讲师(老司机)

项目实战-容器化部署

课程内容介绍

开发篇

实战篇

- lagou-bom项目容器化部署
- mysql数据库容器化部署
- ELK容器化部署：官网ELK部署、企业级应用sebp/elk镜像部署
- 常用开发工具：idea2020.1、JDK1.8.221、maven3.6.2、VMware15、Xshell5、XFTP5、centos7.8

压测篇

- jmeter简介
- jmeter概述
- jmeter常用元件
- jmeter压测微服项目
- jmeter压测数据库
- jmeter版本：apache-jmeter-5.3

运维篇

- rancher-server安装
- rancher部署lagou-bom项目
- rancher高级应用
- rancher部署redis
- rancher部署rabbitmq
- prometheus容器化部署
- 自定义prometheus镜像
- cadvisor容器化部署
- grafana容器化部署
- 使用rancher应用商店快速部署监控系统

节点信息

服务器用户名：root，服务器密码：123456

主机名	IP地址	说明
rancher-server-120	192.168.198.120	rancher-server服务器
rancher-agent-121	192.168.198.121	1.需要内存大。推荐10G内存 2.部署微服项目的工作节点

基础镜像汇总

序号	镜像名称
1	openjdk:8-alpine3.9
2	mysql:5.7.31
3	docker.elastic.co/logstash/logstash:7.7.0
4	docker.elastic.co/kibana/kibana:7.7.0
5	docker.elastic.co/elasticsearch/elasticsearch:7.7.0
6	sebp/elk:770
7	rancher/server:v1.6.30
8	rancher/agent:v1.2.11
9	rancher/scheduler:v0.8.6
10	rancher/net:v0.13.17
11	rancher/dns:v0.17.4
12	rancher/healthcheck:v0.3.8
13	rancher/metadata:v0.10.4
14	rancher/network-manager:v0.7.22
15	rancher/storage-nfs:v0.9.1
16	rancher/net:holder
17	rabbitmq:management-alpine
18	redis:6.0.8-alpine3.12
19	prom/prometheus:v2.21.0
20	google/cadvisor:v0.31.0
21	grafana/grafana:6.7.4

项目实战

更改数据库连接

根据edu-config-boot微服项目配置信息。将数据库连接地址更改为rancher-agent-121服务器地址

```
https://gitee.com/lagoedu/lagou-edu-repo
```

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://192.168.198.121:3306/edu_ad?
useUnicode=true&useSSL=false&characterEncoding=utf8&serverTimezone=Asia/Shanghai
    username: root
    password: admin
```

edu-config-boot

pom.xml

增加springboot打包插件。

```xml
    <build>
        <finalName>${project.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal><!--可以把依赖的包都打包到生成的Jar
包中-->
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

打包项目

```
mvn clean package
```

edu-bom

```
mvn clean install
```

edu-common

```
mvn clean install
```

edu-eureka-boot

pom.xml

增加springboot打包插件。

```
    <build>
        <finalName>${project.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal><!--可以把依赖的包都打包到生成的Jar
包中-->
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

application.yml

```
server:
  port: 8761

spring:
  application:
    name: edu-eureka-boot
```

```yaml
eureka:
  instance:
    hostname: 192.168.198.121
  client:
    fetch-registry: false
    register-with-eureka: false
    service-url:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

打包项目

```
mvn clean package
```

edu-gateway-boot

pom.xml

增加springboot打包插件。

```
    <build>
        <finalName>${project.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal><!--可以把依赖的包都打包到生成的Jar
包中-->
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

application.yml

```
eureka:
  client:
    service-url:
      defaultZone: http://192.168.198.121:8761/eureka/
```

打包项目

```
mvn clean package
```

edu-ad-boot

edu-ad-boot

```
mvn clean install
```

edu-ad-boot-api

```
mvn clean install
```

edu-ad-boot-impl

pom.xml

增加springboot打包插件。

```
    <build>
        <finalName>${project.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal><!--可以把依赖的包都打包到生成的Jar
包中-->
                        </goals>
                    </execution>
                </executions>
            </plugin>
```

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

application.yml

```
spring:
  application:
    name: edu-ad-boot

  cloud:
    config:
      uri: http://192.168.198.121:8090
      label: master
      profile: dev
      name: lagou-edu-ad

eureka:
  client:
    service-url:
      defaultZone: http://192.168.198.121:8761/eureka/
```

打包项目

```
mvn clean install
```

edu-front-boot

pom.xml

增加springboot打包插件。

```
    <build>
        <finalName>${project.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal><!--可以把依赖的包都打包到生成的Jar
包中-->
```

```
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

application.yml

```
server:
  port: 8081
spring:
  application:
    name: edu-front-boot

eureka:
  client:
    service-url:
      defaultZone: http://192.168.198.121:8761/eureka/
  instance:
    prefer-ip-address: true
    instance-id: ${spring.cloud.client.ip-
address}:${spring.application.name}:${server.port}
```

打包项目

```
mvn clean package
```

```
mvn clean install
```

edu-boss-boot

pom.xml

增加springboot打包插件。

```
    <build>
        <finalName>${project.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal><!--可以把依赖的包都打包到生成的Jar
包中-->
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

application.yml

```
server:
  port: 8082

spring:
  application:
    name: edu-boss-boot
eureka:
  client:
    service-url:
      defaultZone: http://192.168.198.121:8761/eureka/

  instance:
    prefer-ip-address: true
    instance-id: ${spring.cloud.client.ip-
address}:${spring.application.name}:${server.port}
```

打包项目

```
mvn clean package
```

制作项目镜像

基础镜像

```
docker pull openjdk:8-alpine3.9
docker pull mysql:5.7.31
```

项目结构

```
cd /data
mkdir -p edu-bom
cd edu-bom
mkdir -p mysql edu-eureka-boot edu-config-boot edu-gateway-boot edu-user-boot
edu-front-boot edu-boss-boot edu-ad-boot
```

sh脚本

```
#!/bin/bash
echo '创建根项目edu-bom目录'
cd /data
mkdir -p edu-bom
cd edu-bom
echo '创建每个子项目目录'
mkdir -p mysql edu-eureka-boot edu-config-boot edu-gateway-boot edu-user-boot
edu-front-boot edu-boss-boot edu-ad-boot
```

授权

```
chmod 777 initlagouedu-bom.sh
```

运行脚本

```
./initlagou.sh
```

mysql数据库

初始化mysql容器时自动导入项目所需要的数据库

Dockerfile

```
FROM mysql:5.7.31
MAINTAINER mysql from date UTC By Asia/Shanghai "laogsiji@lagou.com"
ENV TZ Asia/Shanghai
COPY edu_ad.sql /docker-entrypoint-initdb.d
COPY edu_authority.sql /docker-entrypoint-initdb.d
COPY edu_comment.sql /docker-entrypoint-initdb.d
COPY edu_course.sql /docker-entrypoint-initdb.d

COPY edu_oauth.sql /docker-entrypoint-initdb.d
COPY edu_order.sql /docker-entrypoint-initdb.d
COPY edu_pay.sql /docker-entrypoint-initdb.d
COPY edu_user.sql /docker-entrypoint-initdb.d
```

制作镜像

```
docker build --rm -t lagou/mysql:5.7 .
docker images

我们要使用mysql目录进行数据卷挂载，需要删除mysql目录中所有文件。
rm -rf *
```

运行镜像

```
docker run -itd --name mysql --restart always -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=admin -v /data/edu-bom/mysql:/var/lib/mysql lagou/mysql:5.7
 --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci


查看日志有错误信息。
docker logs -f mysql
```

sqlyog客户端测试

```
192.168.198.121
username:root
password:admin
```

部署edu-eureka-boot项目

打包edu-eureka-boot项目

```
mvn clean package
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-eureka-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY edu-eureka-boot.jar app.jar
EXPOSE 8761
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
cd /data/edu-bom/edu-eureka-boot

docker build --rm -t lagou/edu-eureka-boot:1.0 .
```

运行镜像

```
docker run -itd --name lagoueureka -p 8761:8761 lagou/edu-eureka-boot:1.0

docker logs -f lagoueureka
```

测试项目

```
http://192.168.198.121:8761
```

部署edu-config-boot项目

打包edu-config-boot项目

```
mvn clean package
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-config-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY edu-config-boot.jar app.jar
EXPOSE 8090
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
cd /data/edu-bom/edu-config-boot

docker build --rm -t lagou/edu-config-boot:1.0 .
```

运行镜像

```
docker run -itd --name lagouconfig -p 8090:8090 lagou/edu-config-boot:1.0

docker logs -f lagouconfig
```

部署edu-gateway-boot项目

打包edu-gateway-boot项目

```
mvn clean package
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-gateway-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY edu-gateway-boot.jar app.jar
EXPOSE 9001
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
cd /data/edu-bom/edu-gateway-boot

docker build --rm -t lagou/edu-gateway-boot:1.0 .
```

运行镜像

```
docker run -itd --name lagougateway -p 9001:9001 lagou/edu-gateway-boot:1.0

docker logs -f lagougateway
```

部署edu-ad-boot-impl项目

安装edu-ad-boot项目

```
mvn clean install
```

打包edu-ad-boot-api项目

```
mvn clean install
```

打包edu-ad-boot-impl项目

```
mvn clean package
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-ad-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY edu-ad-boot-impl.jar app.jar
EXPOSE 8001
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
docker build --rm -t lagou/edu-ad-boot:1.0 .
```

运行镜像

```
docker run -itd --name lagouad -p 8001:8001 lagou/edu-ad-boot:1.0

docker logs -f lagouad
```

测试项目

```
http://192.168.198.121:8001/ad/space/getAllSpaces
```

部署edu-boss-boot项目

打包edu-boss-boot项目

```
mvn clean package
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-boss-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY edu-boss-boot.jar app.jar
EXPOSE 8082
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
cd /data/edu-bom/edu-boss-boot

docker build --rm -t lagou/edu-boss-boot:1.0 .
```

运行镜像

```
docker run -itd --name lagouboss -p 8082:8082 lagou/edu-boss-boot:1.0

docker logs -f lagouboss
```

测试项目

使用gateway访问项目

```
http://192.168.198.121:9001/boss/ad/space/getAllSpaces
```

部署edu-front-boot项目

打包edu-front-boot项目

```
mvn clean package
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-front-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY edu-front-boot.jar app.jar
EXPOSE 8081
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
docker build --rm -t lagou/edu-front-boot:1.0 .
```

运行镜像

