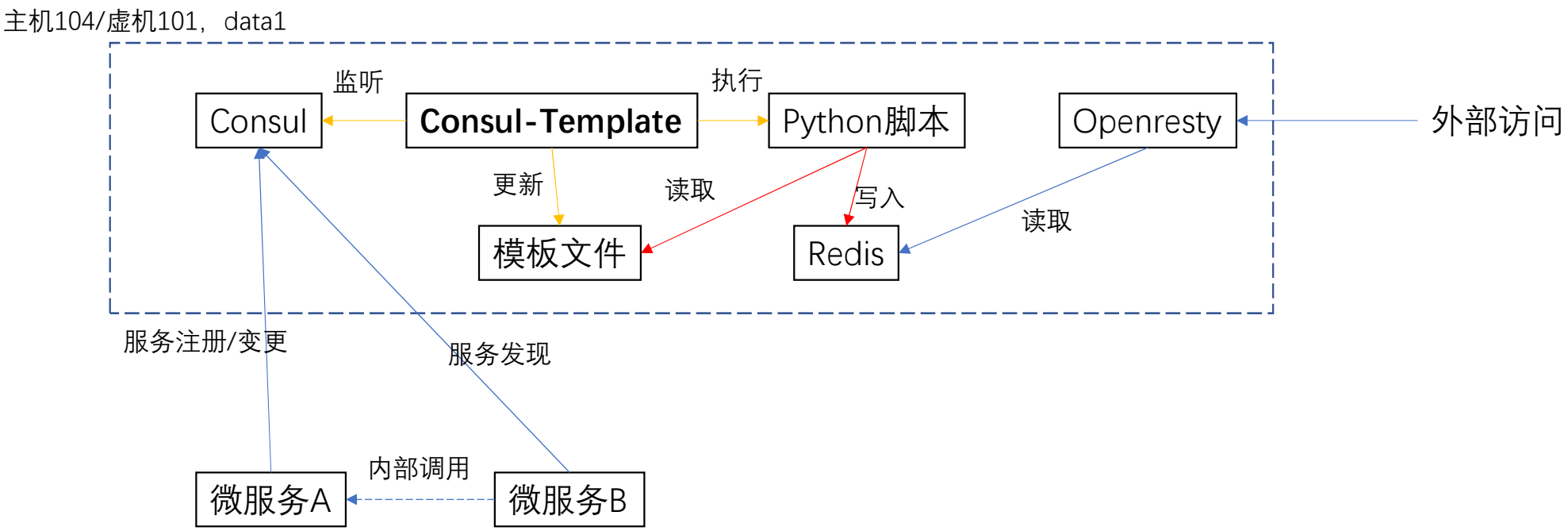
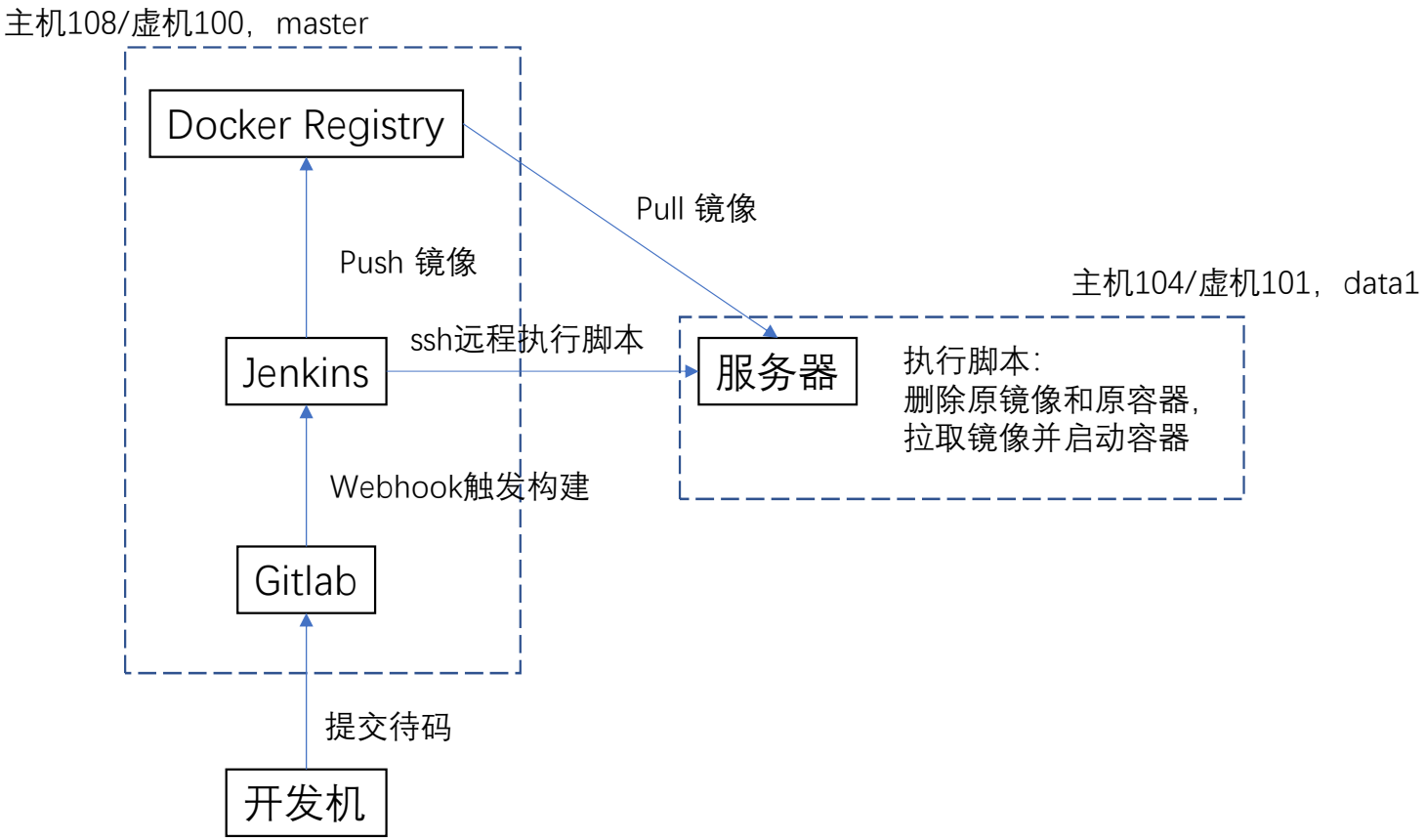
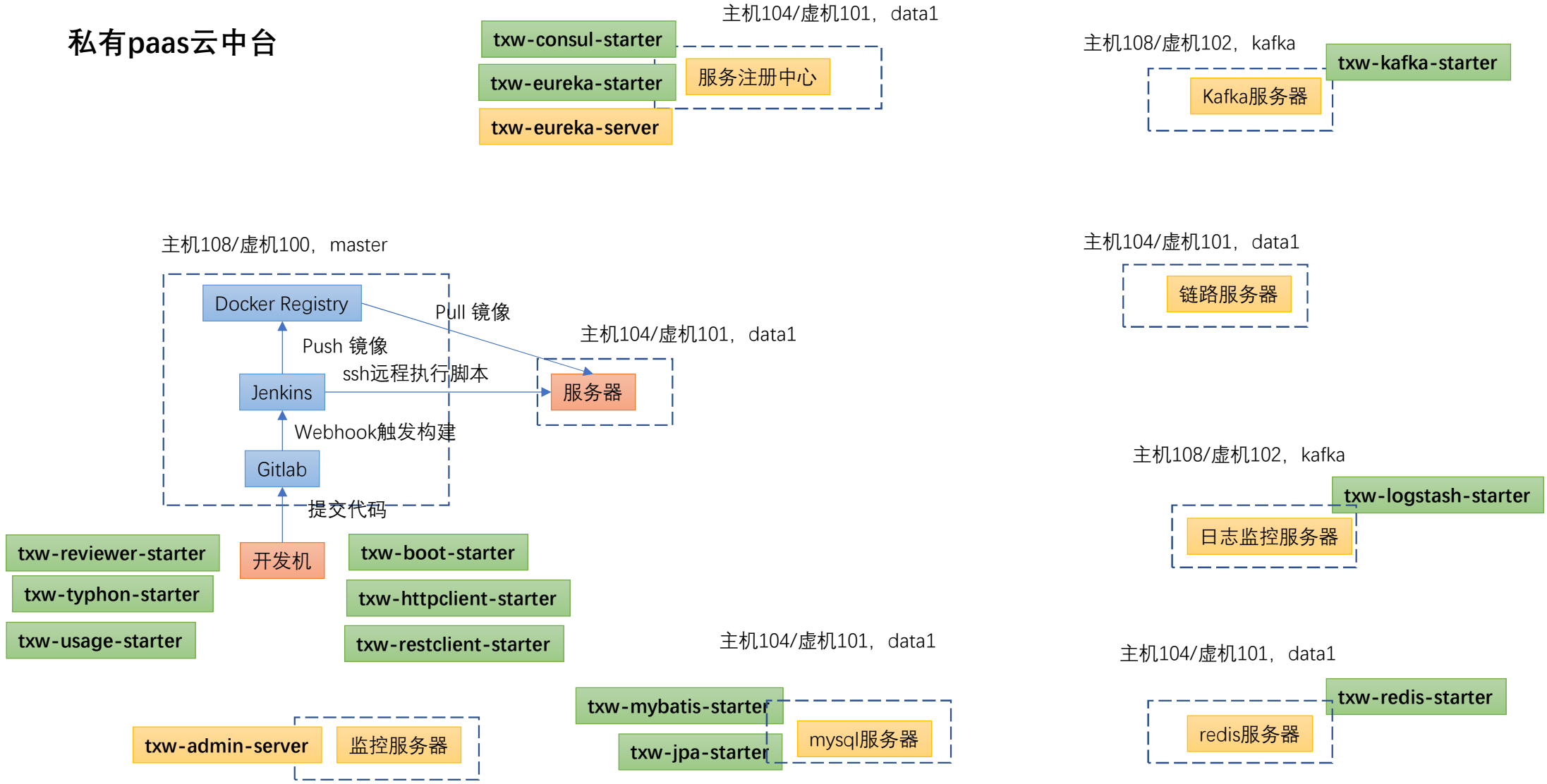
服务注册和发现


