云原生大作业-说明文档-第一组

组员信息

姓名	学号	邮箱	备注
闫慧渊（组长）	201220111	1084683108@qq.com	构建项目，实现接口，限流功能，dockerfile，k8s编排文件，jenkins流水线集成、部署
段霁峰	201250138	1010612107@qq.com	项目说明文档，prometheus+grafana监控，压力测试，统一限流，CRD
陈沁羽	201250224	65421357@qq.com	

本组使用github仓库、微信群进行线上合作，github仓库链接：https://github.com/HuiyuanYan/cloud_native_proj

版本管理

操作人	操作内容	版本号	时间
闫慧渊	创建github仓库，实现两点基础功能点（log_yan.pdf中有详细描述）	/	7.12
闫慧渊	创建DockerFile和K8s编排文件	/	7.13
闫慧渊	修改完善项目文件架构，jenkins流水线部署	/	7.30
段霁峰	创建md格式的项目说明文档（给队友看版），添加功能点“prometheus+grafana的可视化监控”	v0.0.1	7.30
段霁峰	把log_yan.pdf整合进说明文档，添加“压测并观察监控数据”模块内容，把闫慧渊的“jenkins流水线部署”加入文档，做了统一限流功能（bonus）	v0.0.2	7.31

目录

作业要求

开发一个 Spring Boot 应用，并使用云原生功能

功能要求，见 2022-基于云原生技术的软件开发 - 大作业.pdf

- **账号及密码的注意事项（一组）**

1. Portal 地址为 http://172.29.4.36:3000/ ，Jenkins，Harbor等服务可直接从此页面点击进入；
2. Jenkins Harbor Grafana 的账号为nju01，密码为nju012022； （一组）
3. Jenkins 创建项目的名称必须以组名（001 002 003...）开头，不然创建项目 点击确定时会报错；比如一组的项目名为 prometheus-test-demo，项目名可以是 001-prometheus-test-demo，或 001prometheus-test-demo
4. Jenkins 编写 pipeline scripts 时尽量使用 slave 节点；
5. 登录 Harbor 镜像仓库时使用自己组的账号密码，push 镜像到 Harbor 镜像仓库时使用自己组的镜像空间，比如harbor.edu.cn/nju01/dao-2048:latest，镜像空间指中间nju01 字段的内容，分配的镜像空间名称和账号名相同，可自己登录 Harbor 查看；harbor 地址为 harbor.edu.cn，本地 docker login harbor.edu.cn 时，需要在/etc/hosts 目录下加上解析 172.29.4.26 harbor.edu.cn
6. K8s 集群登录账号为 nju01， 用户密码为nju012022；登录方式为 ssh nju01@172.29.4.18
7. K8s 集群 每组只有一个租户的访问及编辑权限，本组租户名为 nju01，查看部署应用可使用 kubectl get pods -n nju01

功能实现

1 基础功能

1.1 构建项目 + 实现接口（闫慧渊）

1. 使用idea插件`Spring boot initializer`初始化一个名为`cloud-native-proj`的springboot项目。
2. 在`src/main/resource/application.properties`文件下修改web服务端口：

```
server.port=8080
```

3. 新建`src/main/java/com/example/cloud_native_proj/controller`文件夹，在该文件夹下新建`UserController.java`文件，并自定义`UserController`类，返回基本的`json`信息。（返回json信息需要导入`alibaba fastjson`依赖）

```
<!--json工具-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.15</version>
</dependency>
```

导入json依赖

```
public class UserController {
    //@ResponseBody
    @RequestMapping("/hello")
    //每秒并发量，设置为10
    @CurrentLimiter(QPS = 10)
    public Object sayHello() {
        JSONObject jsonObject=new JSONObject();

        jsonObject.put("code",200);
        jsonObject.put("group","1");
        jsonObject.put("member","Qinyu Chen,Jifeng Duan,Huiyuan Yan");
        jsonObject.put("msg","Welcome!");
        return jsonObject.toString();
    }
```

UserController类

1.2 实现限流功能（闫慧渊）

1.2.1 操作过程

采用`current limiting`工具实现了接口限流工作，但还没有实现多pod统一限流。

1. 引入依赖

```xml
<!--接口限流，采用Current Limiting 工具进行-->
<dependency>
    <groupId>cn.yueshutong</groupId>
    <artifactId>spring-boot-starter-current-limiting</artifactId>
    <version>0.0.8.RELEASE</version>
</dependency>
```

2. 在 `src` 根文件夹下新建 `application.yaml` 配置文件并写入流量控制的相关配置内容：

```yaml
current:
  limiting:
    #开启全局限流
    enabled: false
    #开启注解限流,可使注解失效
    part-enabled: true
    #每秒并发量 这里的qps是全局限流开启的时候的值,如果使用注解在注解里设置QPS值
    qps: 100
    #开启快速失败,可切换为阻塞
    fail-fast: true
    #系统启动保护时间为0
    initial-delay: 0
```