```
docker run -itd --name lagoufront -p 8081:8081 lagou/edu-front-boot:1.0

docker logs -f lagoufront
```

保存项目镜像

```
docker save \
    lagou/edu-front-boot:1.0 \
    lagou/edu-boss-boot:1.0 \
    lagou/edu-ad-boot:1.0 \
    lagou/edu-gateway-boot:1.0 \
    lagou/edu-config-boot:1.0 \
    lagou/edu-eureka-boot:1.0 \
    lagou/mysql:5.7 \
-o lagou-bom.tar
```

删除项目容器

```
停止所有运行容器
docker stop $(docker ps -qa)

删除所有的容器
docker rm $(docker ps -aq)

删除所有的镜像
docker rmi $(docker images -q)
```

sh脚本

```
#!/bin/bash
echo '停止所有运行容器'
docker stop $(docker ps -qa)

echo '删除所有的容器'
docker rm $(docker ps -aq)

echo '删除所有的镜像'
docker rmi $(docker images -q)
```

授权

```
chmod 777 stopdocker.sh
```

运行脚本

```
./ stopdocker.sh
```

ELK

ELK日志分析简介

ELK可运行于分布式系统之上，通过搜集、过滤、传输、储存，对海量系统和组件日志进行集中管理和准实时搜索、分析，使用搜索、监控、事件消息和报表等简单易用的功能，帮助运维人员进行线上业务的准实时监控、业务异常时及时定位原因、排除故障、程序研发时跟踪分析Bug、业务趋势分析、深度挖掘日志的大数据价值。

ELK主要可解决的问题如下：

1. 日志查询，问题排查，上线检查
2. 服务器监控，应用监控，错误报警，Bug管理
3. 性能分析，安全漏洞分析。

综上，ELK是一套方便、易用的日志分析开源解决方案。

ELK主要组件介绍

ElasticSearch组件

ElasticSearch是一个基于Lucene的开源分布式搜索服务器。它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是第二流行的企业搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。

Logstash组件

Logstash是一个完全开源的工具，它可以对你的日志进行收集、过滤、分析，支持大量的数据获取方法，并将其存储供以后使用（如搜索）。一般工作方式为c/s架构，client端安装在需要收集日志的主机上，server端负责将收到的各节点日志进行过滤、修改等操作在一并发往elasticsearch上去。Logstash事件处理有三个阶段：inputs → filters → outputs。是一个接收，处理，转发日志的工具

Kibana组件

Kibana是一个基于浏览器页面的Elasticsearch前端展示工具，也是一个开源和免费的工具，Kibana可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助您汇总、分析和搜索重要数据日志。

Filebeat组件

Filebeat是一个日志文件托运工具，在你的服务器上安装客户端后，filebeat会监控日志目录或者指定的日志文件，追踪读取这些文件，并且转发这些信息到赋予他的输出位置（常见如elasticsearch、kafka或logstarsh）中存放。Filebeat等同于一个轻量级的logstash，当你需要收集信息的机器配置或日志资源数量不是特别多时，最佳实践是使用filebeat来代替logstash收集日志，filebeat十分小巧且稳定。下图是filebeat的工作流程：当你开启filebeat程序的时候，它会启动一个或多个探测器（prospectors）去检测你指定的日志目录或文件，对于探测器找出的每一个日志文件，filebeat启动收割进程（harvester），每一个收割进程读取一个日志文件的新内容，并发送这些新的日志数据到处理程序（spooler），处理程序会集合这些事件，最后filebeat会发送集合的数据到你指定的地点。

docker官网

包含ELK的docker官网地址

```
https://hub.docker.com/_/elasticsearch
https://hub.docker.com/_/kibana
https://hub.docker.com/_/logstash
```

ELK官网

```
https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html
```

基础镜像

ELK官网镜像

```
docker pull docker.elastic.co/logstash/logstash:7.7.0
docker pull docker.elastic.co/kibana/kibana:7.7.0
docker pull docker.elastic.co/elasticsearch/elasticsearch:7.7.0
```

注意事项

1. 不要下载docker官方的镜像，最好使用 elastic官方仓库里的镜像
2. 如果出现启动不成功，先把其他两个注释掉，一个一个服务单独运行试试
3. logstash需要监听elasticsearch服务，不然logstash会自动停掉

4. logstash和kibana可以不用设置关联 elasticsearch 的环境变量，官网文档说如果是docker镜像有默认设置
5. 如果自己修改了服务名称或者端口要记得修改默认的设置

docker官网镜像

```
docker pull logstash:7.7.0
docker pull kibana:7.7.0
docker pull elasticsearch:7.7.0
```

前置条件

文件创建数

修改Linux系统的限制配置，将文件创建数修改为65536个：

1. 修改系统中允许应用最多创建多少文件等的限制权限。Linux默认来说，一般限制应用最多创建的文件是65535个。但是ES至少需要65536的文件创建数的权限。
2. 修改系统中允许用户启动的进程开启多少个线程。默认的Linux限制root用户开启的进程可以开启任意数量的线程，其他用户开启的进程可以开启1024个线程。必须修改限制数为4096+。因为ES至少需要4096的线程池预备。

```
vi /etc/security/limits.conf
#新增如下内容在limits.conf文件中
es soft nofile 65536
es hard nofile 65536
es soft nproc 4096
es hard nproc 4096
```

系统控制权限

修改系统控制权限，ElasticSearch需要开辟一个65536字节以上空间的虚拟内存。Linux默认不允许任何用户和应用程序直接开辟这么大的虚拟内存。

```
添加参数:新增如下内容在sysctl.conf文件中，当前用户拥有的内存权限大小
vm.max_map_count=262144

echo "vm.max_map_count=262144" > /etc/sysctl.conf

重启生效:让系统控制权限配置生效
sysctl -p
```

```
mkdir -p /data/elk
cd /data/elk
```

docker-compose方式

采用ELK官网提供镜像方式安装。

```
version: '3'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.7.0
    container_name: elasticsearch
    environment:
      - "ES_JAVA_OPTS=-Xms2048m -Xmx2048m"
      - discovery.type=single-node
    ulimits:
      memlock:
        soft: -1
        hard: -1
    hostname: elasticsearch
    restart: always
    ports:
    - 9200:9200
    - 9300:9300
  kibana:
    image: docker.elastic.co/kibana/kibana:7.7.0
    container_name: kibana7
    environment:
      - elasticsearch.hosts=http://elasticsearch:9200
    hostname: kibana
    depends_on:
      - elasticsearch
    restart: always
    ports:
    - 5601:5601
  logstash:
    image: docker.elastic.co/logstash/logstash:7.7.0
    container_name: logstash7
    restart: always
    depends_on:
      - elasticsearch
    ports:
    - 9600:9600
    - 5044:5044
```

启动容器

```
docker-compose up -d
```

测试ELK环境

```
http://192.168.198.121:9200

http://192.168.198.121:5601
```

sebp/elk

镜像`sebp/elk`支持ElasticSearch、Logstash和Kiabana分别启动，也支持单容器启动，同时也支持集群启动

docker官网

```
https://hub.docker.com/r/sebp/elk
```

官网教程

```
https://elk-docker.readthedocs.io/
```

github官网

```
https://github.com/spujadas/elk-docker
```

基础镜像

```
docker pull sebp/elk:770
```

试运行容器

先把对应的文件都拷贝到宿主机当中

```
cd /data
mkdir -p es kibana logstash
```

如果你的主机内存低于4G，建议增加配置设置ES内存使用大小，以免启动不了。

```
docker run -itd --name elk \
    -p 5601:5601 \
    -p 9200:9200 \
    -p 5044:5044 \
 sebp/elk:770
```

只是在开发调试过程中短期运行，其用户数据并无保留的必要，因而可以在容器启动时设置--rm选项，这样在容器退出时就能够自动清理容器内部的文件系统。方便在临时测试使用。示例如下：

```
docker run --rm --name elk  sebp/elk:770
```

注意事项

- --rm选项不能与-d同时使用
- --rm选项也会清理容器的匿名data volumes。
- 所以，执行docker run命令带--rm命令选项，等价于在容器退出后，执行docker rm -v。

查看ELK目录

sebp/elk存放ELK的目录与官方的镜像文件目录不一致，各位小伙伴要注意区别。

通过docker exec -it elk /bin/bash可进入容器中，具体各服务配置文件路径如下

```
docker exec -it elk /bin/bash

cd /opt

es 配置文件路径
  /opt/elasticsearch

logstash配置文件路径
  /opt/logstash/

logstash配置文件路径
  /opt/logstash/
```

复制配置文件

从elk容器内复制ELK的配置文件。

```
容器内的目录copy虚拟机中
docker cp  elk:/opt/kibana/config/ /data/kibana/
docker cp  elk:/opt/elasticsearch/config /data/es
docker cp  elk:/opt/elasticsearch/plugins /data/es
docker cp  elk:/opt/logstash/config /data/logstash

把虚拟机的目录copy容器中
docker cp  /data/logstash elk:/opt/logstash/config
```

删除镜像

```
如果没有采用docker run --rm方式运行镜像。请先停止容器再删除容器
docker stop elk
docker rm elk
```

修改配置文件

elasticsearch.yml

增加跨域配置:

```
http.cors.enabled: true
http.cors.allow-origin: "*"
```

kibana.yml

增加中文配置:

```
i18n.locale: "zh-CN"
```

pipelines.yml

```
- pipeline.id: main
  path.config: "/opt/logstash/config/*.conf"
```

```
input {
  beats {
    port => 5044
  }
}

output {
```

```
  elasticsearch {
    hosts => ["http://192.168.198.121:9200"]
    index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
    #user => "elastic"
    #password => "changeme"
  }
}
```

运行镜像

docker方式

```
docker run -itd --name  elk --restart=always \
     -p 5601:5601 -p 5044:5044 -p 9200:9200 -p 9300:9300 \
     -v /data/kibana/config:/opt/kibana/config/ \
     -v /data/es/config:/opt/elasticsearch/config \
     -v /data/es/plugins:/opt/elasticsearch/plugins \
     -v /data/logstash/config:/opt/logstash/config \
     sebp/elk:770
```

docker-compose方式