主机104/虚机101，data1
监听
执行
Consul
Consul-Template
Python脚本
Openresty
外部访问
更新
读取
写入
读取
模板文件
Redis
服务注册/变更
服务发现
内部调用
微服务A
微服务B

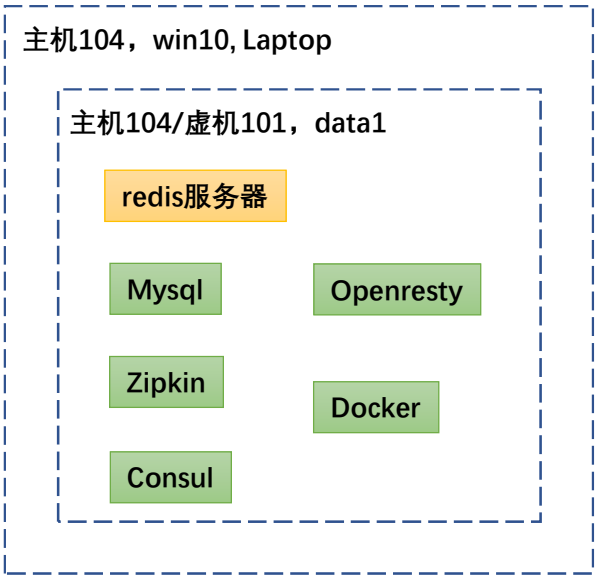
持续集成和部署

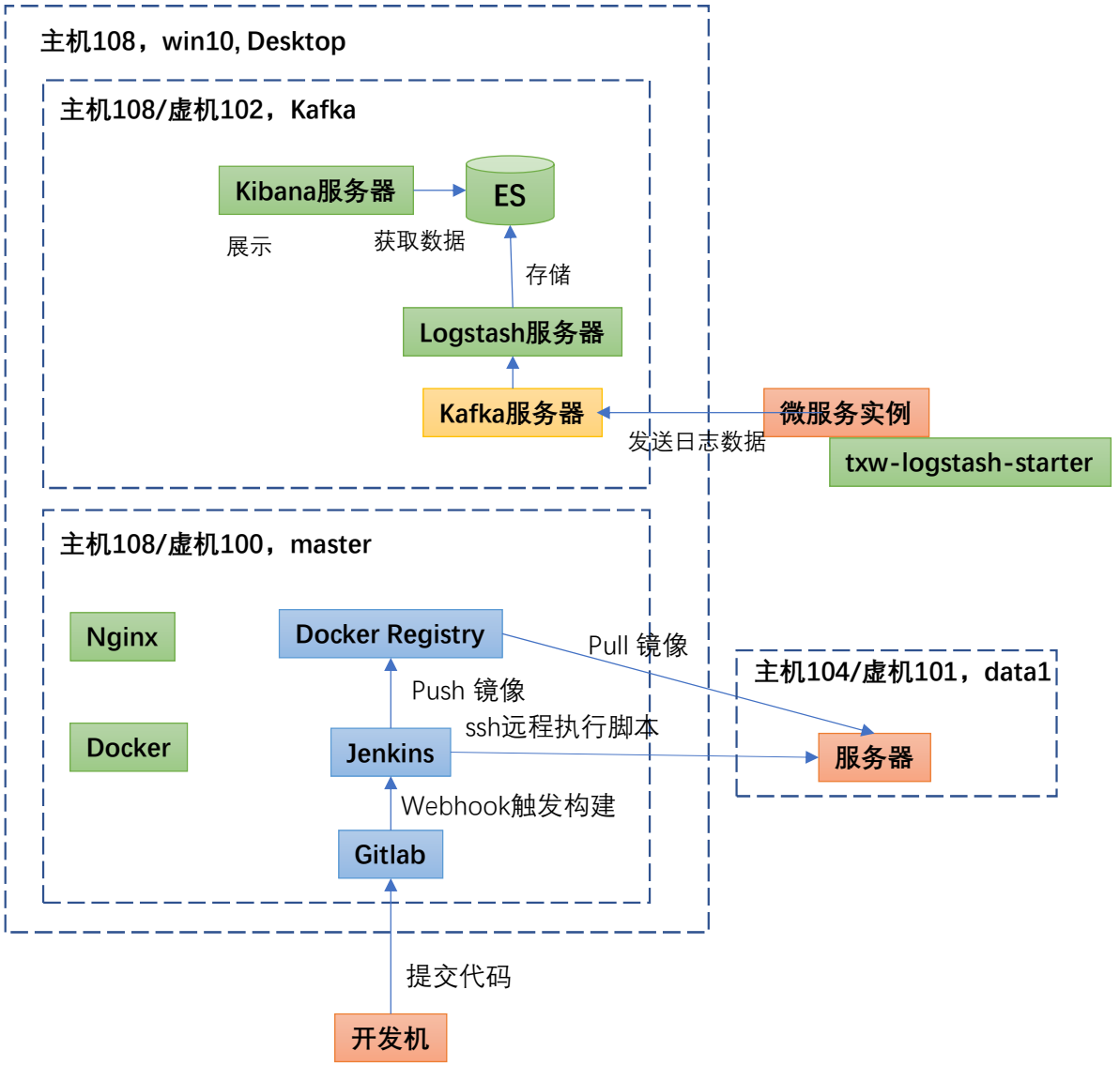

主机108/虚机100，master
Docker Registry
Pull 镜像
Push 镜像
主机104/虚机101，data1
Jenkins
ssh远程执行脚本
服务器
执行脚本:
删除原镜像和原容器，
拉取镜像并启动容器
Webhook触发构建
Gitlab
提交待码
开发机

私有paas云中台


主机104/虚机101，data1
txw-consul-starter
txw-eureka-starter
txw-eureka-server
服务注册中心
主机108/虚机102，kafka
txw-kafka-starter
Kafka服务器
主机108/虚机100，master
Docker Registry
Pull 镜像
Push 镜像
ssh远程执行脚本
Jenkins
Webhook触发构建
Gitlab
提交代码
开发机
主机104/虚机101，data1
服务器
主机104/虚机101，data1
链路服务器
主机108/虚机102，kafka
txw-logstash-starter
日志监控服务器
txw-reviewer-starter
txw-typhon-starter
txw-usage-starter
txw-boot-starter
txw-httpclient-starter
txw-restclient-starter
主机104/虚机101，data1
txw-mybatis-starter
txw-jpa-starter
mysql服务器
主机104/虚机101，data1
txw-redis-starter
redis服务器
txw-admin-server
监控服务器