3. 在 `UserController` 类下添加注解：

```java
public class UserController {
    //@ResponseBody
    @RequestMapping("/hello")
    //每秒并发量，设置为10
    @CurrentLimiter(QPS = 10)
    public Object sayHello() {
```

为测试方便，每秒最大限流为10次（作业最终提交时可修改成100次）。

4. 新建 `src/main/java/com/example/cloud_native_proj/co/FC` 文件夹(Flow Control)，在该文件夹下新建 `MyCurrentLimitHandler.java` 文件，并自定义处理类处理超时函数。

```java
@Component
public class MyCurrentLimitHandler  implements CurrentAspectHandler {
    @Override
    public Object around(ProceedingJoinPoint pjp, CurrentLimiter rateLimiter)  {

        //限流的返回数据可以自己根据需求场景设计

        JSONObject jsonObject=new JSONObject();

        jsonObject.put("code",429);
        jsonObject.put("msg","Too Too many requests");
        return jsonObject.toString();
        //return "Too many!!!";
    }
}
```

1.2.2 功能演示

1. 编译运行

   在项目文件夹下运行命令：

   ```
   mvn clean install package '-Dmaven.test.skip=true'
   ```

   ```
   java -jar target/cloud_native_proj-0.0.1-SNAPSHOT.jar
   ```

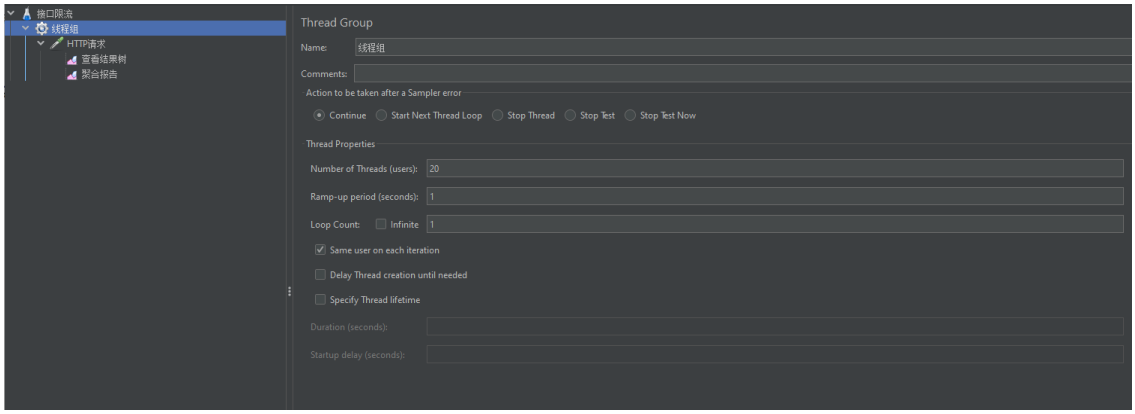
   构建并运行项目。

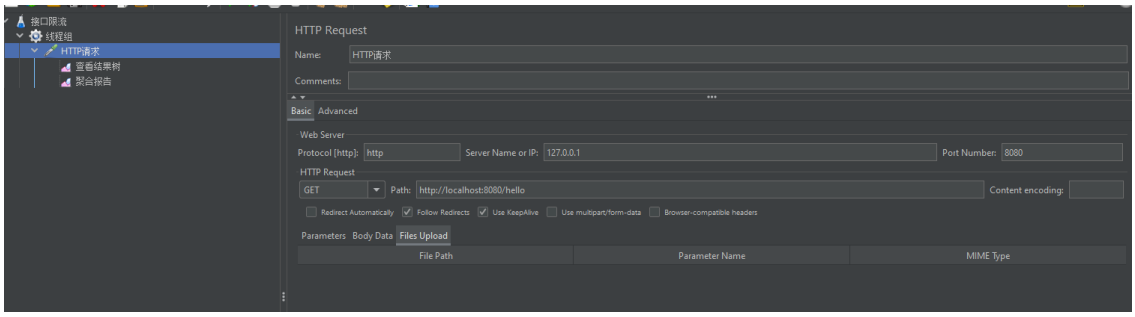
2. 打开浏览器输入网址：`http://localhost:8080/hello`（注意mapping有`/hello`）查看内容：

   {"msg":"Welcome!","code":200,"member":"Qinyu Chen,Jifeng Duan,Huiyuan Yan","group":"1"}

