# 基于云原生技术的软件开发 - 大作业

29组

211250011 王岩

211250026 陈凯权

211250053 江逸群

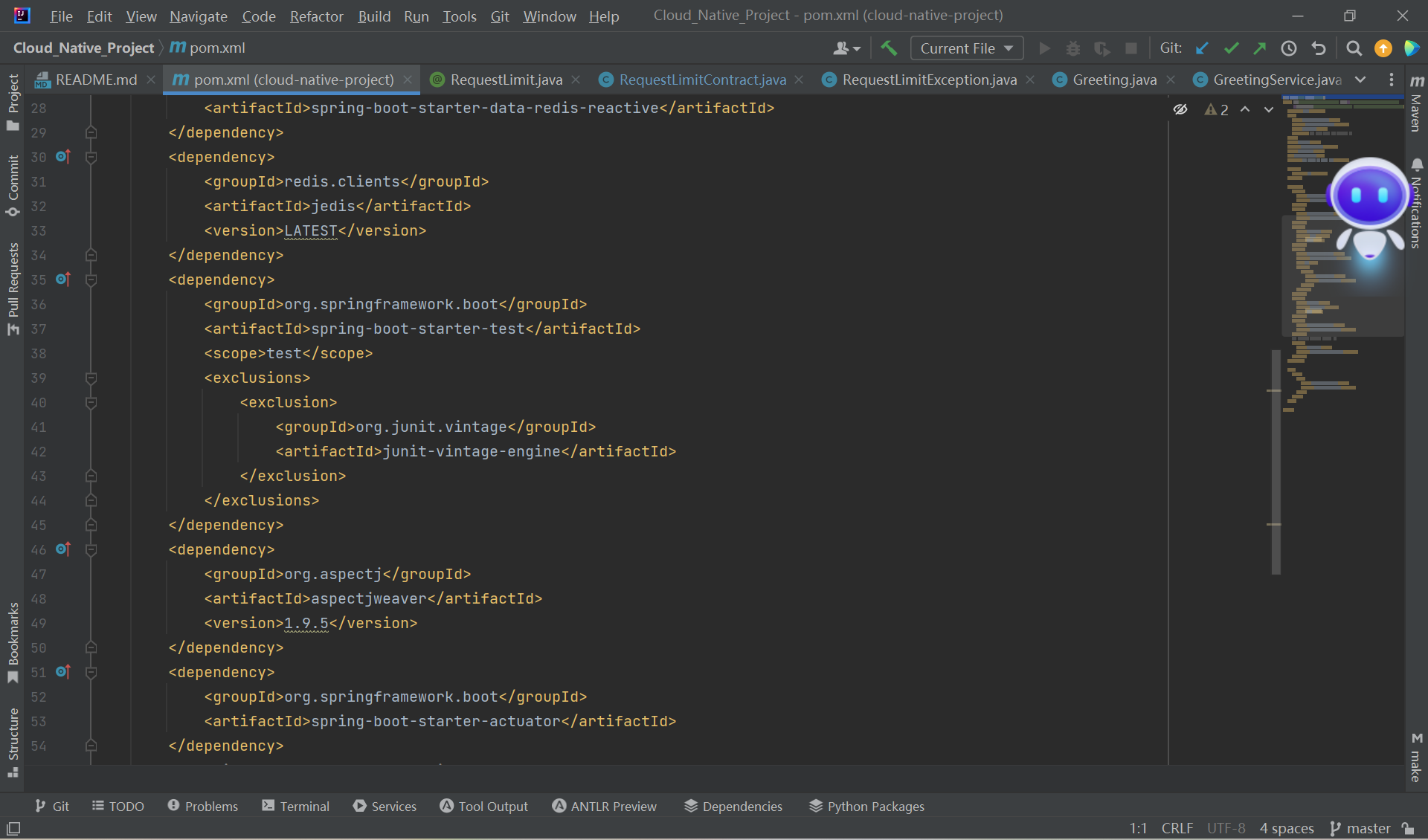
211250204 王怡

**Part1 功能要求**

1. **实现过程**

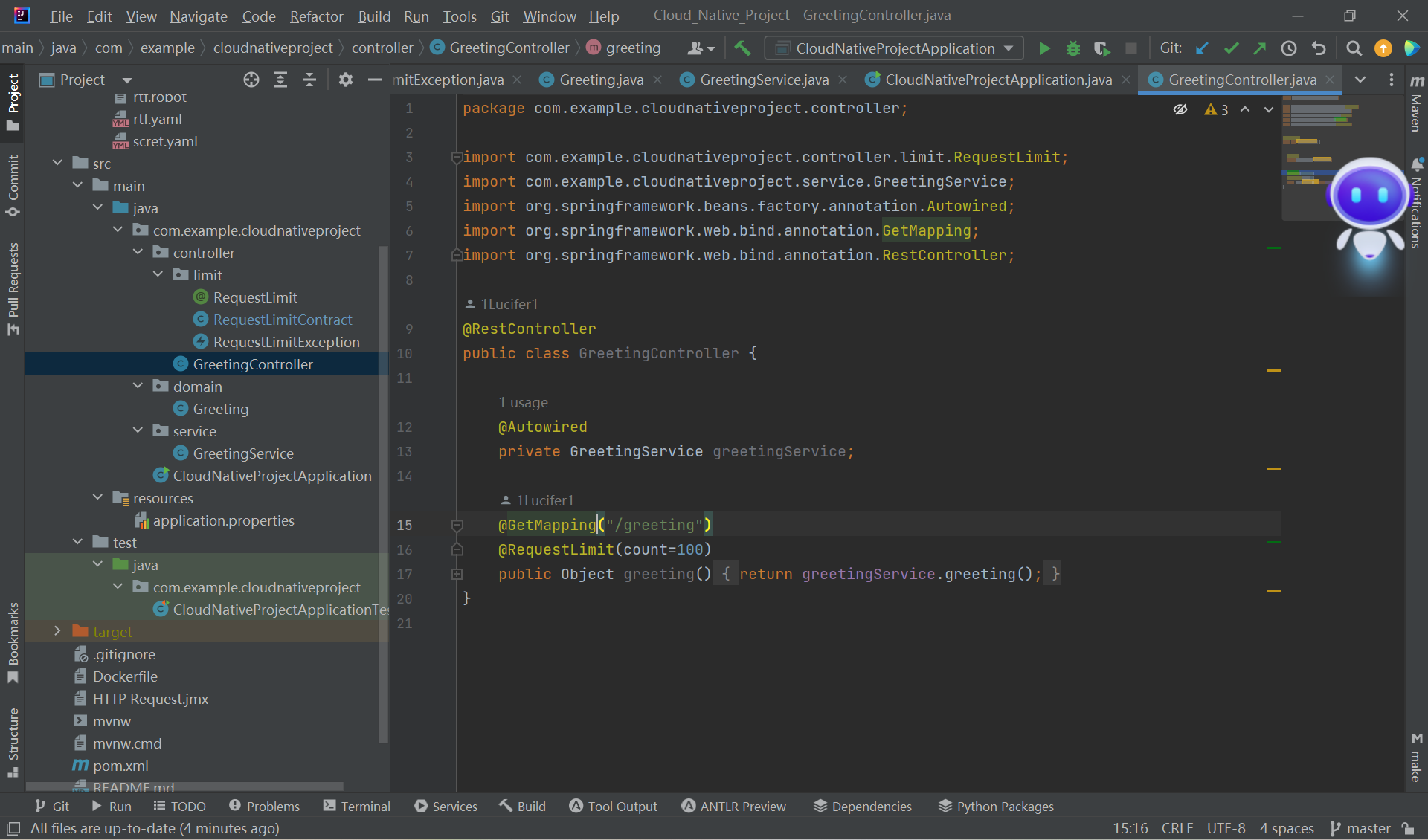
功能实现部分：基于spring boot实现rest接口和限流功能