物理架构





说明：

开发环境：主机104 & 虚机101

生产环境：主机108/虚机100,101,102

1、平时使用开发环境，只需要启动虚机101。

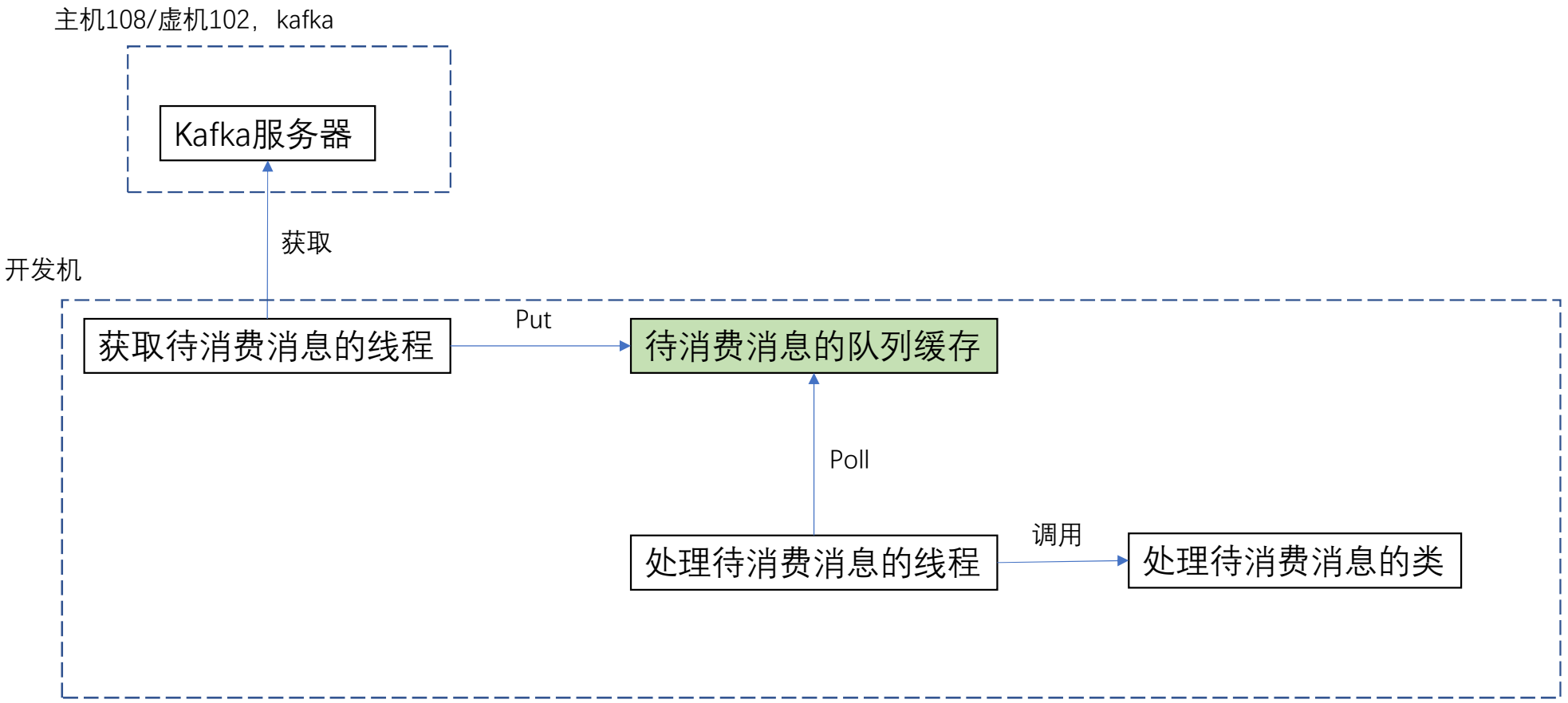
2、如果需要使用持续集成和构建，需要启动主机108和虚机100。

3、如果需要使用Kafka，需要启动主机108和虚机102。

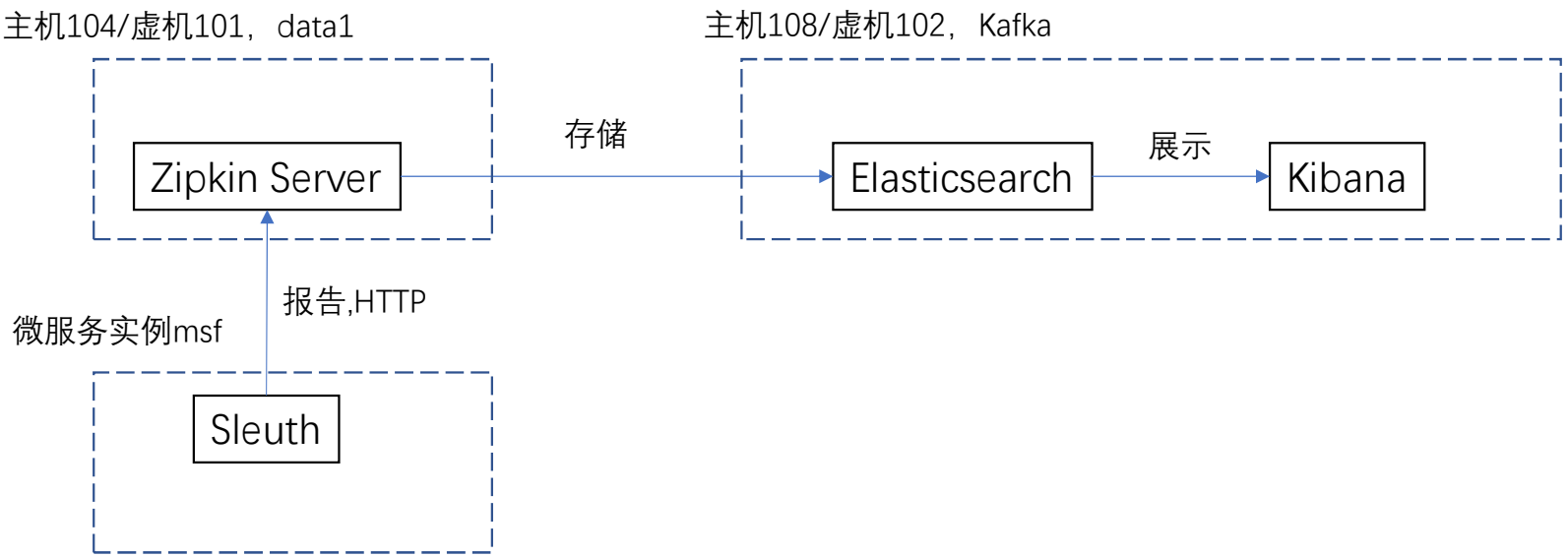
4、完整的生产环境和演示环境，需要启动主机108，虚机100,101和102。

并且开启路由器的虚拟IP映射和虚机的NAT的端口映射。

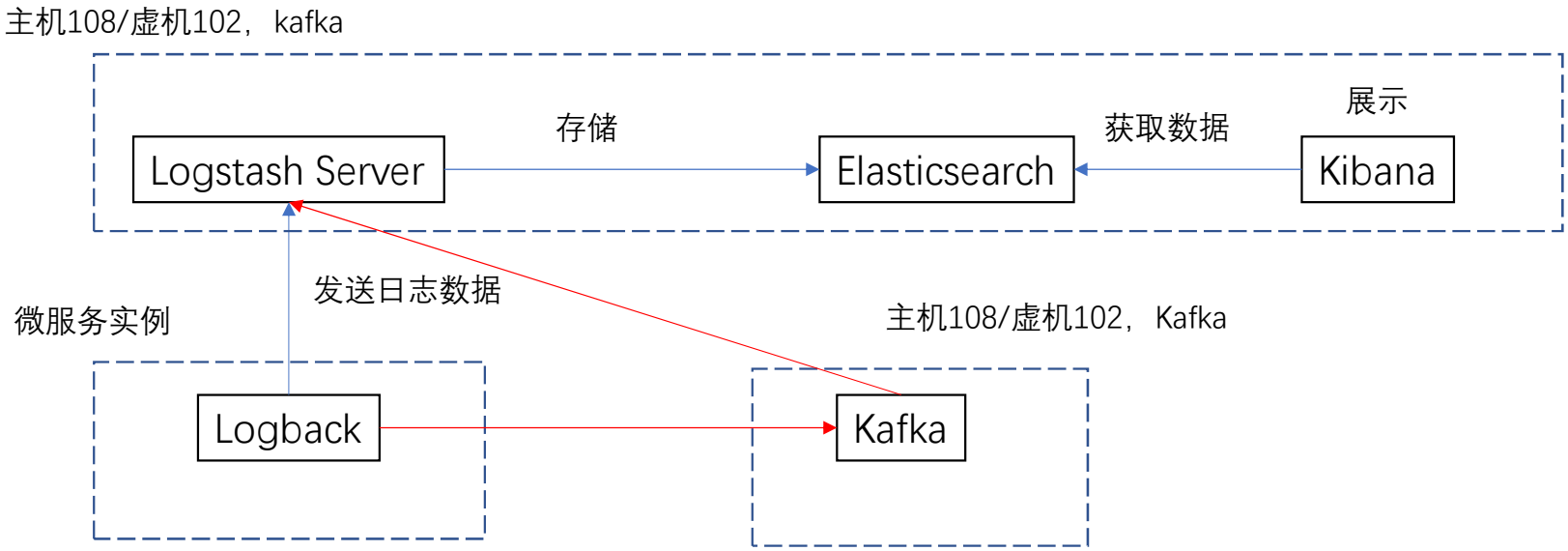
消息队列


主机108/虚机102，kafka
Kafka服务器
获取
开发机
获取待消费消息的线程
Put
待消费消息的队列缓存
Poll
处理待消费消息的线程
调用
处理待消费消息的类