3. 用`Apache Jmeter`工具对接口进行压力测试。

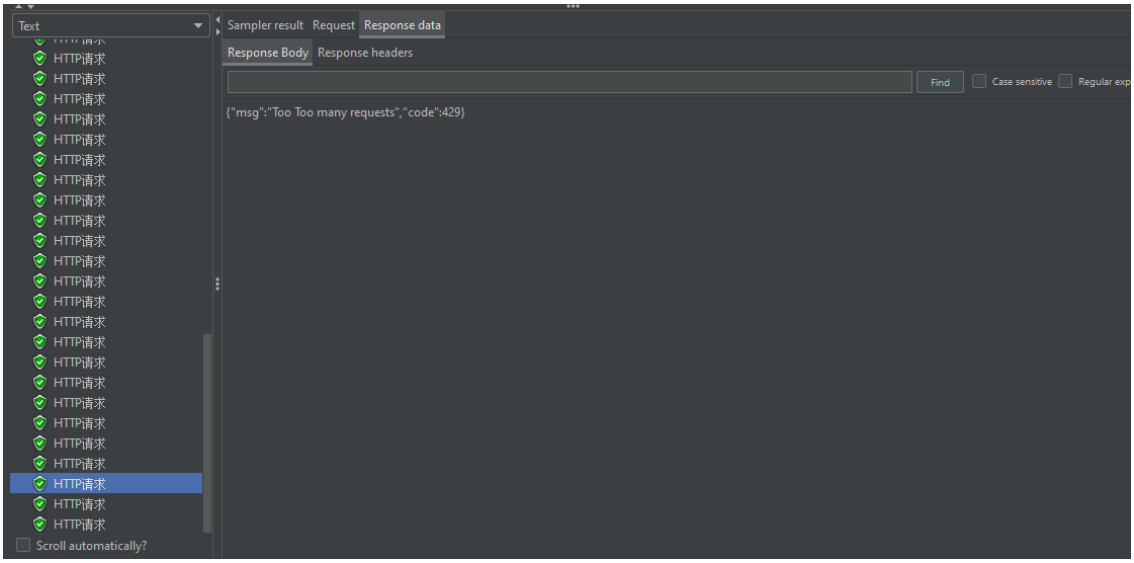
   配置如下：

   会生成jmx文件。

   然后点击右上角绿色箭头运行，在结果树中查看情况。



平均一秒内每十次左右访问就会出现访问次数过多的返回值。

1.3 统一限流（bonus功能）（段霁峰）

可以使用Tomcat容器限流， conf/server.xml 配置中规定了最大线程数

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000" #超时时间
           maxThreads="100" # 这个就是最大线程数，请求的并发大于maxThreads时，请求就会排队执行
           redirectPort="8080" />
```

2 DevOps 要求

2.1 Dockerfile，用于构建镜像（闫慧渊）

在target目录下创建DockerFile并写入如下内容：

```
FROM openjdk:8-jre-alpine
#作者
MAINTAINER Group1 1084683108@qq.com
#复制文件并命名，ADD支持远程获取URL资源
ADD cloud_native_proj-0.0.1-SNAPSHOT.jar /cloud_native_proj.jar
#VOLUME ，VOLUME 指向了一个/tmp的目录，由于 Spring Boot 使用内置的Tomcat容器，Tomcat 默认使用/tmp作为工作目录。这个命令的效果是：
#在宿主机的/var/lib/docker目录下创建一个临时文件并把它链接到容器中的/tmp目录
#VOLUME /tmp
#声明端口
EXPOSE 8080
ENTRYPOINT ["java","-jar","/cloud_native_proj.jar"]
#命令：docker build -t cloud_native_proj .
#docker run --name cloud_native_proj -p 8080:8080 -d cloud_native_proj
```

运行上述最后两行注释命令，即可在本地ip的8080端口使用Web服务。

注意DockerFile第一行**必须是** `FROM xxx`。

2.2 Kubernetes 编排文件（闫慧渊）

在根目录下创建`proj_deploy.yaml`和`proj_svc.yaml`文件，分别写入如下内容：

```
#proj_deploy.yaml
apiVersion: apps/v1   # 1.9.0 之前的版本使用 apps/v1beta2，可通过命令 kubectl api-versions 查看
kind: Deployment    #指定创建资源的角色/类型
metadata:    #资源的元数据/属性
  name: cloud-native-proj #资源的名字，在同一个namespace中必须唯一
  namespace: cloud-native-namespace
spec:
  replicas: 2    #副本数量2
  selector:      #定义标签选择器
    matchLabels:
      app: cloud-native-proj
  template:      #这里Pod的定义
    metadata:
      labels:    #Pod的label
        app: cloud-native-proj
    spec:        # 指定该资源的内容
      containers:
        - name: cloud-native-proj      #容器的名字
          image: cloud_native_proj  #容器的镜像地址
          imagePullPolicy: Never
```

```
#proj_svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: cloud-native-proj-svc
  namespace: cloud-native-namespace
spec:
  type: NodePort
  selector:
    app: cloud-native-proj
  ports:
    - nodePort: 30001 # host's port,集群外部访问端口
      port: 8888      # service's port,集群内部访问端口
      targetPort: 8080  # target pod's port,服务最终端口，所有流量流至该端口，和DokcerFile中EXPOSE端口一致
      #按照k8s_cmd.sh的命令执行完后，即可通过本机ip:30001端口访问Web服务
```

用于创建环境和服务。

然后依次执行下面的命令（我已把命令放在文件`k8s_cmd.sh`中）