首先添加Spring框架相关依赖，主要是添加Spring Web依赖到项目的pom.xml文件



创建一个controller类，定义get请求的接口

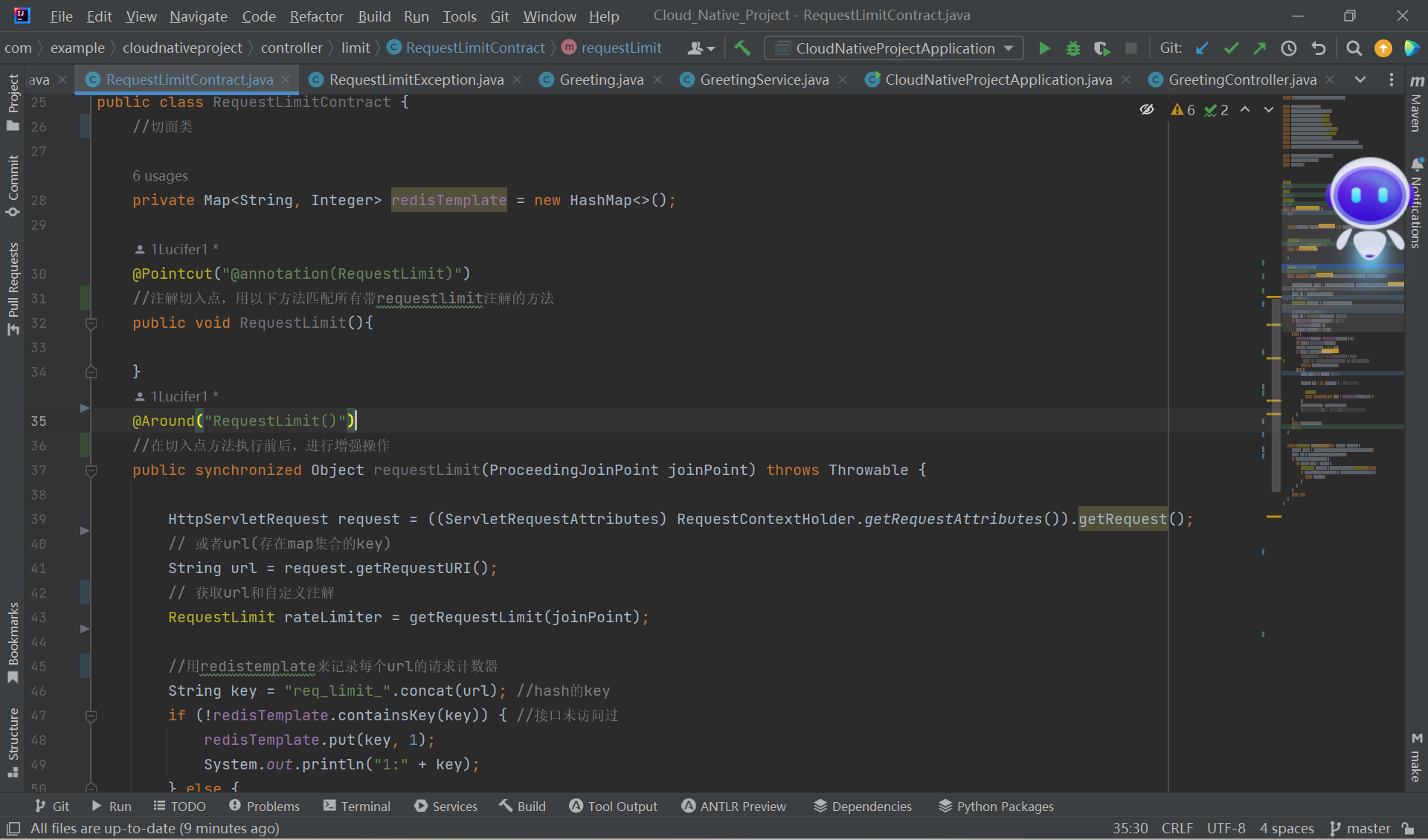
同时运用自定义注解@RequestLimit用于限制接口的请求频率

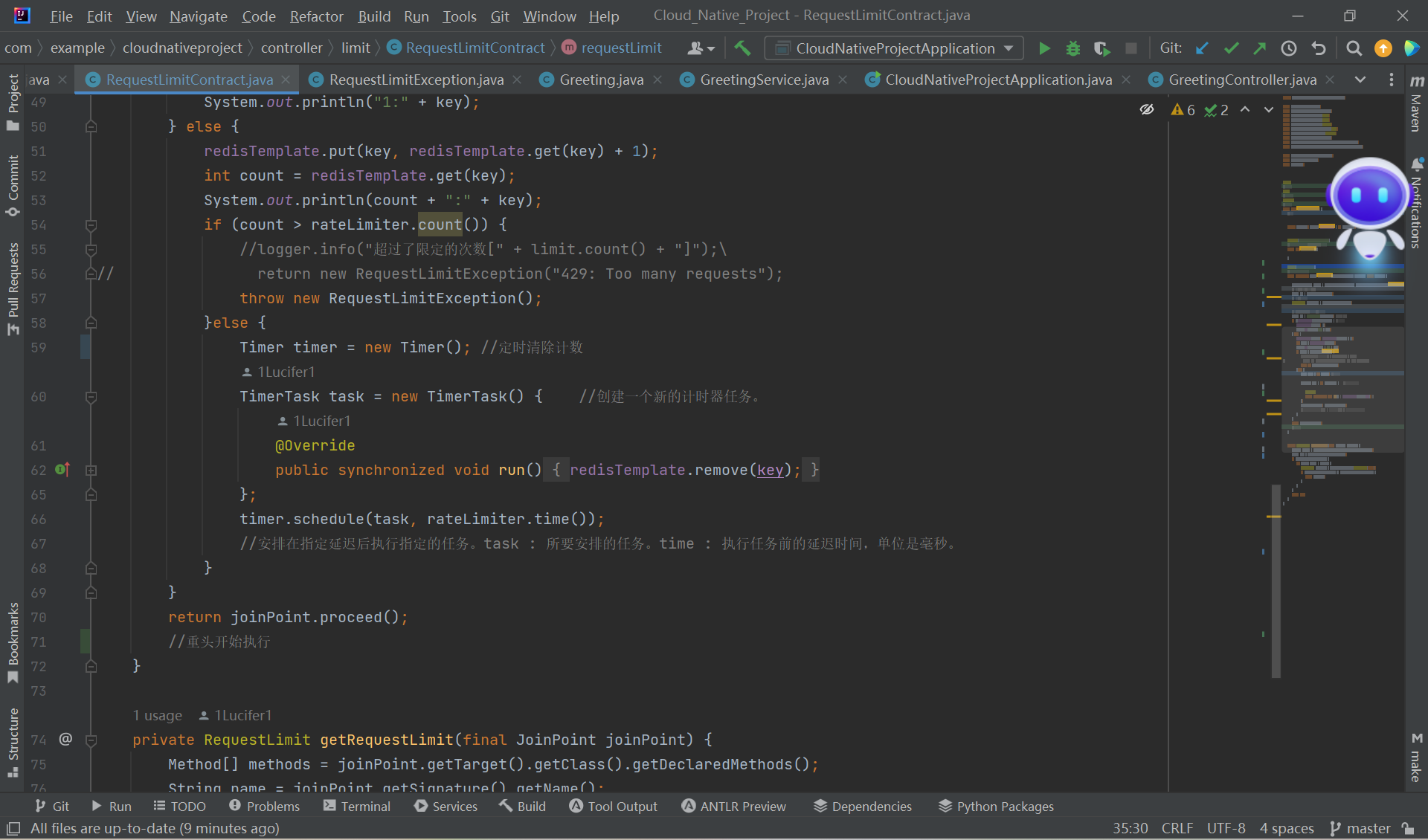


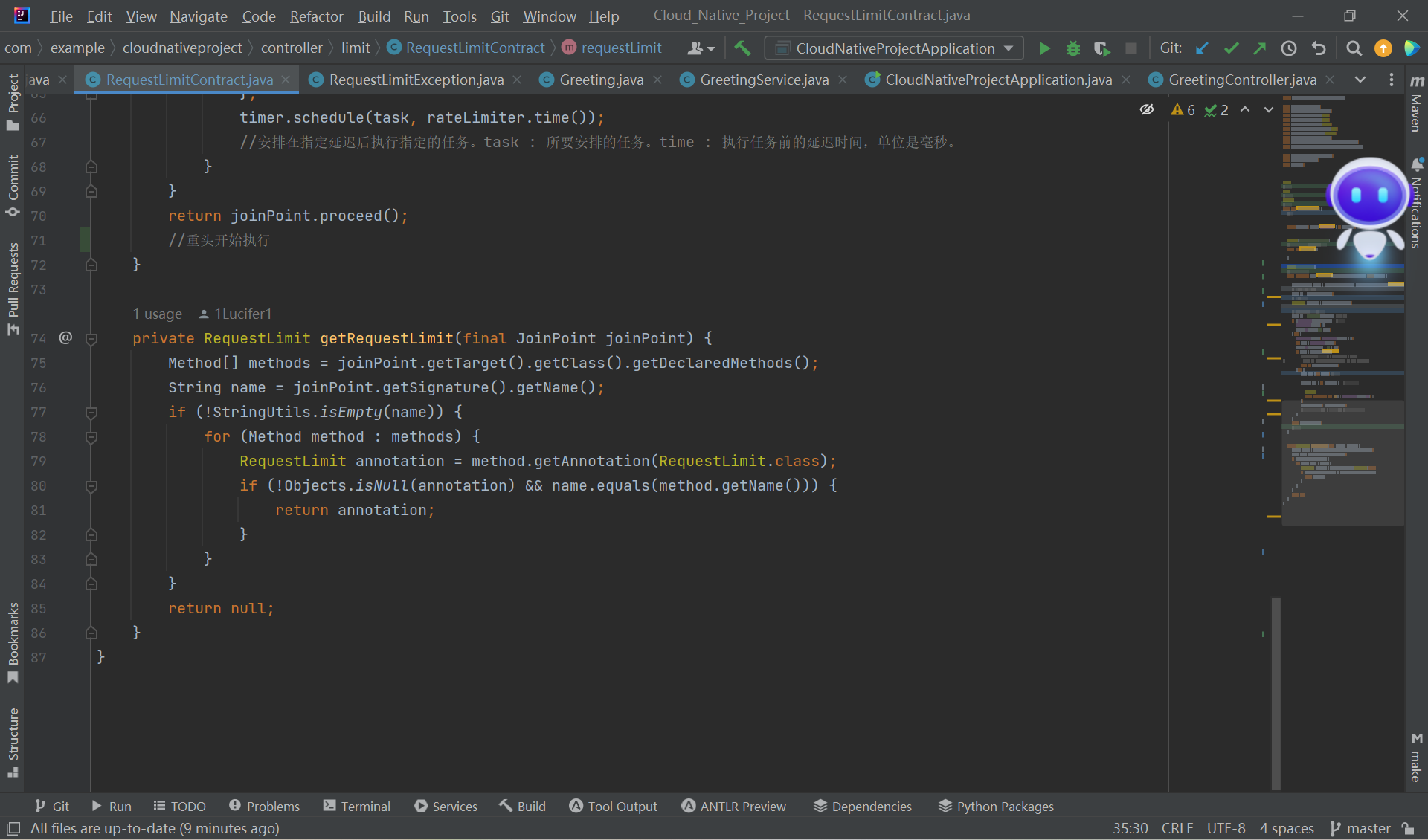