```
version: '3'
services:
  elasticsearch:
    image: sebp/elk:770
    ports:
      - "5601:5601"
      - "9200:9200"
      - "5044:5044"
    volumes:
      - /data/kibana/config:/opt/kibana/config/
      - /data/es/config:/opt/elasticsearch/config
      - /data/es/plugins:/opt/elasticsearch/plugins
      - /data/logstash/config:/opt/logstash/config
    hostname: elasticsearch
    restart: always
```

测试ELK环境

```
http://192.168.198.121:9200

http://192.168.198.121:5601
```

测试篇

jmeter

简介

- jmeter是由Apache公司开发的一个纯Java的开源项目，即可以用于做接口测试也可以用于做性能测试、压力测试。
- jmeter具备高移植性，可以实现跨平台运行。
- jmeter可以实现分布式负载。
- jmeter采用多线程，允许通过多个线程并发取样或通过独立的线程对不同的功能同时取样。
- jmeter具有较高扩展性。

百科

```
https://baike.baidu.com/item/Jmeter/3104456?fr=aladdin
```

官网地址

```
https://jmeter.apache.org/

最新版本下载地址：
https://jmeter.apache.org/download_jmeter.cgi

历史版本下载地址：
https://archive.apache.org/dist/jmeter/binaries/
```

安装要求

JDK8+以上版本

环境变量配置

JMETER_HOME

值为你解压的jmeter安装路径。我的安装路径是在d盘，这个路径根据自己实际安装路径进行填写。然后点击确定保存即可

```
D:\apache-jmeter-5.3
```

PATH

```
%JMETER_HOME%\bin;
```

CLASSPATH

CLASSPATH为选择性配置。可以不进行配置。

变量值固定为：
%JMETER_HOME%\lib\ext\ApacheJMeter_core.jar;%JMETER_HOME%\lib\jorphan.jar;%JMETER_HOME%\lib/logkit-2.0.jar; 做完之后一定要保存

```
%JMETER_HOME%\lib\ext\ApacheJMeter_core.jar;%JMETER_HOME%\lib\jorphan.jar;%JMETER_HOME%\lib/logkit-2.0.jar
```

测试

首先进到你的jmeter安装路径，找到bin文件夹，点击进去，找到jmeter.bat，鼠标右键用管理员方式运行，或者直接双击打开，此时会弹出2个界面：1.个是命令窗口，使用jmeter的时候此命令窗口不能关，你缩小到电脑任务栏即可。2.还有一个界面是jmeter工作页面，你可以在里面进行相关的操作.

jmeter目录结构

bin目录

```
examples目录中有CSV样例启动jmeter

ApacheJMeter.jar
jmeter.bat
jmeter.sh

jar包方式启动
java -jar ApacheJMeter.jar



jmeter分布式集群
```

```
jmeter-server.bat

jmeter配置文件
jmeter.properties
```

jmeter.bat

```
151行左右：set HEAP=-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m

修改jmeter JVM内存使用，设置参数时不要超过自己电脑物理内存的50%
set HEAP=-Xms4g -Xmx4g -XX:MaxMetaspaceSize=1024m

启动的时候，控制台依然显示256m；依鄙人经验，断定--bug，绝对的bug，编程干久了总想追根问底；通过jmeter的源码文件“JMeter.java”找，他这个显示是固定文字，所以不管你设置多少，他总是显示256。
```

jmeter.properties

```
大概在文件的39行，去掉注释信息，修改为简体中文版
language=zh_CN


#大概在文件的1086行，去掉注释信息，修改返回结果集编码
sampleresult.default.encoding=utf-8
```

docs目录

提供jmeter所有类库的api帮助手册。

```
%JMETER_HOME%\docs\api下的index.html
```

extras目录

扩展插件目录。提供了对Ant的支持，可以使用Ant来实现自动化测试，例如批量脚本执行，产生html格式的报表，测试运行时，可以把测试数据记录下来，jmeter会自动生成一个.jtl文件，将该文件放到extras目录下，运行"ant -Dtest=文件名 report"，就可以生成测试统计报表。

lib目录

所用到的插件目录，里面均为jar包。jmeter会自动在jmeter_HOME/lib和ext目录下寻找需要的类，lib下存放JMeter所依赖的外部jar：如httpclient.jar、httpcore.jar、httpmime.jar等等。

其中lib\ext目录下存放有Jmeter依赖的核心jar包，ApacheJMeter_core.jar、ApacheJMeter_java.jar在写client端需要引用，JMeter插件包也在此目录下。

lib\junit下存放junit测试脚本.

注意：无法识别 zip 格式的包文件，所以需要的包文件均要求以 .jar 结尾

Printable_docs目录

用户使用手册。usermanual子目录下是jmeter用户手册，尤其是component_reference.html是最常用的核心元件帮助手册。

```
%JMETER_HOME%\printable_docs\下的index.html
```

Licenses目录

jmeter证书目录

设置中文界面

方式一

jmeter默认是英语环境，但是对于英语不是很好的人来说看起来非常吃力，所以可以通过设置来显示中文。菜单栏【Options】按钮，然后依次单击【Choose language】>【Chinese(simplified)】

方式二

也可以通过设置jmeter.properties配置文件来设置语言为中文。打开D:\apache-jmeter-5.3\bin\jmeter.properties文件。修改language=zh_CN

```
大概在文件的39行，去掉注释信息，修改为简体中文版

language=zh_CN
```

常用设置

改变背景

菜单栏【选项】按钮，然后依次单击【外观】>【Windows】

改变字体

菜单栏【选项】按钮，然后单击【放大】或者【缩小】

jmeter主要元件

1、**测试计划**：是使用 JMeter 进行测试的起点，是其它 JMeter测试元件的容器

2、**线程组**：代表一定数量的用户，它可以用来模拟用户并发发送请求。实际的请求内容在Sampler中定义，它被线程组包含。

3、**配置元件**：维护取样器需要的配置信息，并根据实际的需要修改请求的内容。

4、**前置处理器**：负责在请求之前工作，常用来修改请求的设置

5、**定时器**：负责定义请求之间的延迟间隔。

6、**取样器(Sampler)**：是性能测试中向服务器发送请求，记录响应信息、响应时间的最小单元，如：HTTP Request Sampler、FTP Request Sample、TCP Request Sample、JDBC Request Sampler等，每一种不同类型的sampler 可以根据设置的参数向服务器发出不同类型的请求。

7、**后置处理器**：负责在请求之后工作，常用获取返回的值。

8、**断言**：用来判断请求响应的结果是否如用户所期望的。

9、**监听器**：负责收集测试结果，同时确定结果显示的方式。

10、**逻辑控制器**：可以自定义JMeter发送请求的行为逻辑，它与Sampler结合使用可以模拟复杂的请求序列。

Jmeter元件的作用域和执行顺序

1.元件作用域

配置元件：影响其作用范围内的所有元件。

前置处理器：在其作用范围内的每一个sampler元件之前执行。

定时器：在其作用范围内的每一个sampler有效

后置处理器：在其作用范围内的每一个sampler元件之后执行。

断言：在其作用范围内的对每一个sampler元件执行后的结果进行校验。

监听器：在其作用范围内对每一个sampler元件的信息收集并呈现。

总结：从各个元件的层次结构判断每个元件的作用域。

2.元件执行顺序：

配置元件->前置处理器->定时器->取样器->后置处理程序->断言->监听器

注意事项：

1.前置处理器、后置处理器和断言等组件只能对取样器起作用，因此，如果在它们的作用域内没有任何取样器，则不会被执行。

2.如果在同一作用域内有多个同一类型的元件，则这些元件按照它们在测试计划中的上下顺序依次执行。

Jmeter进行接口测试流程

使用Jmeter进行接口测试的基本步骤如下：

1.测试计划

2.线程组

3.HTTP Cookie管理器

4.Http请求默认值

5.Sampler(HTTP请求)

6.断言

7.监听器(查看结果树、图形结果、聚合报告等)

get方式测试

线程组

新增线程组

HTTP请求

新增http请求

察看结果树

新增结果树

JSON Path Expression

$[0].username

get方式带参

线程组

新增线程组

HTTP请求

新增http请求

察看结果树

新增结果树

JSON Path Expression

$[0].username

post方式测试

键值对格式

无@RequestBody

线程组

新增线程组

HTTP请求

新增http请求

察看结果树

新增结果树

JSON格式

有@RequestBody

线程组

新增线程组

HTTP信息头管理器

HTTP信息头管理器增加参数信息：Content-Type=application/json;charset=UTF-8

HTTP请求

删除参数中的信息。新增消息体参数，参数内容如下：