```
kubectl create namespace cloud-native-namespace
# kubectl get namespace
kubectl apply -f proj_deploy.yaml
# kubectl get deployment -n cloud-native-namespace
kubectl create -f proj_svc.yaml
# kubectl get svc -n cloud-native-namespace
```

分别是创建名为`cloud-native-namespace`的命名空间和在该空间下（yaml脚本中限定）创建`deployment`和`service`。

带注释的命令是用来检查每一步创建是否成功的。

上述步骤都执行成功后，即可通过本机ip的`30001`端口访问Web服务。

2.3 持续集成流水线 + 持续部署流水线（闫慧渊）

2.3.1 Pipeline脚本

```
pipeline{
    agent none
    stages {
        stage('Clone Code') {
            agent {
                label 'master'
            }
            steps {
                echo "1.Git Clone Code"
                sh 'curl "http://p.nju.edu.cn/portal_io/logout"'
                sh 'curl "http://p.nju.edu.cn/portal_io/login?
username=XXX&password=XXX"'
                git url: "https://gitee.com/huiyuanyan/cloud_native_proj.git"
            }
        }

        stage('Maven Build') {
            agent {
                docker {
                    image 'maven:latest'
                    args ' -v /home/nju01:/home/nju01'
                }
            }
            steps {
                echo "2.Maven Build Stage"
                sh 'mvn clean install package \'-Dmaven.test.skip=true\''
            }
        }

        stage('Image Build') {
            agent {
                label 'master'
            }
            steps {
                echo "3.Image Build Stage"
                sh 'docker build -f Dockerfile --build-arg
jar_name=target/cloud_native_proj-0.0.1-SNAPSHOT.jar -t
cloud_native_proj:${BUILD_ID} .'
                sh 'docker tag cloud_native_proj:${BUILD_ID}
harbor.edu.cn/nju01/cloud_native_proj:${BUILD_ID}'
            }
        }

        stage('Push') {
            agent {
```

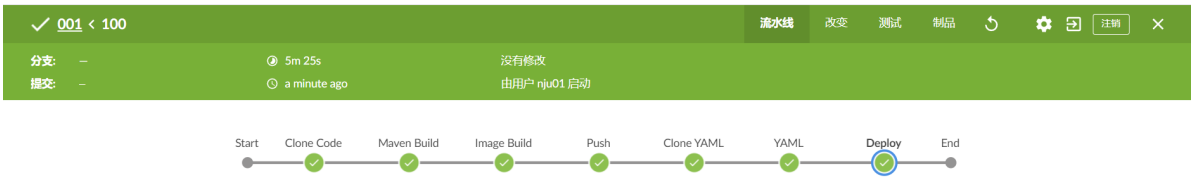
```
                label 'master'
            }
            steps {
                echo "4.Push Docker Image Stage"
                sh "docker login --username=nju01 harbor.edu.cn -p nju012022"
                sh 'docker push
harbor.edu.cn/nju01/cloud_native_proj:${BUILD_ID}'
            }
        }
    }
}
node('slave'){
    container('jnlp-kubectl'){
        stage('Clone YAML'){
            echo "5.Git Clone YAML to Slave"
            sh 'curl "http://p.nju.edu.cn/portal_io/logout"'
            sh 'curl "http://p.nju.edu.cn/portal_io/login?
username=XXX&password=XXX"'
            git url: "https://gitee.com/huiyuanyan/cloud_native_proj.git"
        }
        stage('YAML'){
            echo"6.Change YAML File Stage"
            sh 'ls'
            sh 'ls /home'
            sh 'sed -i "s#{VERSION}#${BUILD_ID}#g"   proj_deploy.yaml'
            sh 'sed -i "s#{VERSION}#${BUILD_ID}#g"   proj_svc.yaml'
        }
        stage('Deploy'){
            echo "7.Deploy To K8s Stage"
            sh "kubectl apply -f proj_deploy.yaml -n nju01"
            sh "kubectl apply -f proj_svc.yaml -n nju01"
        }
    }
}
```

2.3.2 流水线部署成功截图展示

- k8s宿主机部署成功