在上述代码中，@RequestLimit是一个自定义注解，用于标注需要进行限流的接口。该注解可以通过设置count属性来指定每秒允许的请求数量。

在使用该注解时，需要通过AOP（面向切面编程）拦截请求，并进行限流判断。可以创建一个切面类来实现这个功能。

下面展示切面类的设计







首先在切面类中定义一个切入点RequestLimit(),用于匹配带有@RequestLimit注解的方法

限流功能通过增强操作实现，在requestLimit()方法使用@Around注解，在目标方法执行前后进行增强操作。

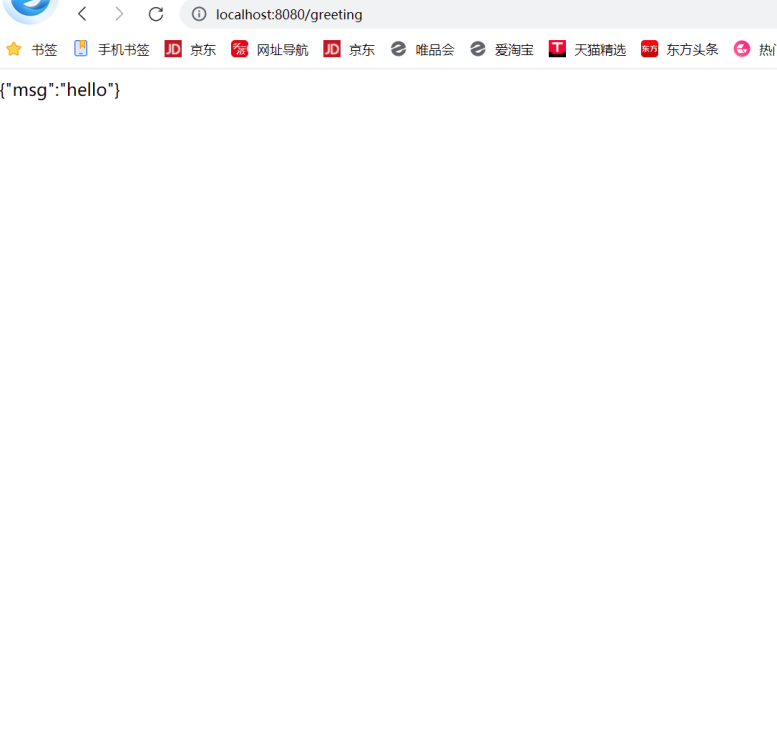
增强操作具体为，获取当前请求的URL; 使用URL作为键将计数器存储在redisTemplate中。

如果计数器不存在，表示URL第一次访问，将计数器初始化为1。否则将计数器增加1，并和注解中的限制（100）比较，如果超过限制，就抛出RequestLimitException异常。