```
{
  "username":"admin",
  "password":"1234"
}
```

察看结果树

新增结果树

优化配置项

经过测试。我们发现测试内容中有很多是重复的，例如：服务器名称、端口号、内容编码等信息。

HTTP请求默认值

测试计划->添加->配置元件->HTTP请求默认值

为所有的线程组添加HTTP请求默认值。将重复的信息配置在HTTP请求默认值中。线程组中不用再次配置。

压测数据库

jmeter是一个轻量级的性能测试工具，这是已经总所周知的问题，今天我们使用jmeter批量向数据库插入数据，这个功能在实际工作中也可以帮我们提高工作效率。

在以往的项目中，当需要做批量数据的性能测试时，一般会使用python或者Perl编写脚本，然后往数据库中插入数据，这个是我能想到的最快速，最不枯燥的方法，但是写脚本需要一定的时间，如果有更高效的方式，为什么不使用呢？无论使用任何方式，我们的共同目标都是把工作高效率的做好。

下面介绍一下Jmeter向mysql数据库中插入数据的操作。

线程组

在测试计划上右键-->添加-->线程(用户)-->线程组(线程组名称为：localmysql5)

配置元件

在线程组上右键-->添加-->配置元件-->JDBC Connection Configuration

Random Variable

添加随机数生成元件，在线程组上右键-->添加-->配置元件-->Random Variable

取样器

在线程组上右键-->添加-->取样器-->JDBC Request

监听器

在线程组上右键-->添加-->监听器-->查看结果树

配置元件

上面是使用Jmeter向数据库中添加数据时需要的主要元件，接下来做一下各个元件的配置。

线程组配置

我们需要插入的数据量可以在线程组的线程数、循环次数进行配置，如下，这里只插入10条数据，使用循环10次进行控制。

增加数据库驱动jar包。

JDBC Connection Configuration配置

重要字段介绍

名称：设置的是该元件的名称，设置名称后【测试计划】树配置元件也会对应更改

Variable Name Bound to Pool：数据库连接池的名称。可以有多个jdbc connection configuration，每个可以起不同的名称，在Jmeter其他元件中只要需要用到数据库的连接信息，直接引用该变量的变量名即可。可以理解为如果Jmeter其他的元件要获取数据库的连接信息，可以通过这个名称进行获取。

注意：变量的命名规范：命名要清晰，建议使用英文，便于引用。

Connection Pool Configuration 、 Connection Validation by Pool 这两部分内容不需要更改，使用默认值即可

Database Connection Configuration（以MySQL数据库为例）：

Database URL: jdbc:mysql://host[:port]/dbname ——>【数据库地址：jdbc:mysql://数据库主机名或IP地址:端口号/需要使用的库名】

JDBC Driver class：com.mysql.jdbc.Driver【其他数据库跟进图一展示进行选择】

Username：数据库名称，即用户名

Password：数据库链接密码

注意：

A.【allowMultiQueries设置为true，就可执行多条sql语句，参数与参数之间连接需要使用“&”符号。使用场景：如果执行多条SQL语句就必须要加这个字段，否则会报错；执行一条SQL语句可以不用加这个字段】

B.【characterEncoding=UTF-8：链接的数据库的编码格式，没有设置该字段，将不会转译中文字符】

设置随机变量

变量名称：randomValue(JDBC Request的SQL语句中使用)

最小值：1

最大值：200

配置JDBC Request

重要字段解释：

Variable Name Bound to Pool：引用JDBC Connection Configuration元件里面的参数值【test】写法两边保持一致即可，不需要使用${}进行引用。错误的不存在的参数不能被引用。

Query type：必填，指SQL请求类型

Select statement：查询语句类型（select），只支持一条查询语句，多条查询语句只执行第一条

Update statement：更新语句类（insert，update，delete），只支持一条更新语句，多条更新语句只执行第一条

Prepared Select statement：支持多条查询（select）语句，查询响应数据只展示第一条SQL的查询结果

Prepared Update statement：支持多条更新（insert，update，delete）语句，响应数据展示多条更新提示

Callable Statement：支持多条查询、更新（insert，update，delete，select）语句，响应数据展示展示多条数据更新结果。如果是多条select语句同时查询，建议使用Callable Statement，响应数据可以展示多条查询结果值

Parameter values：填写参数的具体的值，或者参数的名称。可以利用此字段对SQL语句进行参数化

Parameter types：指Parameter Values参数的数据类型，例如：integer，String，double类型

Parameter values 和Parameter types：必须成对出现，且SQL语句中有多个参数，就必须有多少个parameter values 和Parameter types。

Variable names：自己设置的变量名称，用于存放select操作返回的查询结果。有多个字段返回时，需用逗号隔开

Result variable name:用于存放select操作返回的查询结果集

Query timeout：查询超时时间

Handle result set：定义如何处理由callable statements 语句返回的结果

```
INSERT INTO tbuser (username,password,userroles,nickname) VALUES
('admin${randomValue}','1234','04','管理员${randomValue}')
```

察看结果树

在监听器中新增察看结果树

聚合报告

在监听器中新增聚合报告

用表格察看结果

在监听器中新增用表格察看结果

图形结果

在监听器中新增图形结果

运维篇

rancher-server

官网地址

英文官网地址

```
https://rancher.com/
```

中文官网地址

```
https://www.cnrancher.com/
```

帮助文档地址

```
https://docs.rancher.cn/rancher1/
```

节点信息

服务器用户名：root，服务器密码：123456

主机名	IP地址	说明
rancher-server-120	192.168.198.120	rancher-server服务器
rancher-agent-121	192.168.198.121	1.需要内存大。推荐10G内存 2.部署微服项目的工作节点

基础镜像

docker官网地址

```
https://hub.docker.com/r/rancher/server
```

rancher-server-120节点需要镜像

```
docker pull rancher/server:v1.6.30

镜像备份
docker save rancher/server:v1.6.30 -o rancher-server.1.6.30.tar

导入镜像
docker load -i rancher-server.1.6.30.tar
```

rancher-agent-121节点需要镜像

下载镜像

```
docker pull rancher/agent:v1.2.11
docker pull rancher/scheduler:v0.8.6
docker pull rancher/net:v0.13.17
docker pull rancher/dns:v0.17.4
docker pull rancher/healthcheck:v0.3.8
docker pull rancher/metadata:v0.10.4
docker pull rancher/network-manager:v0.7.22
docker pull rancher/storage-nfs:v0.9.1
docker pull rancher/net:holder
```

备份镜像

```
docker save  \
    rancher/scheduler:v0.8.6   \
    rancher/agent:v1.2.11  \
    rancher/net:v0.13.17 \
    rancher/dns:v0.17.4  \
    rancher/healthcheck:v0.3.8   \
    rancher/metadata:v0.10.4  \
    rancher/network-manager:v0.7.22  \
    rancher/storage-nfs:v0.9.1   \
    rancher/net:holder   \
-o rancher-agent.tar
```

导入镜像

```
docker load -i rancher-agent.tar
```

安装rancher

Docker容器的重启策略如下：

- no:默认策略，在容器退出时不重启容器
- on-failure:在容器非正常退出时（退出状态非0），才会重启容器
- on-failure:3:在容器非正常退出时重启容器，最多重启3次
- always:在容器退出时总是重启容器
- unless-stopped:在容器退出时总是重启容器，但是不考虑在Docker守护进程启动时就已经停止了的容器

```
安装rancher
docker run -itd --name rancher-server --restart=unless-stopped -e JAVA_OPTS="-
Xmx4096m" -p 8080:8080 rancher/server:v1.6.30

查看安装日志情况：
docker logs -f rancher-server
```

运行rancher

```
http://192.168.198.120:8080/
```

rancher部署方式

单容器部署一

（使用容器内部自带的MySQL数据库）

在安装和部署了Docker容器环境的Linux服务器上，使用一个简单的命令就可以启动一个单实例的Rancher。

```
docker run -itd --name rancher-server --restart=unless-stopped -p 8080:8080
rancher/server:v1.6.30
```

单容器部署二

（使用外部MySQL数据库）

除了使用内部的数据库，你可以启动一个Rancher Server并使用一个外部的数据库。启动命令与之前一样，但添加了一些额外的参数去说明如何连接你的外部数据库。

使用外部数据库，需要提前创建数据库名和数据库用户，Rancher服务启动后会自动创建Rancher管理平台需要的数据库表。以下为相关的建库脚本， 可供参考：

```
CREATE DATABASE IF NOT EXISTS cattle COLLATE = 'utf8_general_ci' CHARACTER SET =
'utf8';
GRANT ALL ON cattle.* TO 'cattle'@'%' IDENTIFIED BY 'cattle';
GRANT ALL ON cattle.* TO 'cattle'@'localhost' IDENTIFIED BY 'cattle';
```

启动一个Rancher连接一个外部数据库，你需要在启动容器的命令中添加额外参数。

```
docker run -d --restart=unless-stopped -p 8080:8080 rancher/server \
    --db-host 192.168.198.120 --db-port 3306 --db-user username \
    --db-pass password --db-name cattle
```

大部分的输入参数都有默认值并且是可选的，只有MySQL数据库主机地址配置项是必须配置的。

```
--db-host               数据库主机名或IP地址
--db-port               数据库服务端口(默认为:3306)
--db-user               数据库用户名(默认为:cattle)
--db-pass               数据库用户密码(默认为:cattle)
--db-name               数据库名(默认为:cattle)
```

rancher初始化

配置环境

rancher-server 支持将资源分组归属到多个环境。 每个环境具有自己独立的基础架构资源及服务，并由一个或多个用户、团队或组织所管理。例如，您可以创建独立的“开发(dev)”、“测试(test)”及“生产(pro)”环境以确保环境之间的安全隔离，将“开发”环境的访问权限赋予全部人员，但限制“生产”环境的访问权限给一个小的团队。

选择“Default -->环境管理” 菜单

1. 环境名称：lagouedu
2. 填写名称，点击“创建”按钮

按照上述步骤，添加项目测试环境和生产环境

添加主机

选择基础架构-->主机 菜单，点击添加主机

1. 主机名称：rancher-agent-121
2. 拷贝脚本 ，在rancher-agent-121服务器上执行脚本

添加应用

点击应用(rancher应用一般是指我们的项目名称)-->全部(或用户)，点击“添加应用”按钮

1. 应用名称：lagou-bom

NFS服务

官网地址

使用rancher-server的应用商店进行安装，安装前请参考官方给出的帮助文档
https://docs.rancher.cn/docs/rancher1/rancher-service/storage-services/rancher-nfs/_index/

安装NFS

```
120、121节点都需要安装nfs服务。120节点为nfs服务端。
yum install -y nfs-utils rpcbind


在rancher-server-120节点创建目录
mkdir -p /nfs
chmod 777 /nfs

更改归属组与用户
chown nfsnobody /nfs
或者
chown -R nfsnobody:nfsnobody /nfs

使用NFS4协议方式进行多共享目录配置。所有共享目录的根目录为/nfs/data。服务器端的/etc/exports文件中的配置为：
vi /etc/exports

/nfs  *(rw,sync,no_subtree_check,no_root_squash)

启动服务
systemctl start rpcbind && systemctl start nfs

设置开启启动
systemctl enable rpcbind && systemctl enable nfs

检查配置是否生效
exportfs

showmount -e 192.168.198.120
```

应用商店

配置NFS服务

应用部署

- 使用rancher-server的WEB UI界面部署项目。项目所有镜像部署在rancher-agent-121节点。
- 请各位小伙伴提前删除rancher-agent-121节点部署的微服项目容器。否则会造成部署不成功。

mysql

基础镜像

```
docker pull lagou/mysql:5.7
```

挂载卷

```
基础架构->存储->添加卷
实际上就是在nfs服务器120节点创建一个共享目录
```

部署mysql服务

rabbitmq

docker官网

```
https://hub.docker.com/_/rabbitmq
```

github官网

```
https://github.com/docker-library/rabbitmq
```

基础镜像

```
docker pull rabbitmq:management-alpine

docker save rabbitmq:management-alpine -o rabbitmq.tar
docker load -i rabbitmq.tar
```

部署rabbitmq