```
nju01@172.29.4.18's password:
Last login: Sun Jul 31 03:03:02 2022
[nju01@host-172-29-4-18 ~]$ kubectl get svc -n nju01
NAME                  TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
cloud-native-proj-svc   NodePort   10.106.226.212   <none>        8888:30002/TCP   8h
[nju01@host-172-29-4-18 ~]$ kubectl get deploy -n nju01
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
cloud-native-proj   0/2     2            0           20h
```

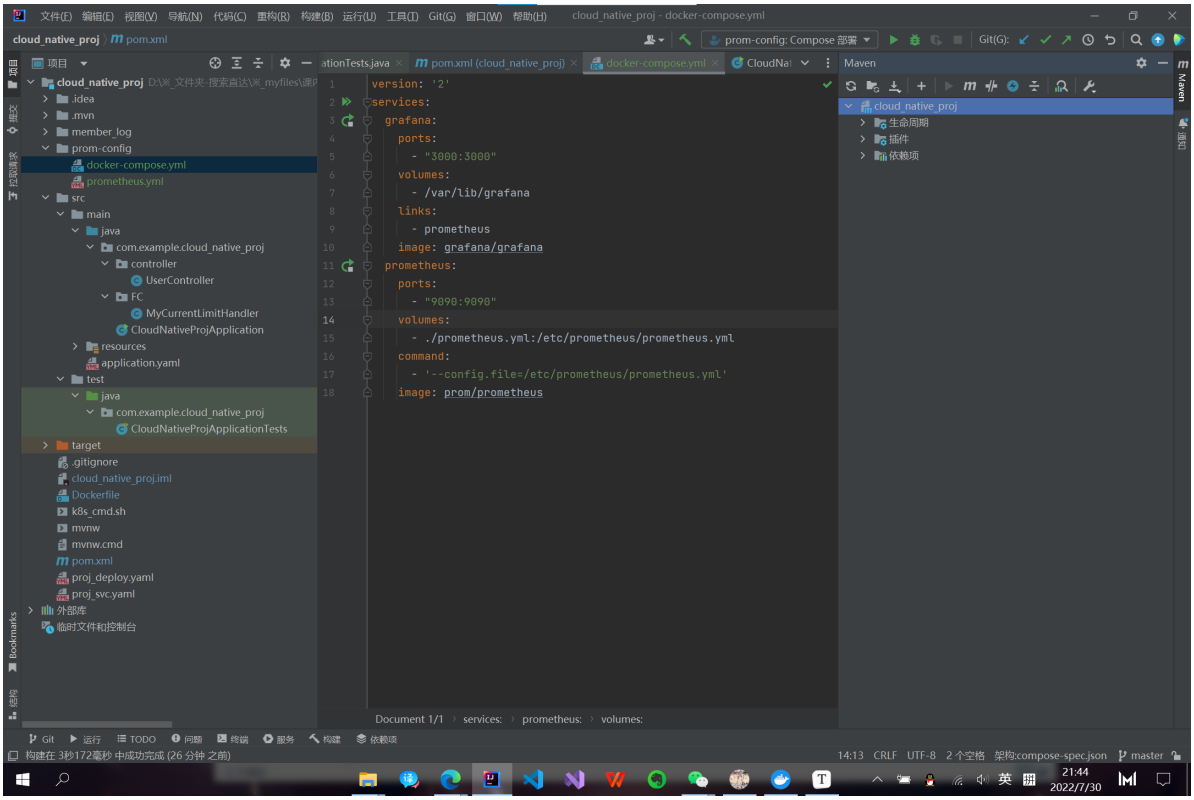
- Jenkins平台部署成功



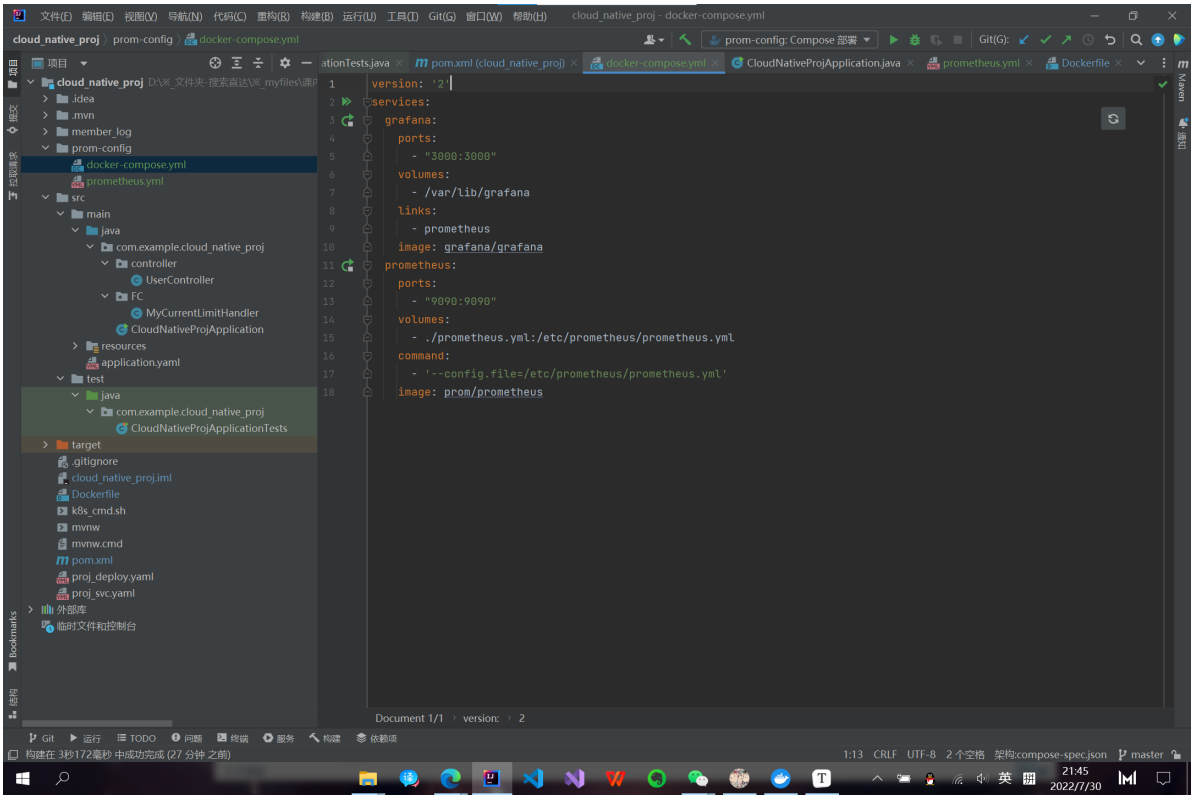
3 扩容场景

3.1 prometheus+grafana的可视化监控（段霁峰）

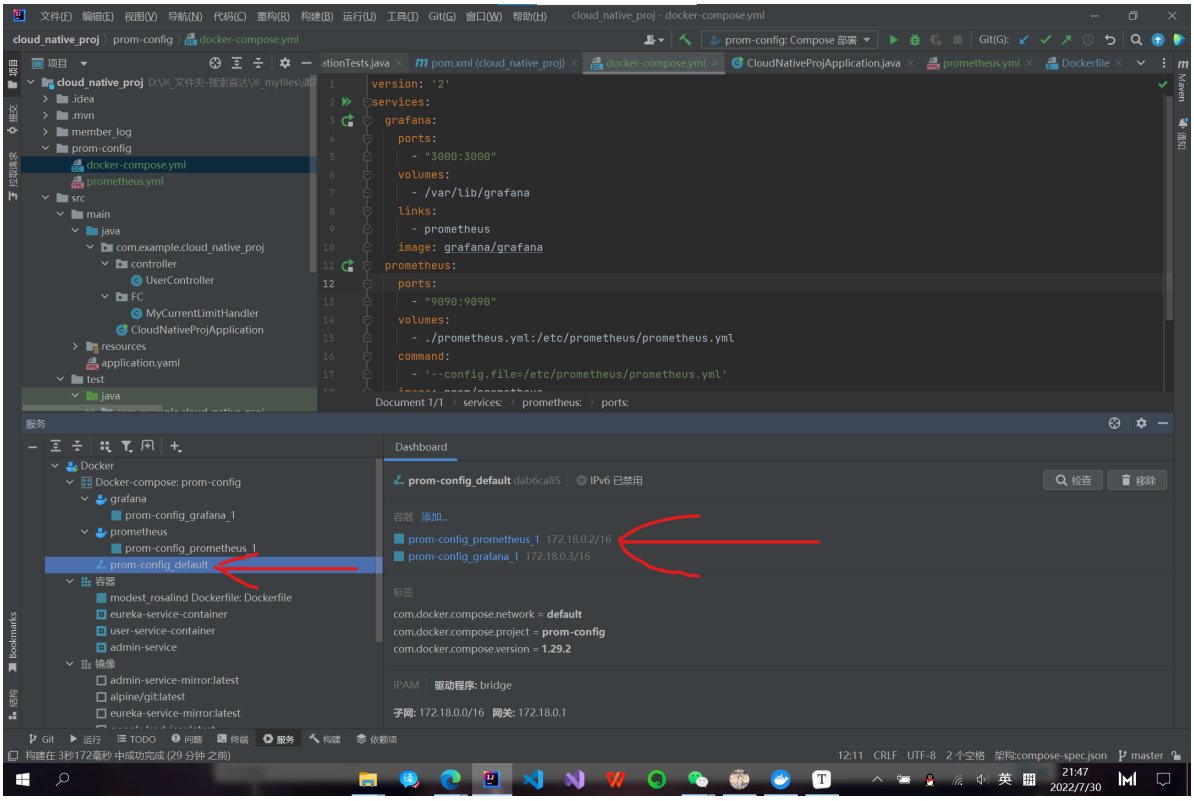
拿到项目首先更新下maven库(我加了一点新的依赖)



更新完之后点击运行docker-compose文件

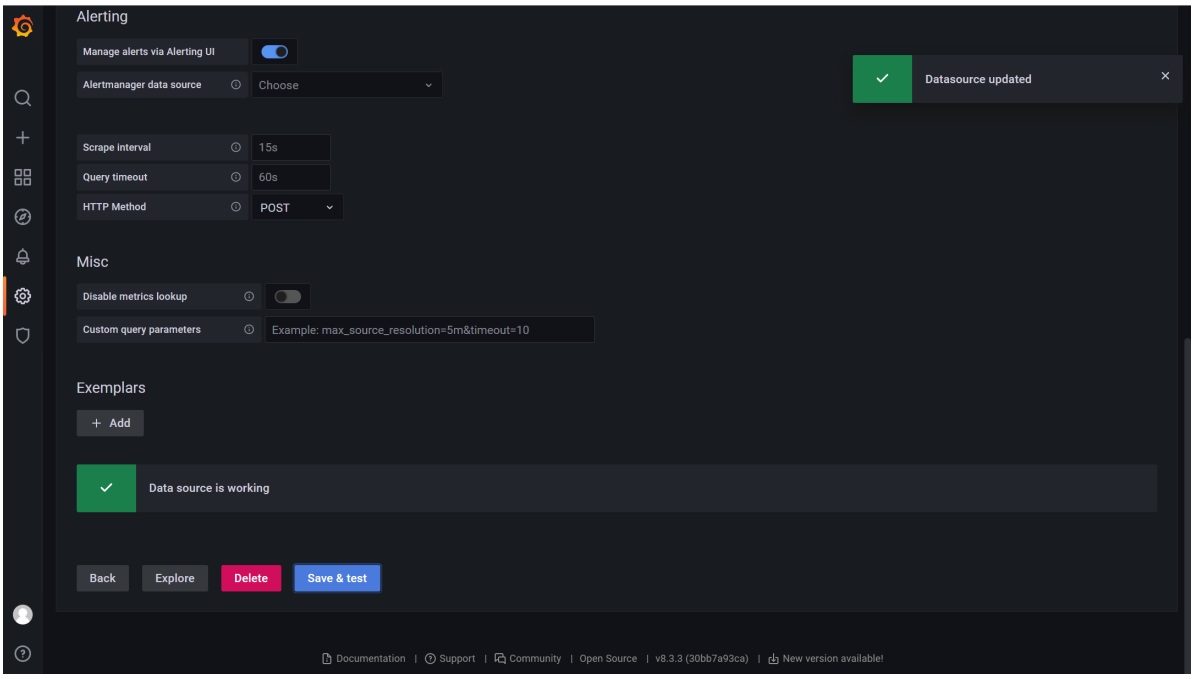