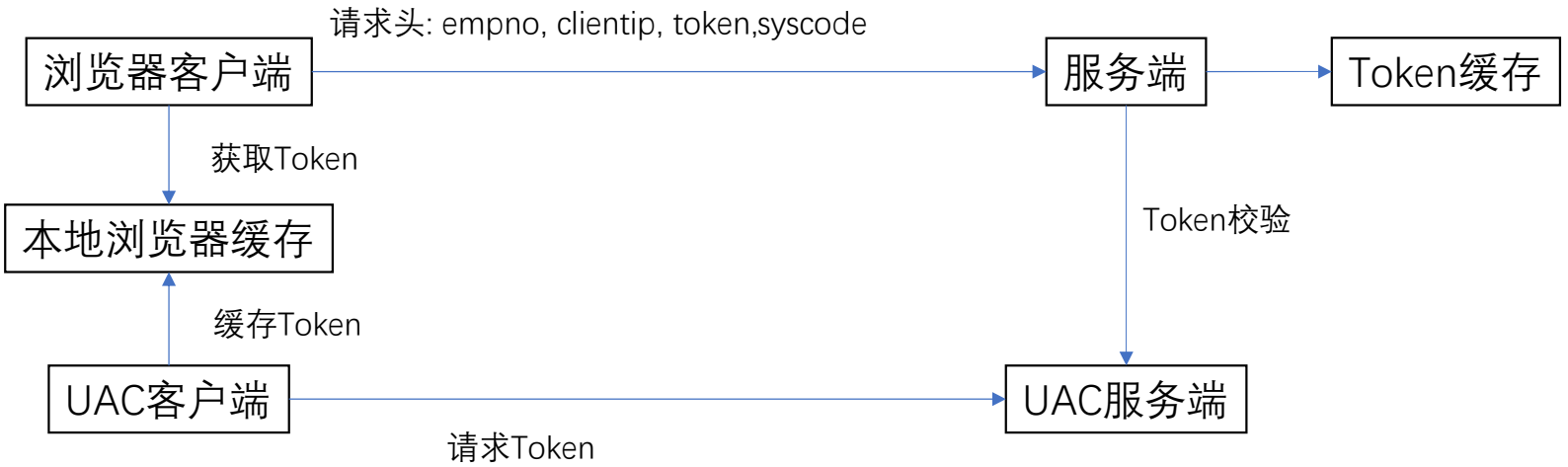
链路追踪


主机104/虚机101，data1
主机108/虚机102，Kafka
Zipkin Server
存储
Elasticsearch
展示
Kibana
报告,HTTP
微服务实例msf
Sleuth

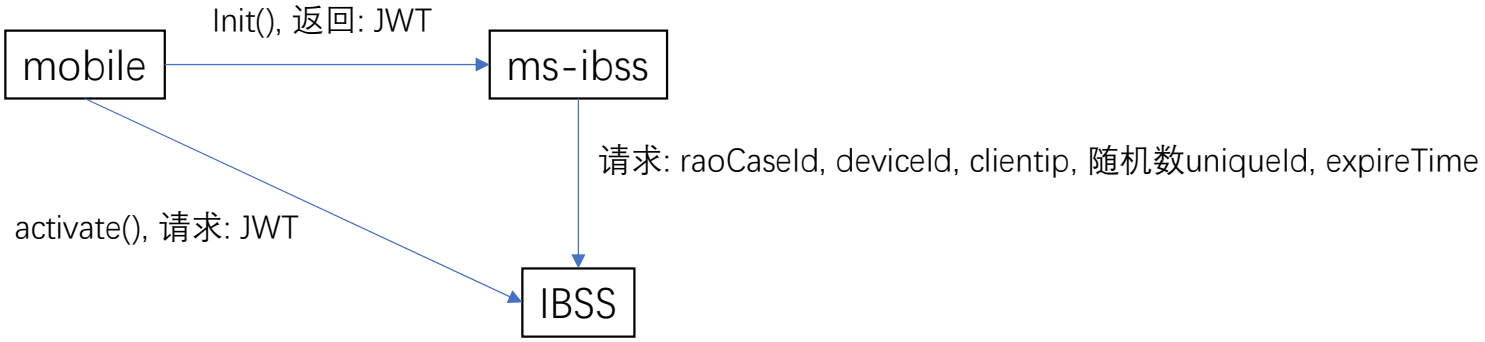
日志监控


主机108/虚机102，kafka
Logstash Server
存储
Elasticsearch
获取数据
展示
Kibana
发送日志数据
微服务实例
主机108/虚机102，Kafka
Logback
Kafka