```
端口映射:5671 5672 4369 15671 15672 25672
```

访问rabbitmq

```
http://192.168.198.121:15672/

默认用户名:guest
密      码:guest
```

redis

docker官网

```
https://hub.docker.com/_/redis
```

基础镜像

```
docker pull redis:6.0.8-alpine3.12
```

部署redis

```
端口映射:6379
```

测试redis

```
使用任意一款客户端软件访问reids

192.168.198.121:6379

set lagouedu laosiji
get lagouedu
```

微服项目部署

创建数据库

```
创建lagou数据库
```

创建用户表

```
创建tbuser表
CREATE TABLE `tbuser` (
  `userid` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `password` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `userroles` varchar(2) COLLATE utf8_bin DEFAULT NULL,
  `nickname` varchar(50) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COLLATE=utf8_bin
```

新增测试数据

```
INSERT INTO tbuser (username,PASSWORD,userroles,nickname) VALUES
('admin','1234','04','管理员'),('lagou','1234','03','拉勾教育')
```

application.yml

```
server:
  port: 8082
spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: admin
    url: jdbc:mysql://192.168.198.121:3306/lagou?characterEncoding=utf-
8&useSSL=false&useTimezone=true&serverTimezone=GMT%2B8

mybatis-plus:
  type-aliases-package: com.lagou.dockerdemo.entity
  mapper-locations: mapper/*.xml
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

打包项目

```
将jar包上传121节点/data/dockerdemo目录

mkdir -p /data/dockerdemo
cd /data/dockerdemo
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker springboot "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY dockerdemo.jar app.jar
EXPOSE 8082
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
docker build --rm -t lagou/dockerdemo:1.0 .
```

运行镜像

```
docker run -itd --name dockerdemo -p 8082:8082 lagou/dockerdemo:1.0
```

测试项目

```
http://192.168.198.121:8082/users
```

删除容器

```
docker stop dockerdemo
docker rm dockerdemo
```

部署项目

使用rancher-server部署dockerdemo项目。

rancher高级应用

扩容缩容

部署微服

部署微服项目时不要暴露端口号，多个容器使用同一个端口会造成端口号冲突。

增加扩容

API->WEBHOOKS，增加扩容服务

```
添加接收器：scalelagoudockerdemo
类型：扩缩容服务
设置步长:2

复制触发地址：
http://192.168.198.120:8080/v1-webhooks/endpoint?
key=xO8WIf4JOcf85PHPVdln2i9t2vIOGSowVemhT1Xq&projectId=1a11

使用postman或者idea的rest client
使用post方式提交
```

负载均衡

添加负载均衡器

部署edu-bom项目

mysql数据库

基础镜像

```
lagou/mysql:5.7
```

配置项

```
MYSQL_ROOT_PASSWORD:admin
端口号:3306:3306
nfs共享目录:mysqlmount
```

edu-eureka-boot项目

基础镜像

```
lagou/edu-eureka-boot:1.0
```

配置项

```
名称：lagoueureka
描述：edu-eureka-boot项目
端口号: 8761
```

测试项目

```
http://192.168.198.121:8761/
```

edu-config-boot项目

基础镜像

```
lagou/edu-config-boot:1.0
```

配置项

```
名称：lagouconfig
描述：edu-config-boot项目
端口号：8090
```

edu-gateway-boot项目

基础镜像

```
lagou/edu-gateway-boot:1.0
```

配置项

```
名称：lagougateway
描述：edu-gateway-boot项目
端口号：9001
```

edu-ad-boot-impl项目

基础镜像

```
lagou/edu-ad-boot:1.0
```

配置项

```
名称：lagouad
描述：edu-ad-boot项目
端口号：8001
```

测试项目

```
http://192.168.198.121:8001/ad/space/getAllSpaces
```

edu-boss-boot项目

基础镜像

```
lagou/edu-boss-boot:1.0
```

配置项

```
名称：lagouboss
描述：edu-boss-boot项目
端口号：8082
```

测试项目

使用gateway访问项目

```
http://192.168.198.121:9001/boss/ad/space/getAllSpaces

如果访问失败，需要把gateway服务重新启动。
标准的部署步骤应该是先部署edu-ad-boot再部署edu-boss-boot最后部署edu-gateway-boot
```

edu-front-boot项目

基础镜像

```
lagou/edu-front-boot:1.0
```

配置项

```
名称：lagoufront
描述：edu-front-boot项目
端口号：8081
```

监控edu-bom系统

本章节技能应用侧重运维工程师，请各位开发小伙伴了解docker方式安装即可。如果小伙伴想深入学习监控内容，请自行查阅相关的资料。

prometheus

官网教程

```
https://prometheus.io/docs/prometheus/latest/installation/
```

github官网

```
https://github.com/prometheus/prometheus
```

docker官网

```
https://hub.docker.com/r/prom/prometheus
```

基础镜像

```
docker pull prom/prometheus:v2.21.0
```

配置文件

prometheus.yml

```
# my global config 全局配置
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default
is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is
every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration 告警配置
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
#配置告警的规则
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
```

```
# Here it's Prometheus itself.配置被监控端
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped
from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']
```

运行镜像

```
mkdir -p /data/prometheus
cd /data/prometheus

docker run -itd --name prometheus \
--restart=always \
-p 9090:9090 \
-v /data/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml \
prom/prometheus:v2.21.0
```

测试prometheus

```
http://192.168.198.121:9090
```

基础镜像

```
docker pull alpine:3.11.6
```

下载tar包

```
官网下载速度很慢，推荐大家使用迅雷等工具进行下载
https://github.com/prometheus/prometheus/releases/download/v2.21.0/prometheus-
2.21.0.linux-amd64.tar.gz
```

准备制作镜像

```
将tar包上传服务器/data目录中
cd /data
tar zxf prometheus-2.21.0.linux-amd64.tar.gz

进入prometheus-2.21.0.linux-amd64目录中
cd prometheus-2.21.0.linux-amd64

vi Dockerfile
```

Dockerfile

```
FROM alpine:3.11.6
# 作者信息
MAINTAINER laosiji Docker springboot "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai
COPY prometheus                                /bin/prometheus
COPY promtool                                  /bin/promtool
COPY prometheus.yml                            /etc/prometheus/prometheus.yml
COPY console_libraries/
/usr/share/prometheus/console_libraries/
COPY consoles/                                 /usr/share/prometheus/consoles/
WORKDIR    /prometheus
EXPOSE     9090
ENTRYPOINT [ "/bin/prometheus" ]
CMD        [ "--config.file=/etc/prometheus/prometheus.yml", \
             "--storage.tsdb.path=/prometheus", \
             "--web.console.libraries=/usr/share/prometheus/console_libraries",
\
             "--web.enable-lifecycle", \
             "--web.console.templates=/usr/share/prometheus/consoles" ]
```

制作镜像

```
docker build --rm -t lagou/prometheus:v2.21.0 .
```

运行镜像

```
将prometheus.yml文件复制到/data/prometheus目录中
mkdir -p /data/prometheus
cd /data/prometheus

docker run -itd --name prometheus \
--restart=always \
-p 9090:9090 \
-v /data/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml \
lagou/prometheus:v2.21.0

docker logs -f prometheus

docker exec -it prometheus sh
date
```

测试prometheus

```
http://192.168.198.121:9090
```

cadvisor

用于监控docker容器。

github官网

```
https://github.com/google/cadvisor
```

docker官网

```
https://hub.docker.com/r/google/cadvisor
```

基础镜像

```
docker pull google/cadvisor:v0.31.0
```

运行镜像

```
docker run -d \
--restart=always \
--volume=/:/rootfs:ro \
--volume=/var/run:/var/run:ro \
--volume=/sys:/sys:ro \
--volume=/var/lib/docker/:/var/lib/docker:ro \
--volume=/dev/disk/:/dev/disk:ro \
--publish=8080:8080 \
--detach=true \
--name=cadvisor \
google/cadvisor:v0.31.0
```

测试cadvisor

```
http://192.168.198.121:8080/

http://192.168.198.121:8080/metrics
```

修改prometheus配置文件

增加一个监控docker的job。

```
  - job_name: "docker"
    static_configs:
      - targets: ['192.168.198.121:8080']
```

重启prometheus

需要耐性等待一会才能采集到docker的性能指标。

```
docker restart prometheus
```

grafana

- prometheus界面不是很友好，各种配置需要手写
- prometheus对docker、K8S监控都有成熟解决方案

grafana简介

Grafana是一个可视化面板（Dashboard），有着非常漂亮的图表和布局展示，功能齐全的度量仪表盘和图形编辑器。Grafana是一个跨平台的开源的度量分析和可视化工具，可以通过将采集的数据查询然后可视化的展示，并及时通知。它主要有以下六大特点：

- 展示方式：快速灵活的客户端图表，面板插件有许多不同方式的可视化指标和日志，官方库中具有丰富的仪表盘插件，比如热图、折线图、图表等多种展示方式
- 数据源：Graphite，InfluxDB，OpenTSDB，Prometheus，Elasticsearch，CloudWatch和KairosDB等
- 通知提醒：以可视方式定义最重要指标的警报规则，Grafana将不断计算并发送通知，在数据达到阈值时通过Slack、PagerDuty等获得通知
- 混合展示：在同一图表中混合使用不同的数据源，可以基于每个查询指定数据源，甚至自定义数据源
- 注释：使用来自不同数据源的丰富事件注释图表，将鼠标悬停在事件上会显示完整的事件元数据和标记
- 过滤器：Ad-hoc过滤器允许动态创建新的键/值过滤器，这些过滤器会自动应用于使用该数据源的所有查询

官网地址

```
https://grafana.com/
```

官网教程

```
https://grafana.com/docs/grafana/latest/installation/docker/
```

docker官网

```
https://hub.docker.com/r/grafana/grafana
```

基础镜像

```
主流版本
docker pull grafana/grafana:6.7.4


最新版本。WEB UI界面相对于6.X版本有很优雅的体验。感兴趣的小伙伴可以自行尝试
docker pull grafana/grafana:7.2.0
```

运行镜像

```
docker run -d \
--restart=always \
--name=grafana \
-p 3000:3000 \
grafana/grafana:6.7.4
```

测试grafana

```
http://192.168.198.121:3000/

默认用户名/密码：
admin/admin
第一次登陆提示要修改admin用户默认密码。
```

