.cloud.nacos.config.shared-configs[1].data-id= common111.properties
spring.cloud.nacos.config.shared-configs[1].group= GROUP_APP1
spring.cloud.nacos.config.shared-configs[1].refresh= true
spring.cloud.nacos.config.shared-configs[2]= common222.properties



#spring.cloud.nacos.config.ext-config[0]=ext.properties
spring.cloud.nacos.config.extension-configs[0].data-
id= extension1.properties
spring.cloud.nacos.config.extension-configs[0].refresh= true
spring.cloud.nacos.config.extension-configs[1]= extension2.properties
spring.cloud.nacos.config.extension-configs[2].data-id= extension3.json
```

删除 Hystrix 配置：

```
feign:
  hystrix:
    enabled: true
```

步骤4：代码调整

1. 删除服务发现的注解 `@EnableDiscoveryClient` 。
2. 服务消费端的 `@Reference` 注解添加 url 属性，属性值为 `dubbo：//服务名:端口` ，添加 `check=false` 属性。在兼容双注册的场景下，这个代码需要额外添加。

步骤5：应用部署

代码改造完并且成功编译打包后，进入应用部署环节。在 TSF 平台创建 Mesh 应用类型进行部署，更多信息请参见 虚拟机托管应用 ，按照文档指引操作即可。