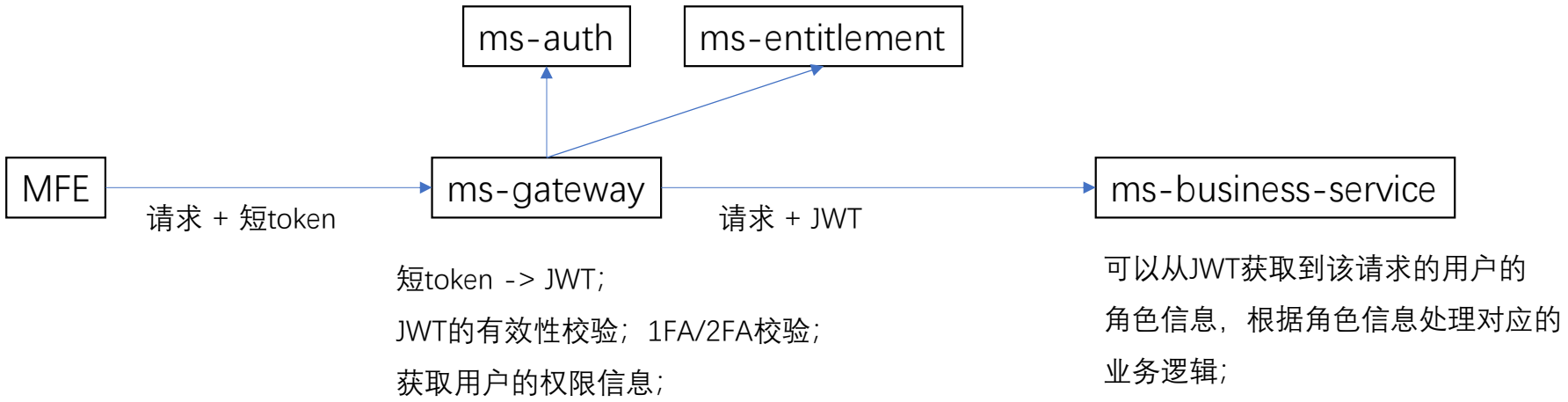
Token认证 – 中兴UAC方案



Token认证 – 华侨永亨IBSS方案



Token认证 – OCBC方案

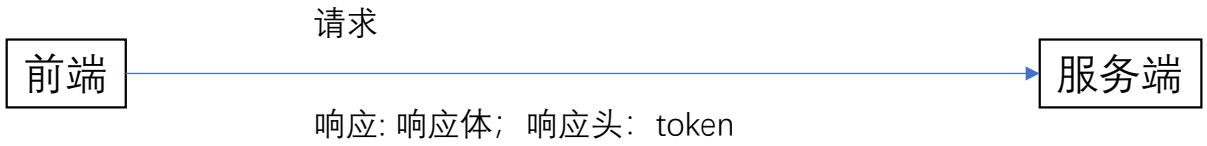


优点：gateway统一负责路由转发，token的校验，鉴权。

问题：

每一次客户端请求资源，都需要通过gateway的校验和鉴权。

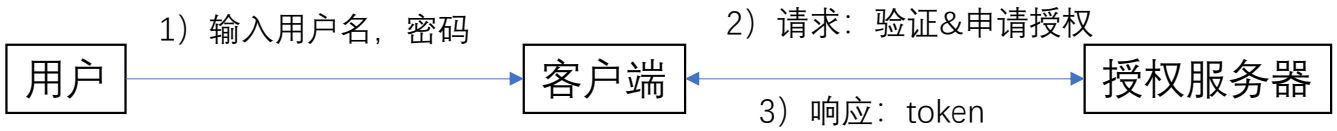
gateway容易成为性能瓶颈。而且没有必要（每一次）



1、服务端对响应体用加密算法A加密得到token，返回给前端；

2、前端用解密算法B对token解密，得到响应体，校验响应体的数据一致性；

Spring Security



Spring Security已经完成了身份验证和授权的功能。解决了保护自己的问题。

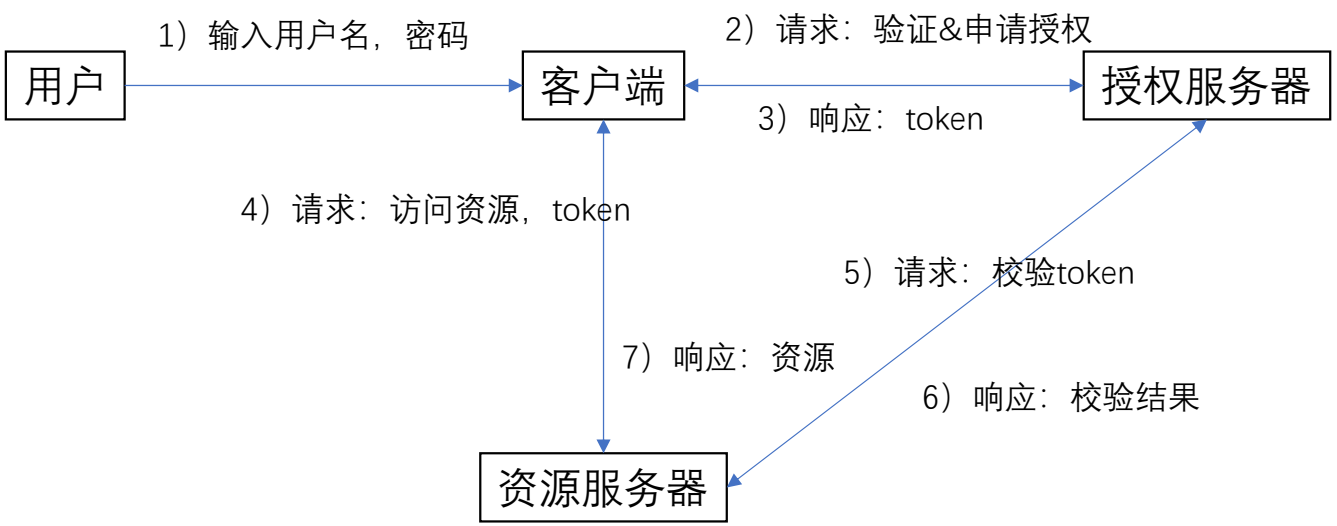
包括配置用户的身份信息，需要鉴权的资源范围和权限。可以开启方法级别的保护。

并返回身份信息JSessionID。

问题：

这种sessionID的cookie的方案对于微服务多实例的分布式环境，需要解决如何共享session的问题。

Spring Security + OAuth2



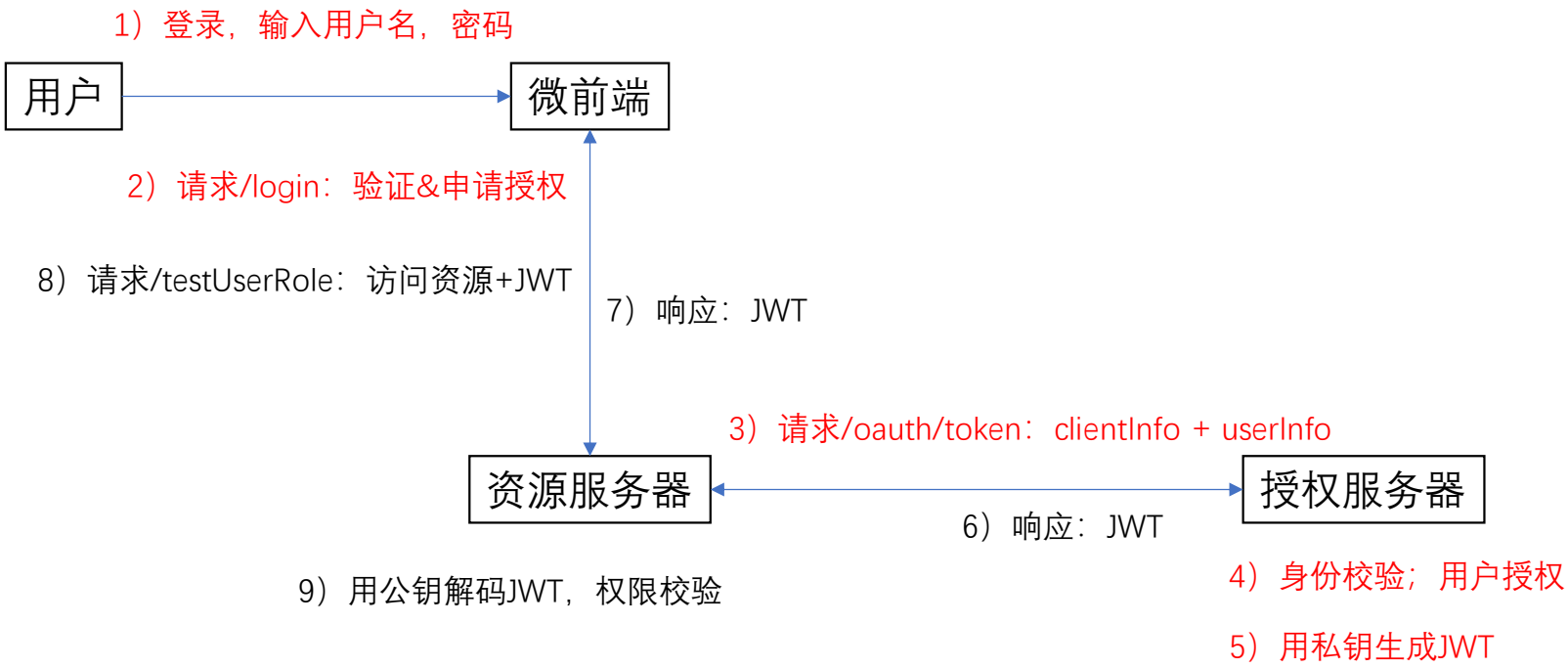
OAUTH2解决的是微服务环境下，提供统一的身份验证和授权服务，解决了保护别人的问题。