配置datasource

```
数据源选择prometheus
```

配置dashboard

grafana官方也为我们准备很多成熟的dashboard插件。我们可以通过下载官方插件管理我们的项目。

官网地址

```
https://grafana.com/grafana/dashboards
```

插件案例

```
Docker主机监控模板：  193
Linux主机监控模板：  9276

grafana监控docker主机有成熟的插件。直接点击import按钮，输入193插件即可
```

grafana中文版

github官网地址

```
https://github.com/WangHL0927/grafana-chinese
```

docker官网地址

```
https://hub.docker.com/r/w958660278/grafana-cn
```

基础镜像

```
docker pull w958660278/grafana-cn:6.7.3.0001-dev
```

rancher部署监控系统

使用rancher-server的应用商店部署prometheus。应用商店中的prometheus非rancher-server官方版本。

应用商店->社区共享

前置条件

请将agent-121节点部署的prometheus、cadvisor、grafana容器停止并删除，以免造成端口号冲突。

教程地址

```
https://github.com/infinityworks/Guide_Rancher_Monitoring
```

skywalking监控

官网介绍

skywalking是一个优秀的国产开源框架，2015年由个人吴晟（华为开发者）开源， 2017年加入apache孵化器。

skywalking是分布式系统的应用程序性能监视工具，专为微服务、云原生架构和基于容器化技术（docker、K8s、Mesos）架构而设计。skywalking是观察性分析平台和应用性能管理系统。提供分布式追踪、服务网格遥测分析、度量聚合和可视化一体化解决方案。

为什么需要服务追踪

在微服务架构下，由于进行了服务拆分，一次请求往往需要涉及多个服务，每个服务可能是由不同的团队开发，使用了不同的编程语言，还有可能部署在不同的机器上，分布在不同的数据中心。服务跟踪系统可以跟踪记录一次用户请求都发起了哪些调用，经过哪些服务处理，并且记录每一次调用所涉及的服务的详细信息，通过查看完整的调用链路，形成拓补图可以更加直观的了解业务，也可以针对当前的系统进行分析，是否需要扩容、优化接口、失败缓解，还有通过日志快速定位是调用失败的环节。

Apache Skywalking(Incubator)专门为微服务架构和云原生架构系统而设计并且支持分布式链路追踪的APM系统。Apache Skywalking(Incubator)通过加载探针的方式收集应用调用链路信息，并对采集的调用链路信息进行分析，生成应用间关系和服务间关系以及服务指标。Apache Skywalking (Incubating)目前支持多种语言，其中包括Java，.Net Core，Node.js和Go语言。

目前skywalking已经支持从6个可视化维度剖析分布式系统的运行情况。总览视图是应用和组件的全局视图，其中包括组件和应用数量，应用的告警波动，慢服务列表以及应用吞吐量；拓扑图从应用依赖关系出发，展现整个应用的拓扑关系；应用视图则是从单个应用的角度，展现应用的上下游关系，TopN的服务和服务器，JVM的相关信息以及对应的主机信息。服务视图关注单个服务入口的运行情况以及此服务的上下游依赖关系，依赖度，帮助用户针对单个服务的优化和监控；调用链展现了调用的单次请求经过的所有埋点以及每个埋点的执行时长；告警视图根据配置阈值针对应用、服务器、服务进行实时告警。

skywalking中默认使用的端口有8080、11800、12800，请保证这些端口未被占用 。

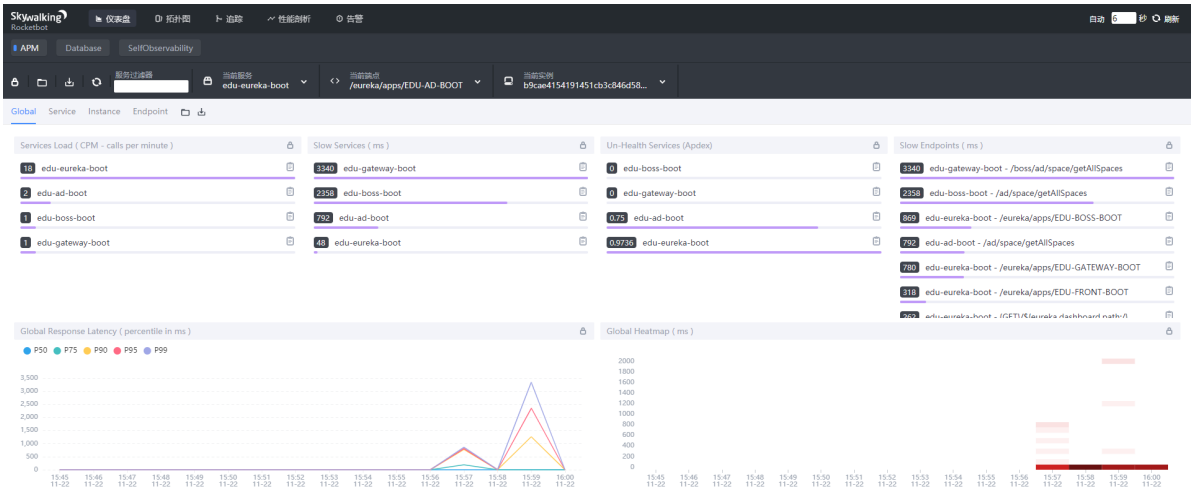
服务追踪的实际应用

捋清业务

我们都知道，在一般场景下，我们很难直观的了解系统的运行、业务的流程，因为传统的都是文字需求说明和枯燥的代码。通过链路追踪，可以根据调用链路来捋清楚服务间的调用关系，如果 API 设计符合规范，甚至可以直观的了解调用的服务作用。这对于刚刚接触系统的开发人员十分重要。