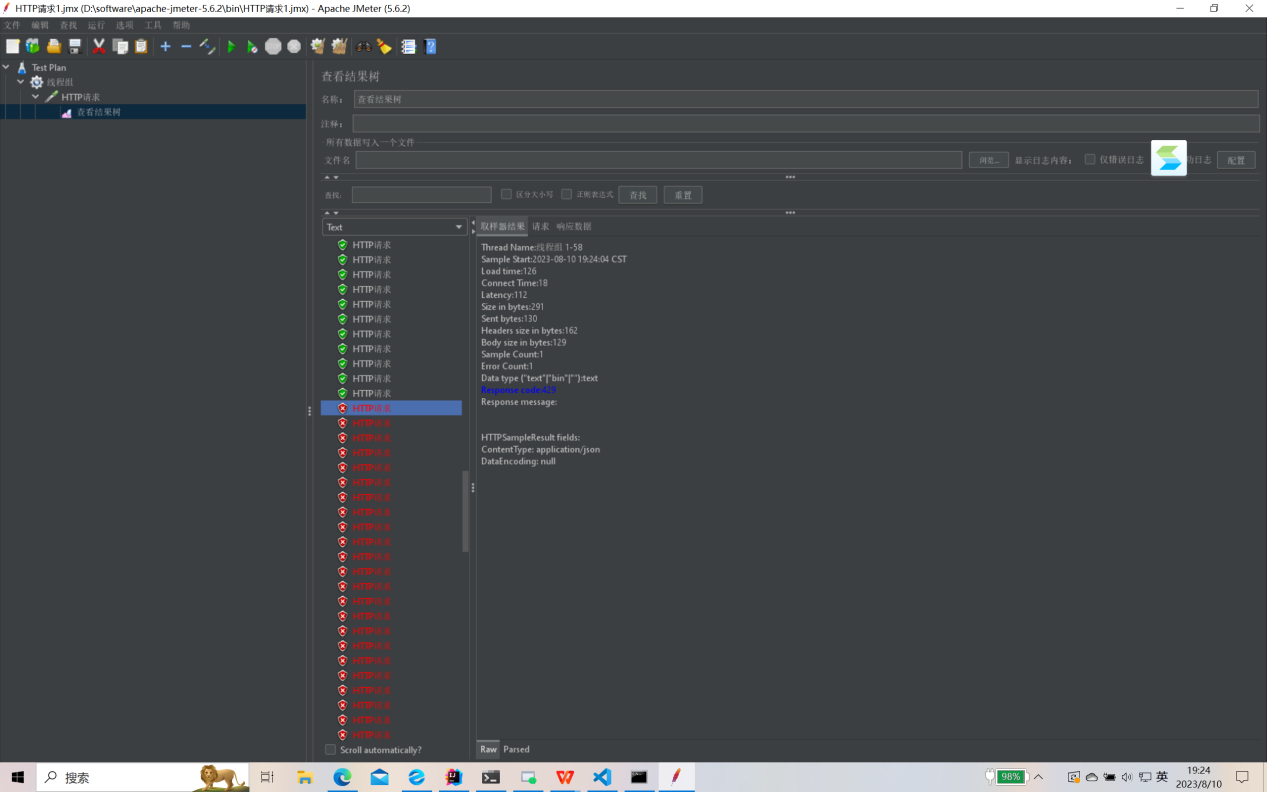
在计数器递增后，使用Timer定时器来调度一个任务，在一定时间后（注解中的时间），清除计数器。

注：红字部分是算法的核心思路。

1. **效果**
2. **REST接口**

****

1. **限流429**

****

接口提供限流功能，如上图所示，访问过多时返回429(Too many request)