授权服务器负责身份验证，生成校验token，鉴权。

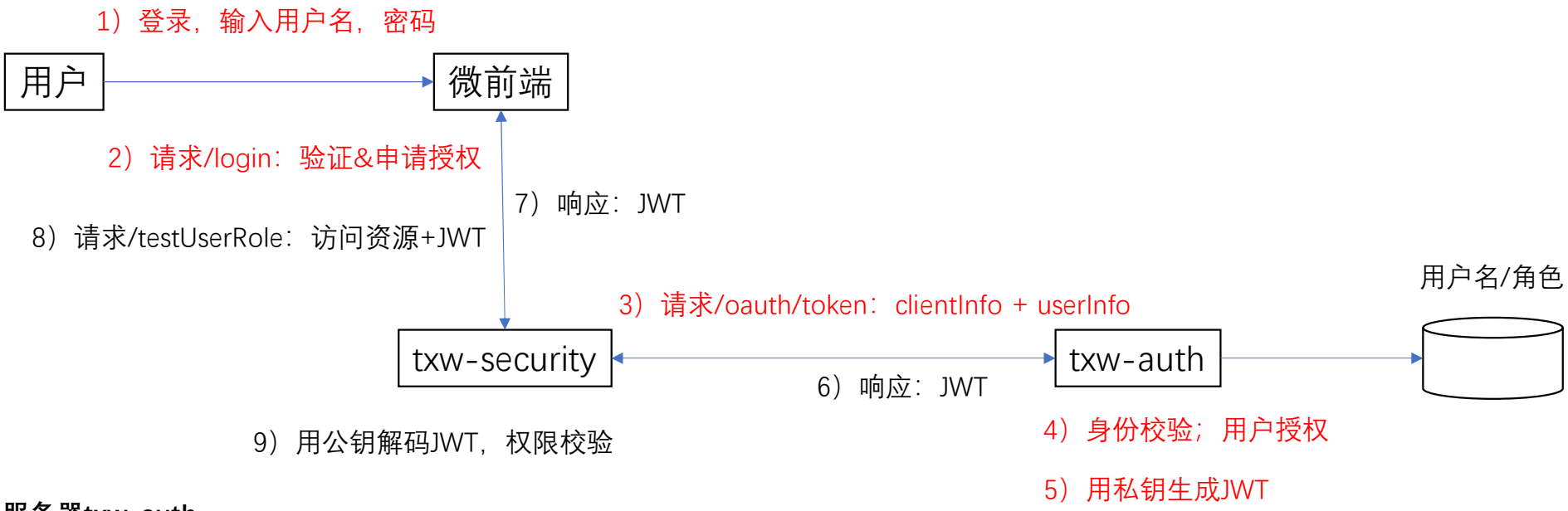
问题：

每一次客户端请求资源，资源服务器都需要请求授权服务器鉴权。鉴权服务器容易成为瓶颈。

Spring Security + OAuth2 + JWT



Spring Security + OAuth2



1. 授权服务器txw-auth

集成了spring-boot-starter-security和spring-cloud-starter-oauth2

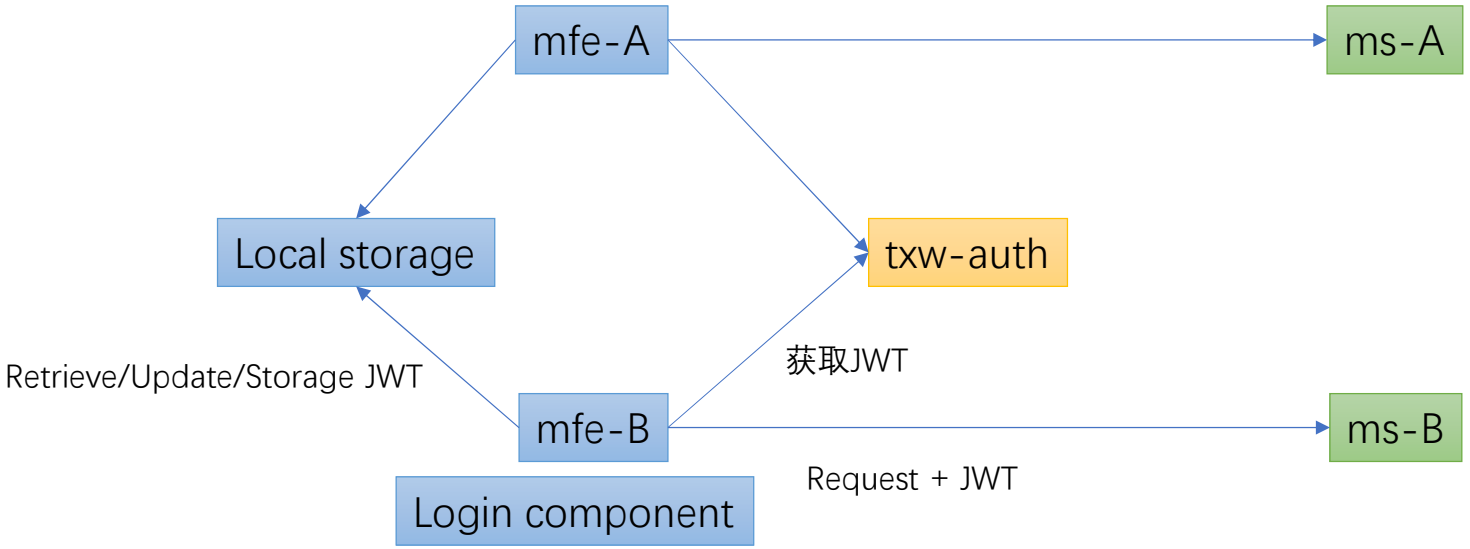
负责身份验证，生成token（token包含了用户的身份和角色信息）。

问题：需要增加接口，方便管理用户和角色信息。

2. 资源服务器txw-security

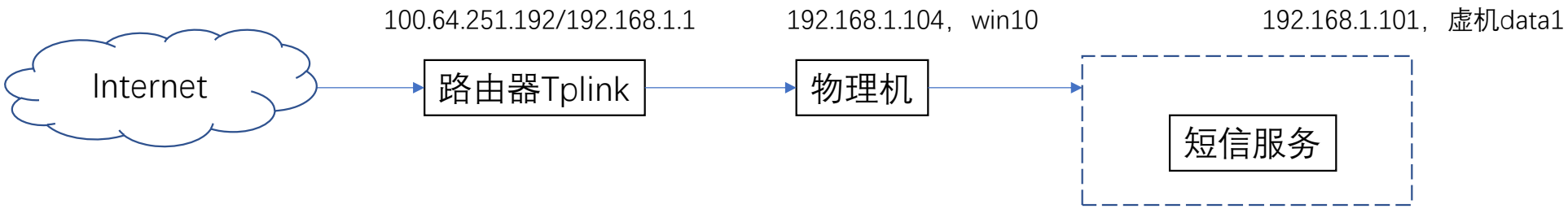
集成了spring-boot-starter-security

需要配置资源（接口）需要的角色。比如@PreAuthorize("hasAuthority('USER') or hasAnyRole('USER')")