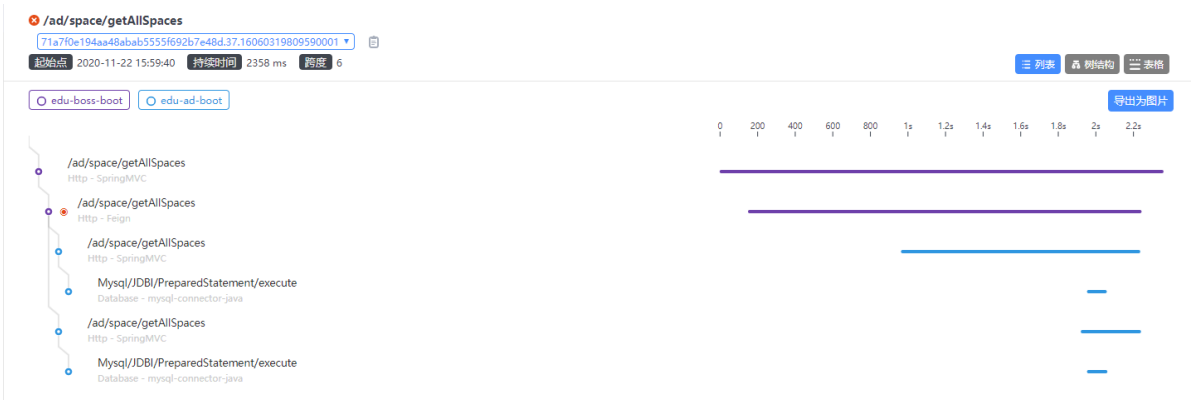
分析耗时

链路的基本功能，服务间的调用耗时记录，如果服务耗时过长，会影响整体的用户体验，甚至会抛出超时异常等，这样的情况在微服务架构中也是时有发生。



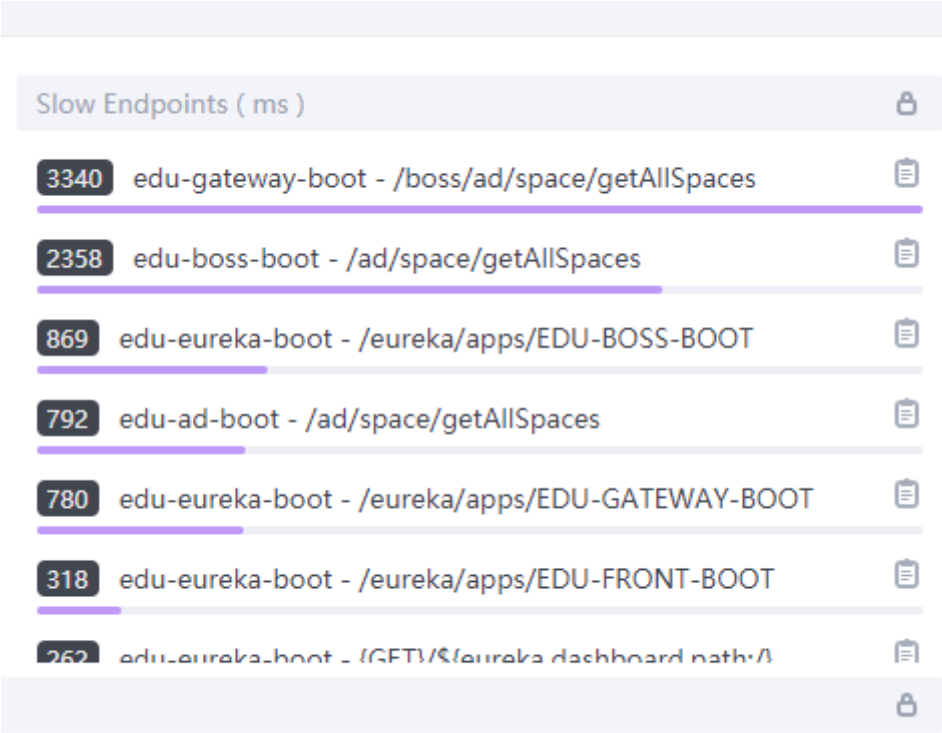
可视化错误

微服务调用链路发生错误，可以直观的显示查看，定位到被调用服务的接口，及时排查微服务中错误原因。



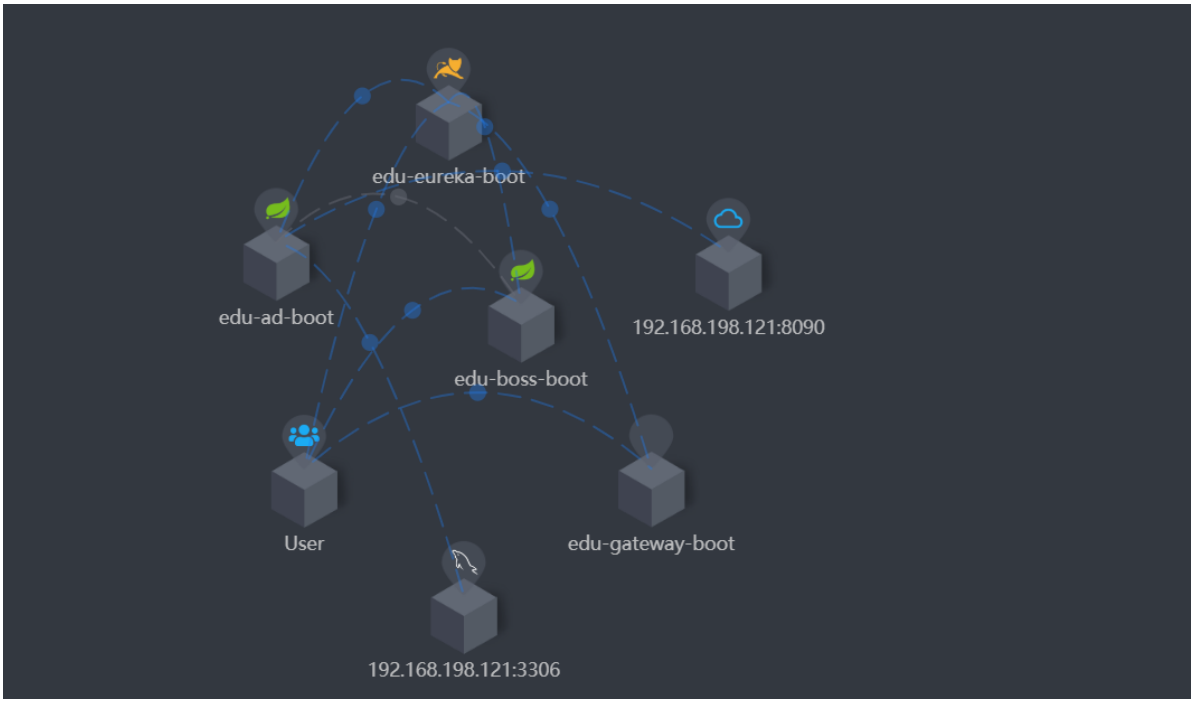
优化链路

显示完整的调用链路，根据业务分析合理性、可读性、健壮性，是否重复调用某一个服务，是否链路过长，有没有可以优化的，链路是否清晰。有些场景比较复杂，比如数据中心比较分散，服务分布在不同的数据中心，但是服务中心之间因为地域原因，距离远，延迟高，这可能不符合设计要求，因此就要根据链路来找到最近的数据中心，然后配置调用最近的数据中心的服务。



生成网络拓扑

通过服务追踪系统中记录的链路信息，可以生成一张系统的网络调用拓扑图，它可以反映系统都依赖了哪些服务，
以及服务之间的调用关系是什么样的，可以一目了然。除此之外，在网络拓扑图上还可以把服务调用的详细信息也标出来，也能起到服务监控的作用。



基础镜像

拉取镜像

尝鲜版

```
docker pull elasticsearch:7.9.0

默认es存储数据镜像
docker pull apache/skywalking-oap-server:8.1.0-es7

webUI界面镜像
docker pull apache/skywalking-ui:8.1.0

制作微服项目镜像
docker pull openjdk:8-alpine3.9
```

稳定版

```
docker pull elasticsearch:7.5.1

默认es存储数据镜像
docker pull apache/skywalking-oap-server:6.6.0-es7

webUI界面镜像
docker pull apache/skywalking-ui:6.6.0

制作微服项目镜像
docker pull openjdk:8-alpine3.9
```

备份镜像

```
最新版：
docker save apache/skywalking-oap-server:8.1.0-es7 apache/skywalking-ui:8.1.0
elasticsearch:7.9.0 -o skywalking8.1.0.tar

稳定版：
docker save apache/skywalking-oap-server:6.6.0-es7 apache/skywalking-ui:6.6.0
elasticsearch:7.5.1 -o skywalking6.6.0.tar
```

导入镜像

```
最新版：
docker load -i skywalking8.1.0.tar

稳定版：
docker load -i skywalking6.6.0.tar
```

下载压缩包

压缩包下载完成，将压缩包上传agent-121节点/opt目录下。

docker官网地址

```
https://hub.docker.com/r/apache/skywalking-oap-server
```

官网地址

```
http://skywalking.apache.org/
```

中文官网

```
http://skywalking.apache.org/zh/
```

下载地址

下载8.1.0中的tar包版本

```
https://skywalking.apache.org/zh/downloads/
```

8.1.0版本

可以通过官网链接地址，使用清华大学镜像地址进行下载。

```
https://mirrors.tuna.tsinghua.edu.cn/apache/skywalking/8.1.0/apache-skywalking-apm-8.1.0.tar.gz
```

前置条件

文件创建数

修改Linux系统的限制配置，将文件创建数修改为65536个：

1. 修改系统中允许应用最多创建多少文件等的限制权限。Linux默认来说，一般限制应用最多创建的文件是65535个。但是ES至少需要65536的文件创建数的权限。
2. 修改系统中允许用户启动的进程开启多少个线程。默认的Linux限制root用户开启的进程可以开启任意数量的线程，其他用户开启的进程可以开启1024个线程。必须修改限制数为4096+。因为ES至少需要4096的线程池预备。

```
vi /etc/security/limits.conf
#新增如下内容在limits.conf文件中
es soft nofile 65536
es hard nofile 65536
es soft nproc 4096
es hard nproc 4096
```

系统控制权限

修改系统控制权限，ElasticSearch需要开辟一个65536字节以上空间的虚拟内存。Linux默认不允许任何用户和应用程序直接开辟这么大的虚拟内存。

```
vi /etc/sysctl.conf

添加参数:新增如下内容在sysctl.conf文件中，当前用户拥有的内存权限大小
vm.max_map_count=262144

重启生效:让系统控制权限配置生效
sysctl -p
```

docker-compose安装

使用docker-compose方式安装skywalking。

github官网地址

根据官网地址进入docker目录中，修改docker-compose.yml文件内容

```
https://github.com/apache/skywalking
```

docker-compose文件

```
version: '3.3'
services:
  elasticsearch:
    image: elasticsearch:7.9.0
    container_name: elasticsearch
    restart: always
    ports:
      - 9200:9200
    environment:
      discovery.type: single-node
      TZ: Asia/Shanghai
    ulimits:
      memlock:
        soft: -1
        hard: -1
  oap:
    image: apache/skywalking-oap-server:8.1.0-es7
    container_name: oap
    depends_on:
      - elasticsearch
    links:
      - elasticsearch
    restart: always
    ports:
      - 11800:11800
```

```
      - 12800:12800
    environment:
      SW_STORAGE: elasticsearch7  # 指定ES版本
      SW_STORAGE_ES_CLUSTER_NODES: elasticsearch:9200
      TZ: Asia/Shanghai
  ui:
    image: apache/skywalking-ui:8.1.0
    container_name: ui
    depends_on:
      - oap
    links:
      - oap
    restart: always
    ports:
      - 8899:8080
    environment:
      SW_OAP_ADDRESS: oap:12800
      TZ: Asia/Shanghai
```

启动服务

```
docker-compose up -d
```

查看启动

```
docker-compose ps
```

测试ui界面

在window系统中使用google浏览器访问skywalking-ui界面

```
elasticsearch启动时间比较长，需要耐心等待几分钟
http://192.168.198.120:8899/
```

H2方式安装

数据存储在内存中,容易造成数据丢失。开发环境临时使用还可以，不推荐生产环境使用。

基础镜像

```
docker pull apache/skywalking-oap-server:latest

前端UI可以下载8.X或者6.X版本
apache/skywalking-ui:6.6.0
```

运行镜像

```
docker run -itd --name skywalking  -p 11800:11800 -p 12800:12800 --restart
always apache/skywalking-oap-server:latest

docker run -itd --name skywalking-ui  -p 8080:8080 --link skywalking:skywalking
-e SW_OAP_ADDRESS=skywalking:12800 --restart always apache/skywalking-ui:6.6.0
```

整合springboot工程

springboot项目

pom.xml文件

新增跳过单元测试plugin配置

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.3.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.lagou</groupId>
    <artifactId>skywalkingdemo1</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>skywalkingdemo1</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
```

```
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <build>
        <finalName>skywalkingdemo1</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <!--跳过单元测试-->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <configuration>
                    <skip>true</skip>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

application.yml文件

为防止与其他项目端口号冲突，将项目端口号更改为9988

```
server:
  port: 9988
```

HelloController

```
@RestController
public class HelloController {
    //正常访问接口
    @RequestMapping("/sayBoot")
    public String sayBoot(){
        return "Hello skywalking!";
    }

    //异常访问接口
    @RequestMapping("/exception")
    public String exception(){
        int i = 1/0;
        return "Hello skywalking!";
    }
}
```

本地测试项目

```
idea开发工具中启动项目，进行测试
http://localhost:9988/sayBoot
```

打包项目

```
mvn clean package
```

容器化部署微服项目

准备工作

```
新建自定义镜像目录
mkdir -p /data/skywalking

将skywalkingdemo1.jar复制到/data/skywalking目录中
```

配置agent

```
解压缩压缩包：
tar zxf apache-skywalking-apm-8.1.0.tar.gz

rm -rf apache-skywalking-apm-8.1.0.tar.gz
mv apache-skywalking-apm-bin/ skywalking

将agent目录复制到/data/skywalking备用
cd /opt/skywalking
cp -r /opt/skywalking/agent /data/skywalking
cd /data/skywalking/agent/config
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker skywalking springboot "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories
```

```
# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY agent/ /opt/skyagent/
COPY skywalkingspringbootdemo1.jar /app.jar

EXPOSE 9988


ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=skywalkingspringbootdemo1","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

制作镜像

```
cd /data/skywalking
docker build --rm -t lagou/skywalkingdemo1:1.0 .
```

运行镜像

```
docker run -itd --name skywalkingdemo1 -p 9988:9988 lagou/skywalkingdemo1:1.0
```

测试项目

```
docker logs -f skywalkingdemo1

http://192.168.198.121:9988/sayBoot

docker stop skywalkingdemo1
docker rm skywalkingdemo1
```

整合SpringCloud工程

整合其实很简单，不需要引入依赖，也不需要添加任何代码，我们只需要在启动jar包时配置参数即可。edu-bom项目在本章节中采用docker-compose方式进行一键式部署。

基础镜像

拉取镜像