可以看到生成了两个容器并放到了一个网络下（左边的红色箭头），左边的红色箭头表明了grafana应该从哪里获取 数据



访问`127.0.0.1:3000`，也就是grafana界面，默认的用户名密码都是admin，**系统会让改密码，新密码也设为admin，方便记**

添加数据源（这里用到了上面的地址），添加成功！

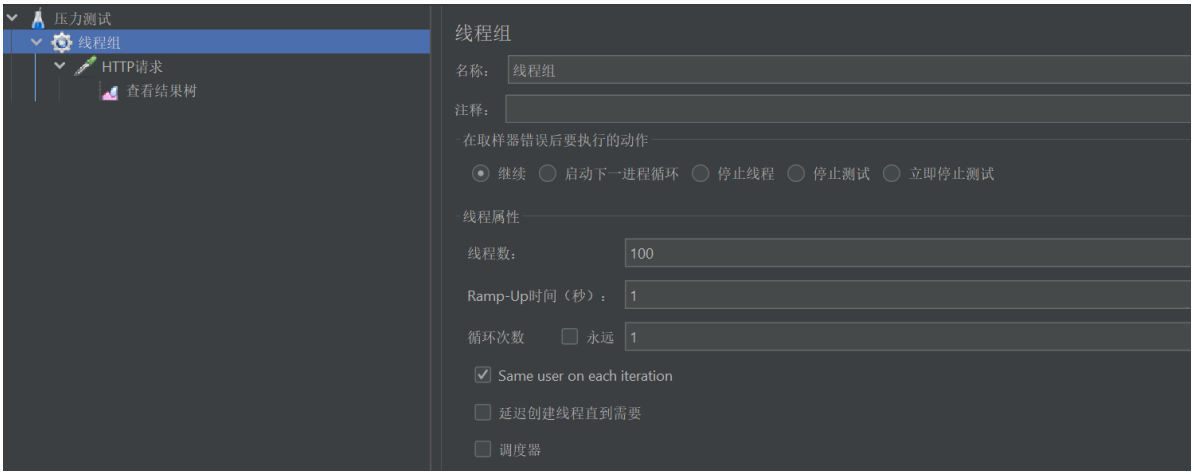


做了一个监控面板监控http访问次数（用到了prometheus提供的数据）

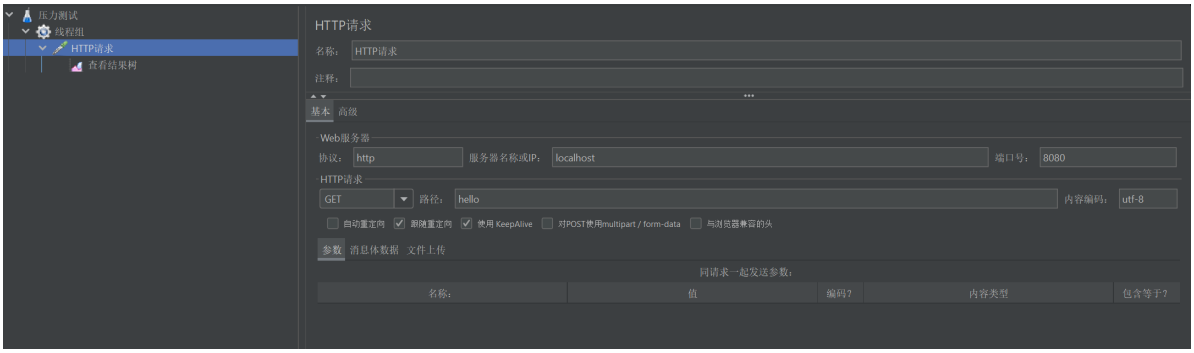
3.2 压测并观察监控数据（段霁峰）

下载Jmeter工具，用管理员权限打开，添加线程组，参数如下：

- 压力测试参数-线程组:

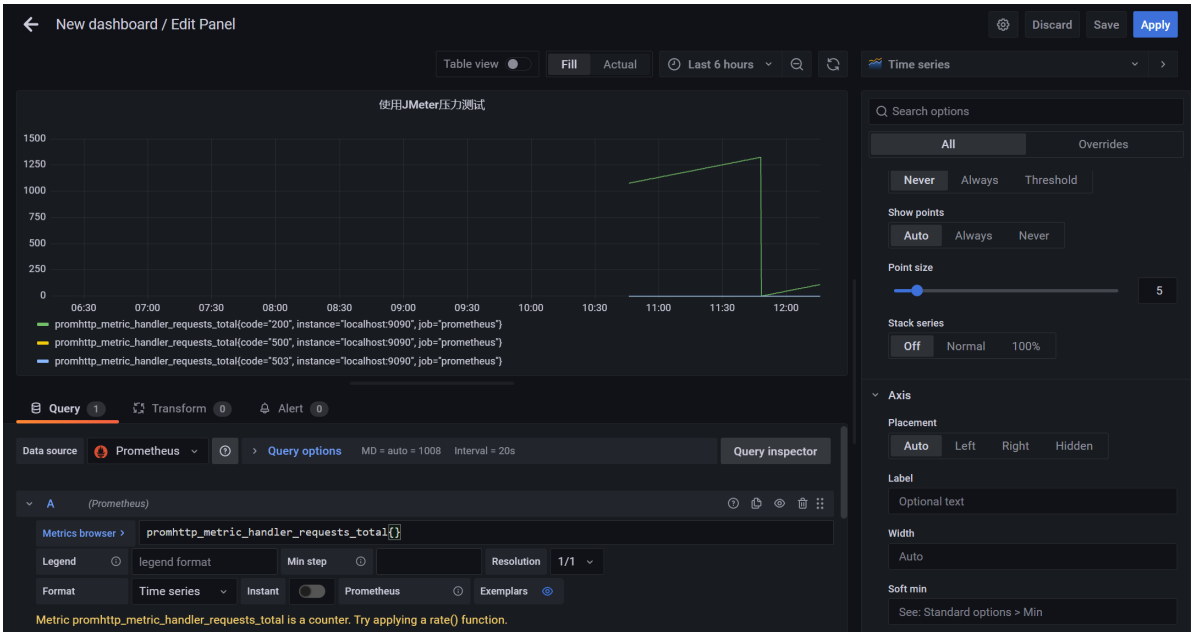


- 压力测试参数-http请求:



新建结果树，点击运行，可以看到访问都成功了：

访问`127.0.0.1:3000`，新建一个面板监控数据：



可以看到折线，是因为我一开始多点了几下，创建面板之后又点了一次

3.3 实现 Rolling Update CRD 以及 Controller（段霁峰）

3.3.1 定义 CRD 模型

3.3.2 定义 Controller

3.3.3 Watch Deployment 滚动升级产生的事件，通过 CRD 模型进行记录（RS 的变化，Pod 的变化）

3.3.4 提供 API，可以查询从滚动升级开始到结束，可以使用 informer 来 watch 资源的变化，并把变化信息打印出来。

3.4 Auto Scale（bonus功能）