1. **统一限流**

如前面限流方法所介绍的，采取redisTemplate实现分布式限流，以应对后端服务多个实例的情况。

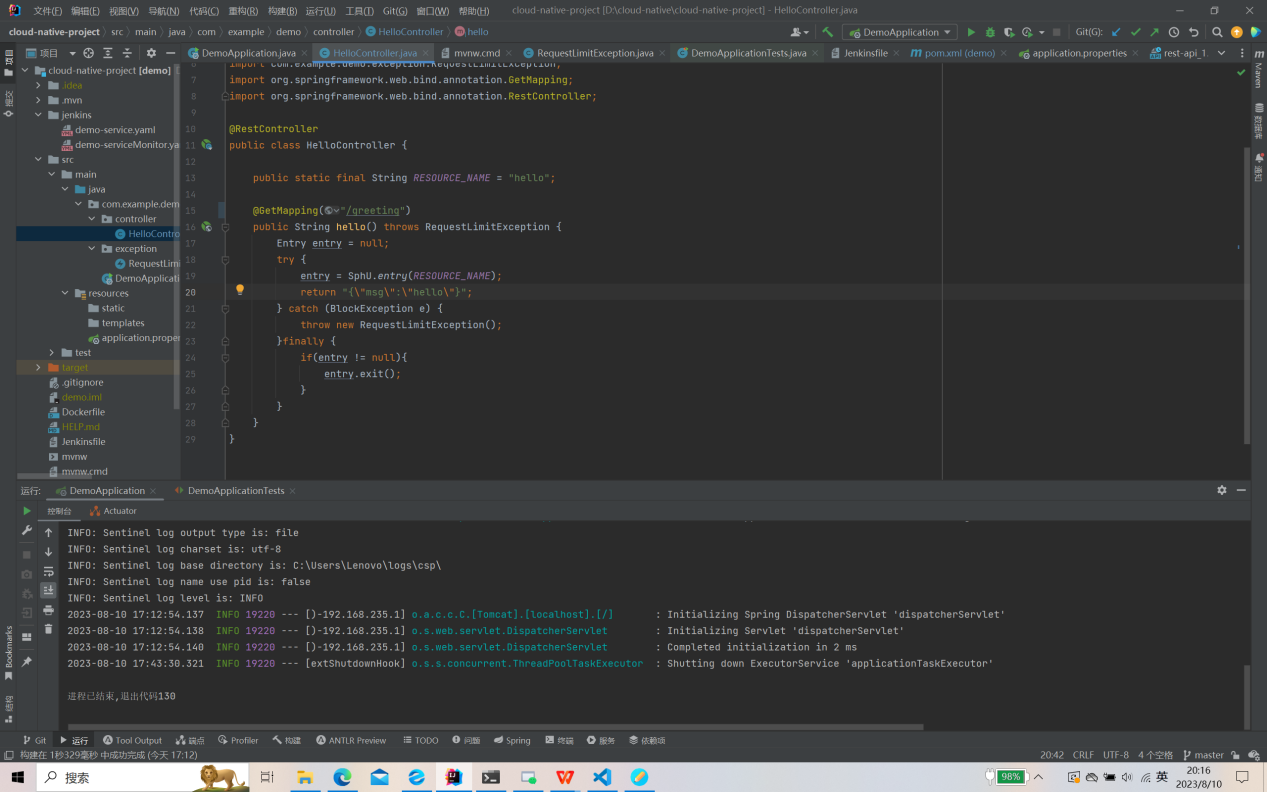
1. **补充**

我们提供了另外一种统一限流方式，采取Sentinel，Sentinel 是面向分布式、多语言异构化服务架构的流量治理组件，主要以流量为切入点，从流量路由、流量控制、流量整形、熔断降级、系统自适应过载保护、热点流量防护等多个维度来帮助开发者保障微服务的稳定性。