```
docker pull openjdk:8-alpine3.9
docker pull mysql:5.7.31
```

项目结构

```
cd /data
mkdir -p edu-bom

cd edu-bom
mkdir -p mysql edu-eureka-boot edu-config-boot edu-gateway-boot edu-front-boot
edu-boss-boot edu-ad-boot

cd mysql
mkdir -p docker-entrypoint-initdb.d lagou
```

配置agent

```
cd /data
tar zxf apache-skywalking-apm-8.1.0.tar.gz
rm -rf apache-skywalking-apm-8.1.0.tar.gz

cp -r /data/apache-skywalking-apm-bin/agent/ /data/edu-bom/edu-ad-boot/
cp -r /data/apache-skywalking-apm-bin/agent/ /data/edu-bom/edu-boss-boot/
cp -r /data/apache-skywalking-apm-bin/agent/ /data/edu-bom/edu-config-boot/
cp -r /data/apache-skywalking-apm-bin/agent/ /data/edu-bom/edu-eureka-boot/
cp -r /data/apache-skywalking-apm-bin/agent/ /data/edu-bom/edu-front-boot/
cp -r /data/apache-skywalking-apm-bin/agent/ /data/edu-bom/edu-gateway-boot/
```

Dockerfile

edu-eureka-boot

```
COPY agent/ /opt/skyagent/

ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-eureka-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

完整文件

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-eureka-boot  "laosiji@lagou.com"
# 修改源
```

```
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai
COPY agent/ /opt/skyagent/
COPY edu-eureka-boot.jar app.jar
EXPOSE 8761
ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-eureka-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

edu-config-boot

```
COPY agent/ /opt/skyagent/

ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-config-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

完整文件

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-eureka-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai
COPY agent/ /opt/skyagent/
COPY edu-config-boot.jar app.jar
EXPOSE 8090
ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-config-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

edu-boss-boot

```
COPY agent/ /opt/skyagent/

ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-boss-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

完整文件

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-eureka-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai
COPY agent/ /opt/skyagent/
COPY edu-boss-boot.jar app.jar
EXPOSE 8082
CMD sleep 30
ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-boss-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

edu-gateway-boot

```
COPY agent/ /opt/skyagent/

ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-gateway-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

完整文件

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-eureka-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
```

```
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai
COPY agent/ /opt/skyagent/
COPY edu-gateway-boot.jar app.jar
EXPOSE 9001
CMD sleep 60
ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-gateway-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

edu-ad-boot

```
COPY agent/ /opt/skyagent/

ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-ad-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

完整文件

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-eureka-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai
COPY agent/ /opt/skyagent/
COPY edu-ad-boot-impl.jar app.jar
EXPOSE 8001
CMD sleep 20
ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-ad-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

edu-front-boot

```
COPY agent/ /opt/skyagent/

ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-front-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

完整文件

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker edu-eureka-boot  "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-stable/community/" >>
/etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai
COPY agent/ /opt/skyagent/
COPY edu-front-boot.jar app.jar
EXPOSE 8081
CMD sleep 30
ENTRYPOINT ["java","-Xmx512m","-javaagent:/opt/skyagent/skywalking-agent.jar","-
Dskywalking.agent.service_name=edu-front-boot","-
Dskywalking.collector.backend_service=192.168.198.120:11800","-jar","/app.jar"]
```

docker-compose.yml

```
version: '3'
services:
  lagou-mysql:
    build:
      context: ./mysql
    environment:
      MYSQL_ROOT_PASSWORD: admin
    restart: always
    container_name: lagou-mysql
    volumes:
    - /data/edu-bom/mysql/lagou:/var/lib/mysql
    image: lagou/mysql:5.7
    ports:
      - 3306:3306
  lagou-eureka:
    build:
      context: ./edu-eureka-boot
    restart: always
    ports:
      - 8761:8761
    container_name: edu-eureka-boot
    hostname: edu-eureka-boot
    image: lagou/edu-eureka-boot:1.0
    depends_on:
      - lagou-mysql
  lagou-config:
```

```
    build:
      context: ./edu-config-boot
    restart: always
    ports:
    - 8090:8090
    container_name: edu-config-boot
    hostname: edu-config-boot
    image: lagou/edu-config-boot:1.0
    depends_on:
      - lagou-eureka
  lagou-ad:
    build:
      context: ./edu-ad-boot
    restart: always
    ports:
      - 8001:8001
    container_name: edu-ad-boot
    hostname: edu-ad-boot
    image: lagou/edu-ad-boot:1.0
    depends_on:
      - lagou-config
  lagou-boss:
    build:
      context: ./edu-boss-boot
    restart: always
    ports:
    - 8082:8082
    container_name: edu-boss-boot
    hostname: edu-boss-boot
    image: lagou/edu-boss-boot:1.0
    depends_on:
      - lagou-ad

  lagou-front:
    build:
      context: ./edu-front-boot
    restart: always
    ports:
      - 8081:8081
    container_name: edu-front-boot
    hostname: edu-front-boot
    image: lagou/edu-front-boot:1.0
    depends_on:
      - lagou-boss

  lagou-gateway:
    build:
      context: ./edu-gateway-boot
    restart: always
    ports:
      - 9001:9001
    container_name: edu-gateway-boot
    hostname: edu-gateway-boot
    image: lagou/edu-gateway-boot:1.0
    depends_on:
      - lagou-boss
```

制作镜像

```
制作镜像
docker-compose build

docker-compose up -d

删除镜像
docker-compose down -v --rmi all
```

访问项目

```
http://192.168.198.121:8761/

http://192.168.198.121:8001/ad/space/getAllSpaces

http://192.168.198.121:9001/boss/ad/space/getAllSpaces
```

总结

skywalking就介绍到这里，本章节仅仅只是入门，简单使用Skywalking，实际上里面还有很多功能没有介绍，有兴趣的同学可以按照上面的教程安装部署，然后自己探索一下。在现在微服务架构比较流行的环境下，如果没有一个调用链追踪框架，会导致很难排查线上服务调用的问题。skywalking是目前发展势头最快的技术框架的技术框架，因为对代码是无侵入性的，所以目前很多公司都采用skywalking。

客户端agent相关配置说明

```
# 命名空间，用于隔离跨进程传播的header。如果进行了配置，header将为HeaderName:Namespace.
# agent.namespace=${SW_AGENT_NAMESPACE:default-namespace}


# 展示界面中现实服务名称
agent.service_name=${SW_AGENT_NAME:lizz-gw}

# 每3秒采样道数默认情况下，负或零表示关闭
# agent.sample_n_per_3_secs=${SW_AGENT_SAMPLE:-1}

# 鉴权是否开启取决于后端的配置，可查看application.yml的详细描述。对于大多数的场景，需要后端对鉴权进行扩展。目前仅实现了基本的鉴权功能。
# agent.authentication = ${SW_AGENT_AUTHENTICATION:xxxx}

# 单个线段中的最大跨距量。
# 通过这个配置项，SkyWalking可以估计应用程序内存开销。
# agent.span_limit_per_segment=${SW_AGENT_SPAN_LIMIT:150}

# 如果段的操作名称以这些后缀结尾，则忽略这些段。
#
agent.ignore_suffix=${SW_AGENT_IGNORE_SUFFIX:.jpg,.jpeg,.js,.css,.png,.bmp,.gif,.ico,.mp3,.mp4,.html,.svg}

# 如果为true，则SkyWalking代理将在“/debugging”文件夹中保存所有检测到的类文件。
# SkyWalking可能会要求这些文件，以解决兼容问题。
# agent.is_open_debugging_class = ${SW_AGENT_OPEN_DEBUG:true}

# 如果为true，SkyWalking代理将把所有检测到的类文件缓存到内存或磁盘文件中（由类缓存模式决定），
# 允许其他javaagent增强那些由SkyWalking agent增强的类。
# agent.is_cache_enhanced_class = ${SW_AGENT_CACHE_CLASS:false}

# 插入指令的类缓存模式：内存或文件
# 内存：将类字节缓存到内存中，如果插入指令的类太多或太大，则可能会占用更多内存
# 文件：在“/class cache”文件夹中缓存类字节，当应用程序退出时自动清理缓存的类文件
# agent.class_cache_mode = ${SW_AGENT_CLASS_CACHE_MODE:MEMORY}

# 操作名称最大长度
# 注意，在目前的实践中，我们不建议长度超过190。
# agent.operation_name_threshold=${SW_AGENT_OPERATION_NAME_THRESHOLD:150}

# 如果为true，则当用户创建新的配置文件任务时，skywalking代理将启用配置文件。否则禁用配置文件。
# profile.active=${SW_AGENT_PROFILE_ACTIVE:true}

# 并行监视器段计数
# profile.max_parallel=${SW_AGENT_PROFILE_MAX_PARALLEL:5}

# 最大监视段时间（分钟），如果当前段监视时间超出限制，则停止它。
# profile.duration=${SW_AGENT_PROFILE_DURATION:10}

# 最大转储线程堆栈深度
# profile.dump_max_stack_depth=${SW_AGENT_PROFILE_DUMP_MAX_STACK_DEPTH:500}

# 快照传输到后端缓冲区的大小
```

```
#
profile.snapshot_transport_buffer_size=${SW_AGENT_PROFILE_SNAPSHOT_TRANSPORT_BUF
FER_SIZE:50}

# skywalking后端服务地址。
collector.backend_service=${SW_AGENT_COLLECTOR_BACKEND_SERVICES:127.0.0.1:11800}

# 日志文件名
logging.file_name=${SW_LOGGING_FILE_NAME:skywalking-api.log}

# 日志记录级别
logging.level=${SW_LOGGING_LEVEL:WARN}

# 日志文件存储目录
# logging.dir=${SW_LOGGING_DIR:""}

# 日志文件最大值，default: 300 * 1024 * 1024 = 314572800
# logging.max_file_size=${SW_LOGGING_MAX_FILE_SIZE:314572800}

# 最大历史记录日志文件。当发生滚动时，如果日志文件超过这个数字，
# 然后删除最旧的文件。默认情况下，负数或零表示禁用。
# 如果不限制个数可能到只日志文件过大，磁盘爆满。
logging.max_history_files=${SW_LOGGING_MAX_HISTORY_FILES:5}

# mysql 插件配置
# plugin.mysql.trace_sql_parameters=${SW_MYSQL_TRACE_SQL_PARAMETERS:false}

# Kafka 插件配置
# plugin.kafka.bootstrap_servers=${SW_KAFKA_BOOTSTRAP_SERVERS:localhost:9092}
```