JWT包括用户的所有权限点
包括Role_User, ms-A-Role_User, ms-B-Role_User

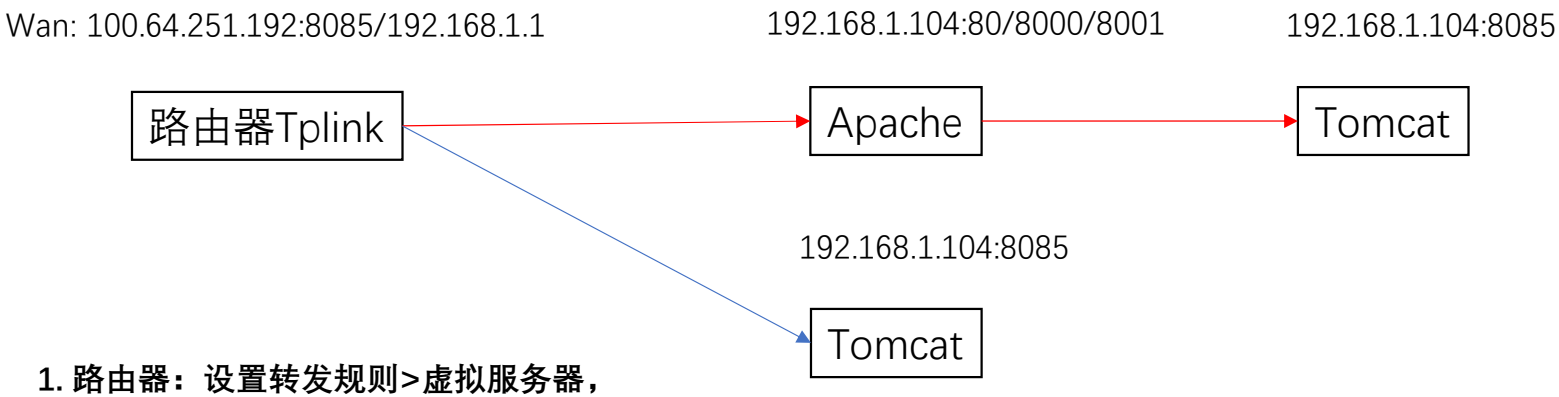
外网访问物理机/虚拟机



路由器：设置转发规则>虚拟服务器，**100.64.98.158**:8505 -> 192.168.1.104:8505

虚拟机：设置端口转发规则，192.168.1.104:8505 -> 192.168.1.101:8505

路由器：运行状态，查看IP地址



1. 路由器：设置转发规则>虚拟服务器，

100.64.251.192**:8001 -> 192.168.1.104:8001**

100.64.251.192**:8085** -> 192.168.1.104:8085

http://100.64.251.192:8001/swagger-ui.html

http://100.64.251.192:8001/test.html

http://100.64.251.192:8085/swagger-ui.html

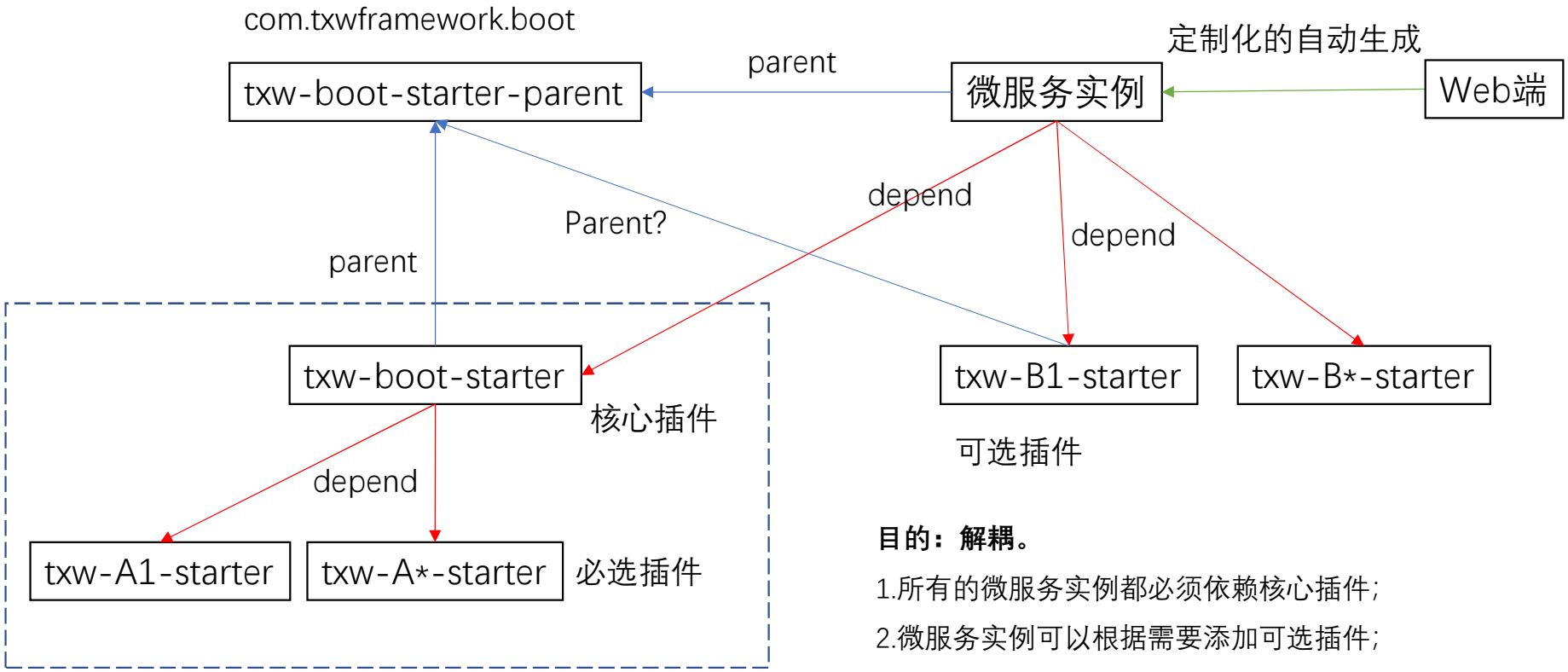
2. Apache：设置反向代理规则，

192.168.1.104:8001 -> 192.168.1.104:8085

http://localhost:8001/test.html

http://192.168.1.104:8001/swagger-ui.html

微服务框架拆解设计



目的：解耦。

1.所有的微服务实例都必须依赖核心插件；

2.微服务实例可以根据需要添加可选插件；

3.核心插件包括了必选插件，实现了常用工具类等；

4.核心插件和可选插件都可以独自迭代和发布版本；统一的版本管理；

5.微服务实例可以选择插件的某个版本进行依赖。

1.　可选插件txw-B*-starter

对开源组件做了定制化的二次封装，对于微服务实例而言是可选的插件。这样降低了耦合。

按照最小知识原则，高内聚，低耦合，按照需求，独立的迭代和发布版本；

比如。。。

2. 必选插件txw-A*-starter

对开源组件做了定制化的二次封装，对于微服务实例而言是必选的插件。

被txw-boot-starter在pom中声明了依赖关系。

在txw-boot-starter中往往有相关的代码。对微服务实例而言，代码不可见，有更好的封装性。

如果对于不同的微服务实例，需求不同，代码和配置不同，则不适合作为必选插件。

3. **核心插件txw-boot-starter**

目的是为了让所有的依赖txw-boot-starter的微服务实例，具有某种**统一性和强制性**。

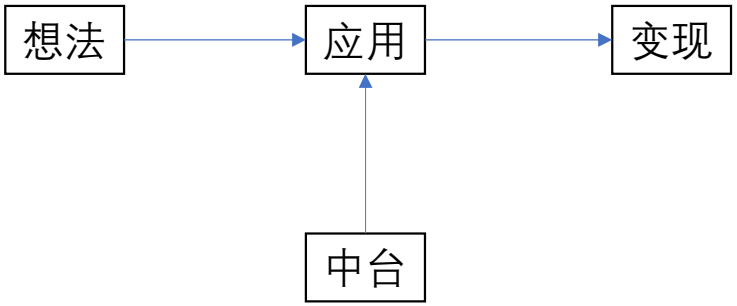