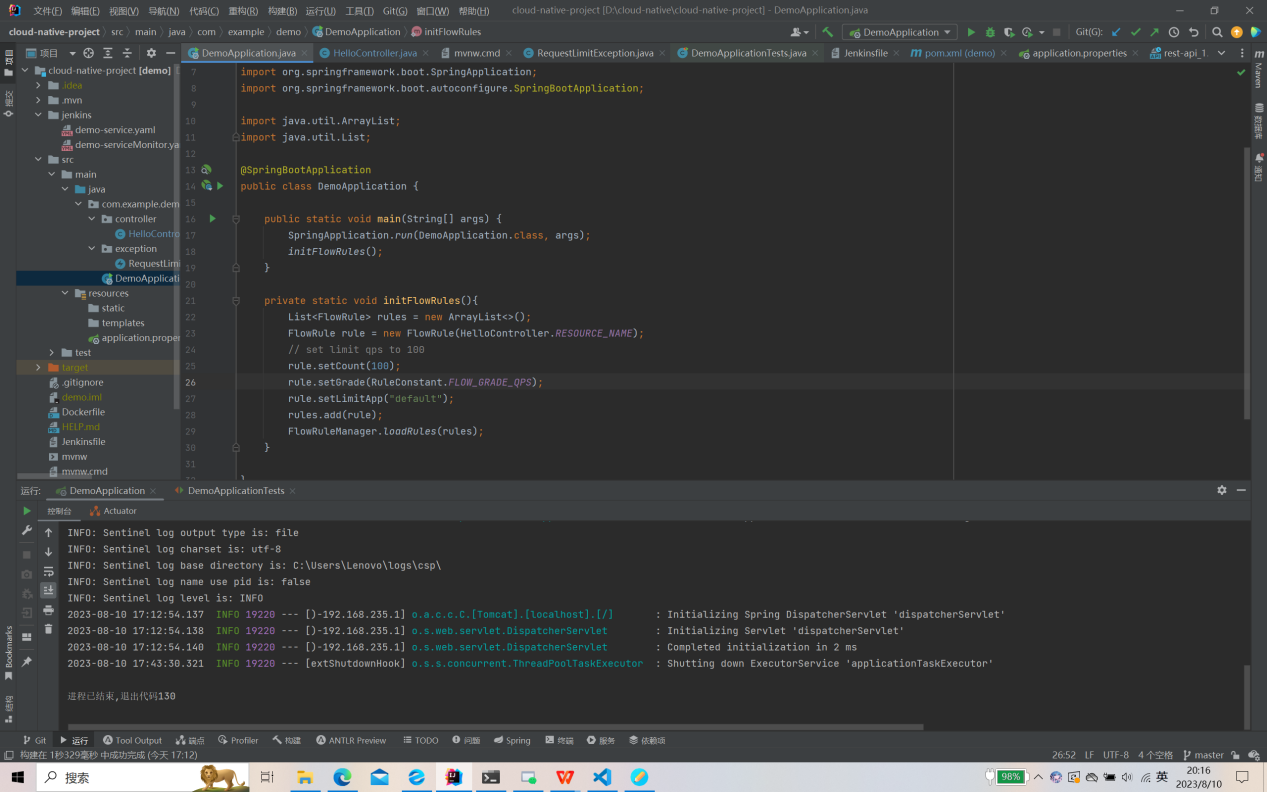
1. 导入相关依赖



1. 定义资源



1. 定义规则



限定每秒最多只能通过100个请求。

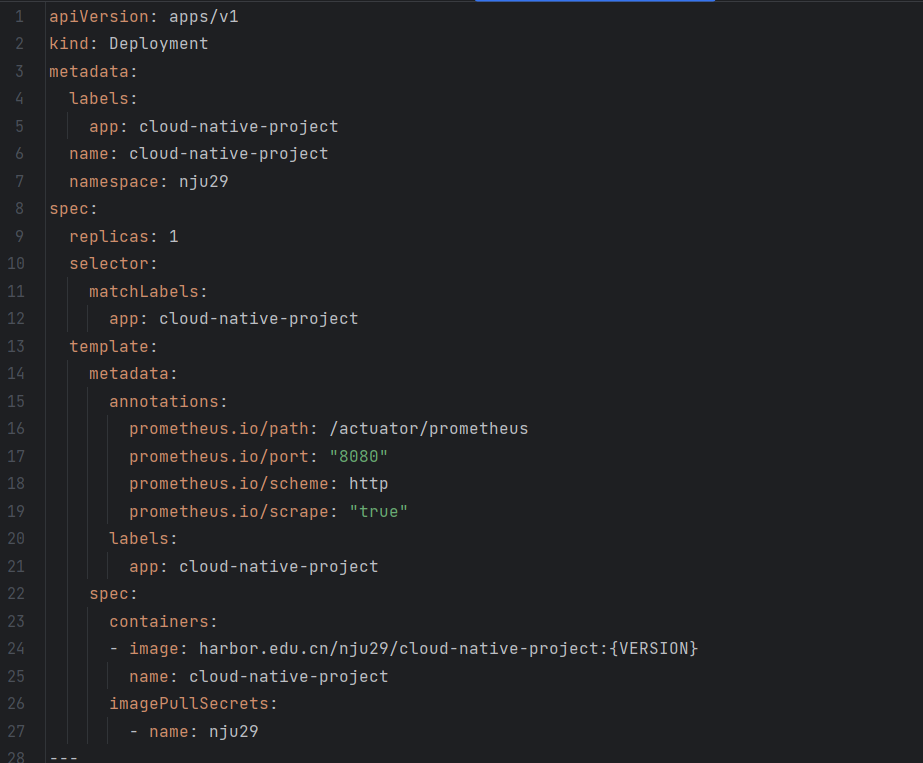
**Part2 DevOps要求**

1. **创建Dockerfile文件，用于构建镜像**

在根目录下创建Dockerfile文件，基于openjdk:8-jre-alpine构建所需镜像，内容如下



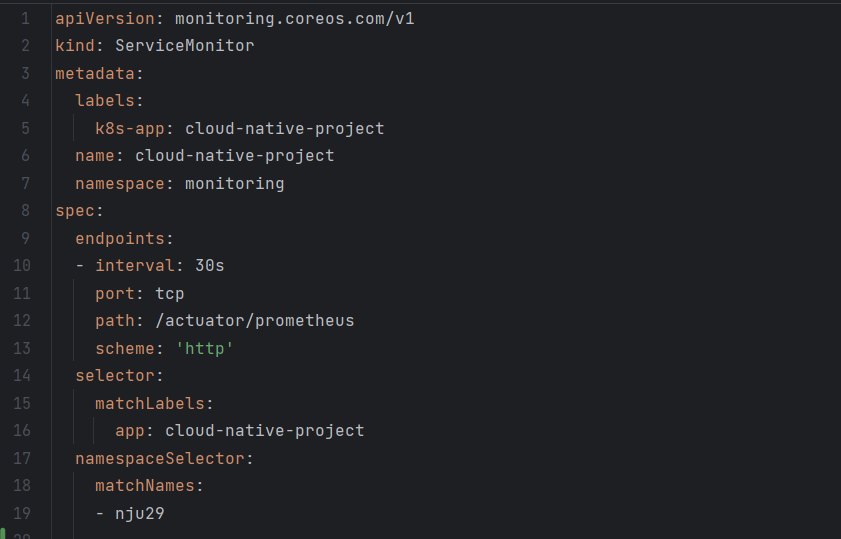
1. **为该项目准备Kubernetes编排文件，用于在Kubernetes集群上创建Deployment和Service**
   1. Deployment