而**可扩展性**通过插件的配置文件和配置项实现。

多个必选插件和txw-boot-starter关联在一起，有更强的耦合关系。比如服务注册，国际化等。

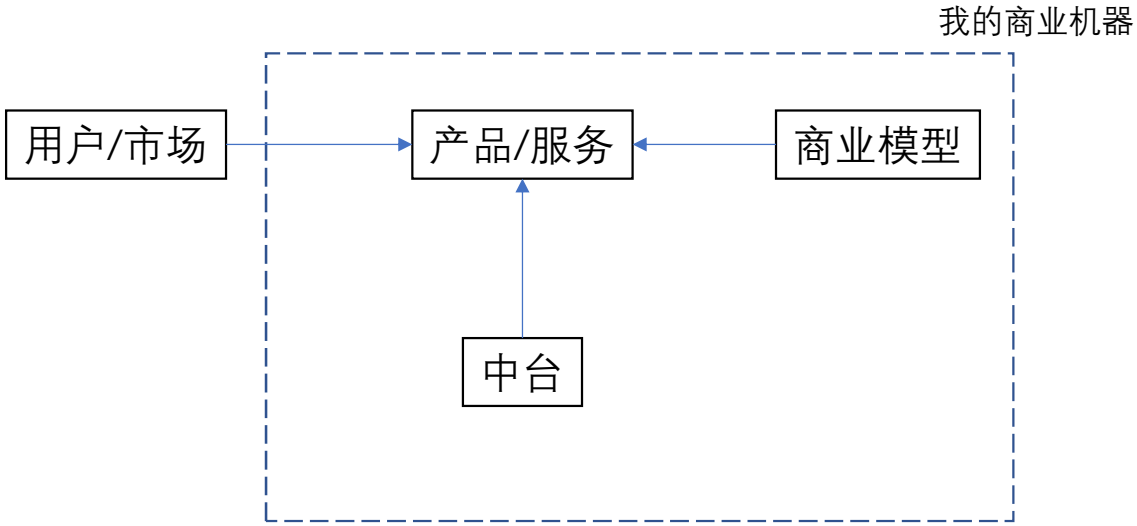
4. txw-boot-starter-parent

目的：当以上插件包括微服务实例存在一起公有的依赖项，而且需要统一的版本管理时，为依赖项的版本的统一管理提供方便。

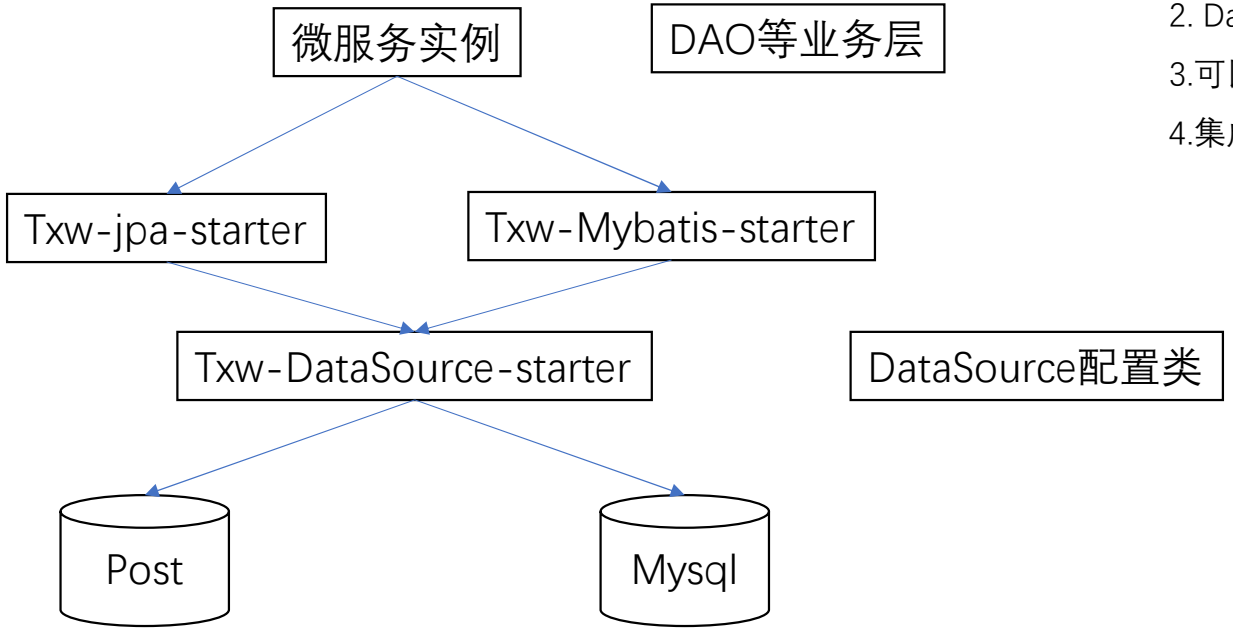
SuperCell



我想要的



数据持久化



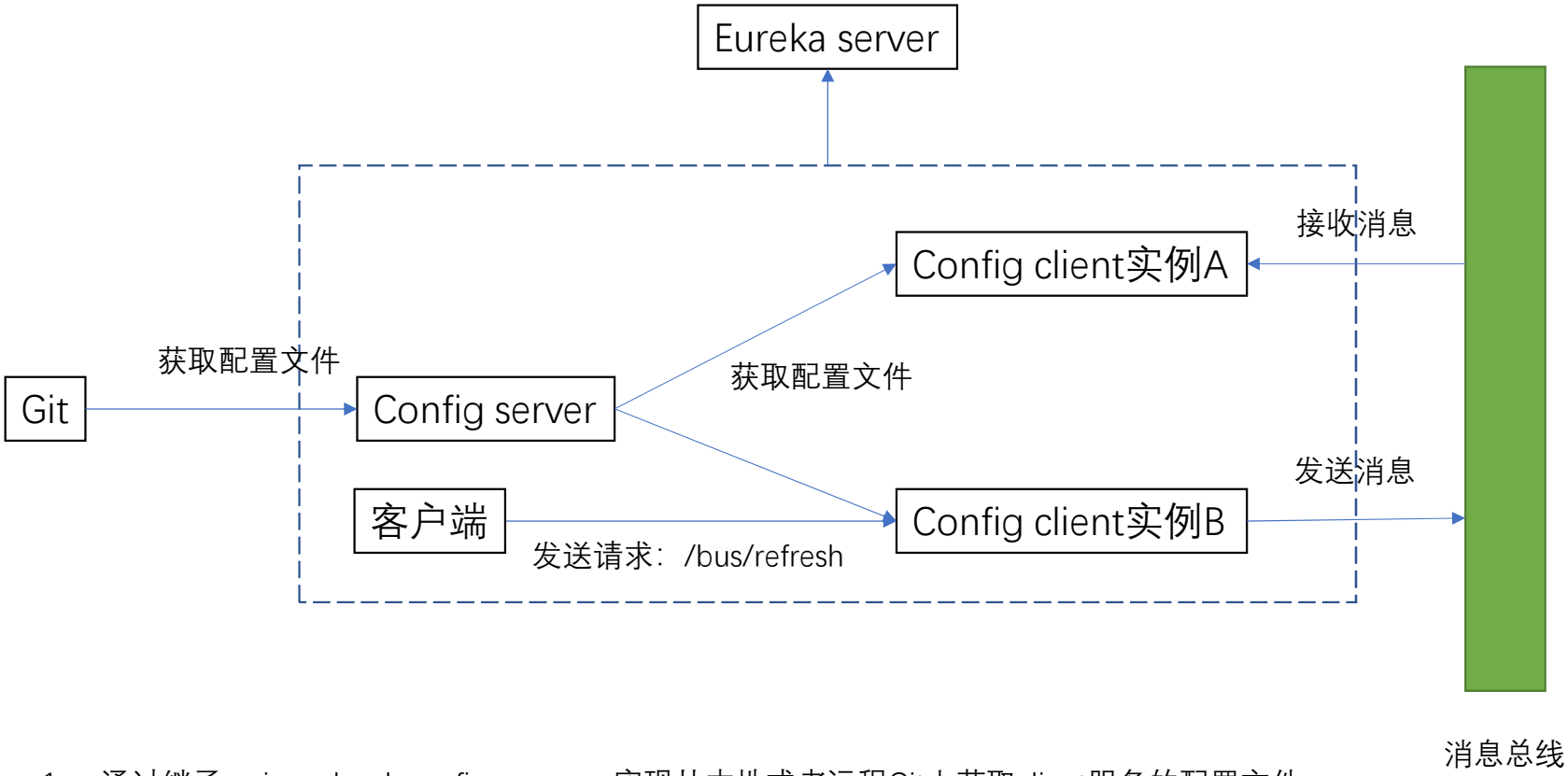
1.可以适应不同的数据库，比如Mysql，Post；

2. DataSource类可以配置；？

3.可以选择不同的持久化层；

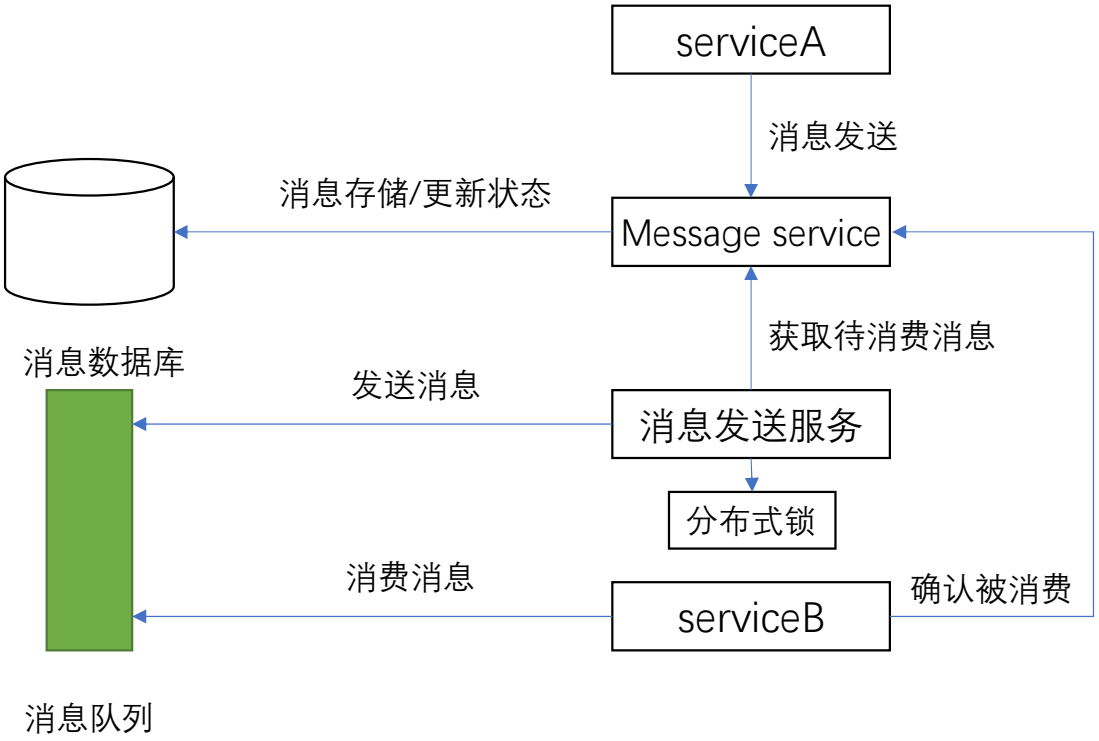
4.集成方实现Dao层及以上的业务层；

Spring Cloud 配置中心



1. 通过继承spring-cloud-config-server，实现从本地或者远程Git上获取client服务的配置文件；
2. Client服务通过集成spring-cloud-starter-config，实现从config server获取不同环境的配置文件；
3. 通过在client服务集成spring-cloud-starter-bus-amqp，实现实时更新服务的所有实例；

分布式事务 - 基于消息服务实现最终一致性



1、**serviceA**修改数据，如果正常则调用**Message service**的**消息发送接口**，否则则回滚；

2、Message service把消息存储到**数据库**，状态：待消费；

3、**消息发送服务**定时调用**Message service**的**获取待消费消息接口**，把消息发送到**消息队列**，调用**Message service**的**发送次数加一接口，**更新消息的发送次数；

如果该消息满足死亡条件，则调用**Message service**的**确认死亡接口**，更新消息状态：死亡；

4、**serviceB**从**消息队列**获取消息，如果处理成功，则调用**Message service**的**确认被消费接口**，**Message service**更新消息状态：已消费；~~并手动确认MQ的消费；~~

当发生异常，则没有~~手动~~确认MQ的消费，这时~~MQ~~会重新发送，直到重发上限。

注：

通过Message service持久化消息获得可靠性。

通过分布式锁，保证在同一时刻只有一个任务程序分发消息，也可以通过分布式任务调度的框架。