* 1. Service



* 1. serviceMonitor



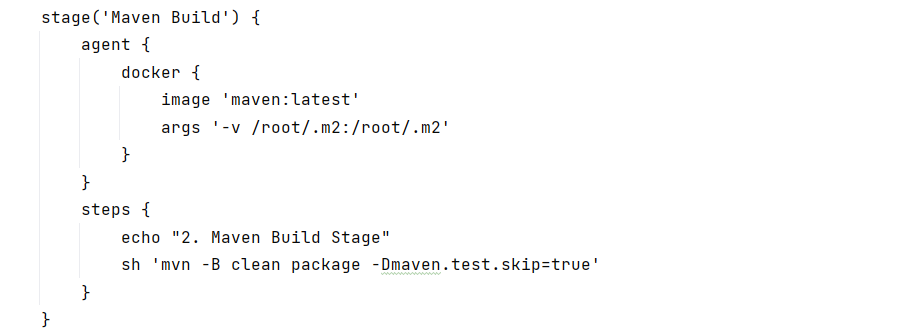
1. **编写Jenkins持续集成&持续部署&持续测试流水线**

**采用了持续集成流水线和持续部署流水线合二为一的流水线策略**

1. 克隆代码



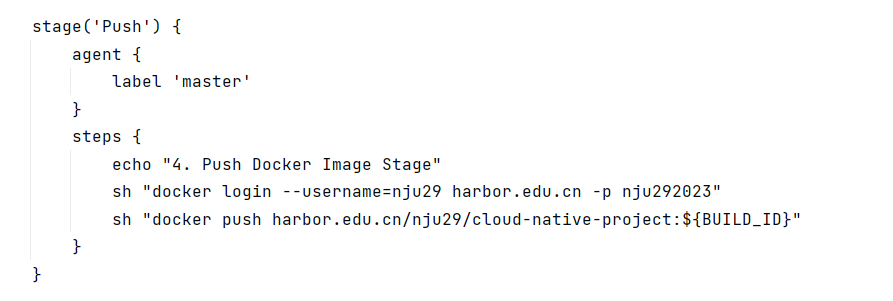
1. Maven打包



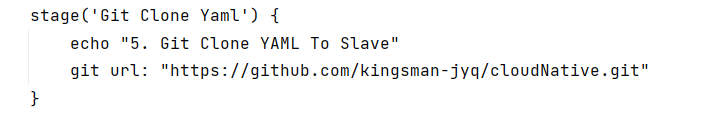
1. 构建镜像



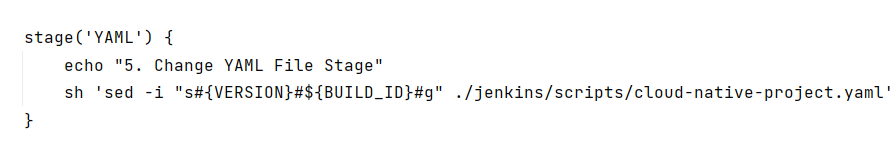
1. 推送镜像



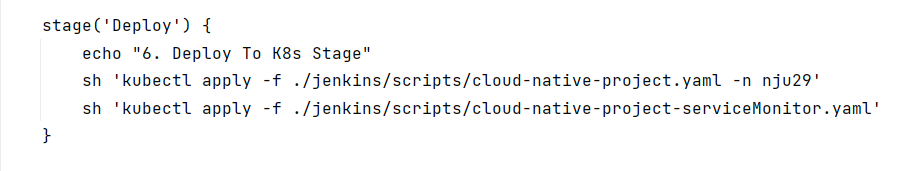
1. 拉取编排文件



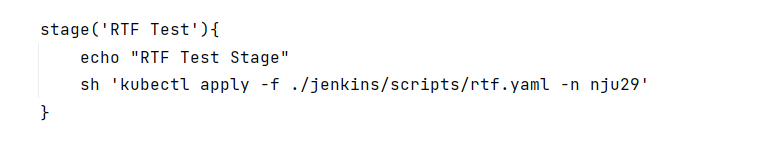
1. 替换YAML文件变量



1. 部署



8) RTF测试



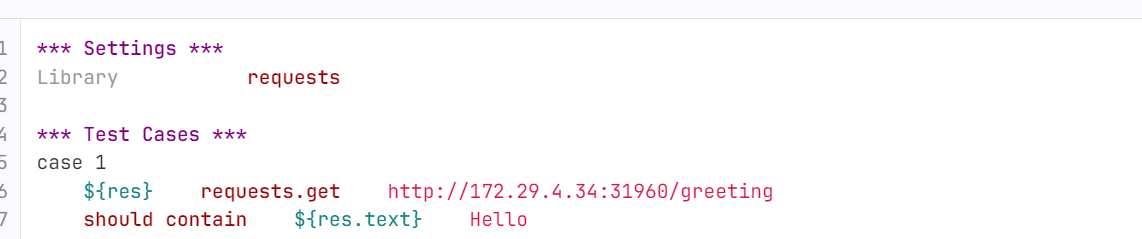
**其中rtf测试文件如下：**

**配置文件**





采用gitlab放置了一份脚本文件，便于直接拉取获得文件内容，同时将python镜像先前push到harbor仓库中，避免连接超时。test执行脚本如下：

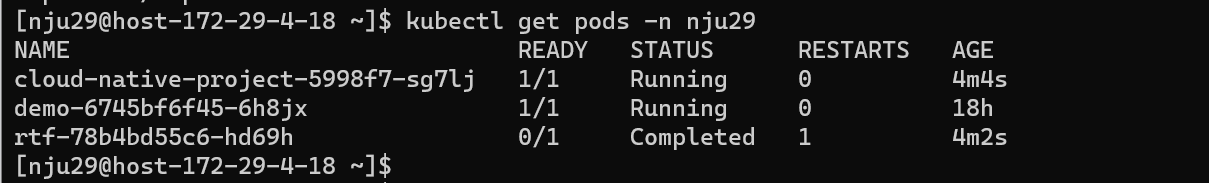


最终运行结果



部署结果

登录k8s节点进行查看，发现以下结果：



可以发现cloud-native-project处于正常Running状态，而测试也已完成。

使用kubectl logs查看测试pod日志，查看测试结果：

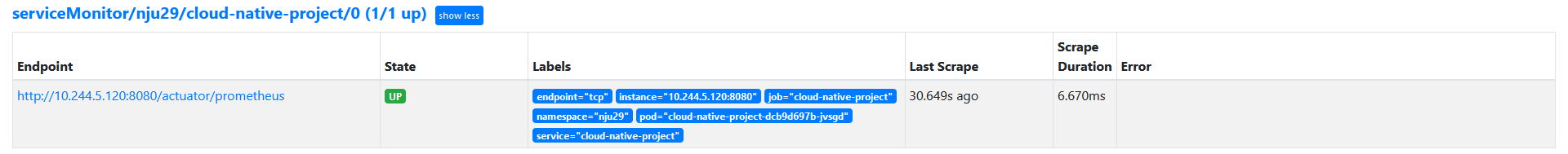


结果显示通过PASS。

由于Jenkins没有公网ip，无法实现代码提交到仓库自动触发流水线，不过或许部署到学校的gitlab平台可以？不过先前使用gitlab出现了一些问题，并未再加以尝试。

**Part3 扩容场景**

1. **Prometheus 采集监控指标**

****

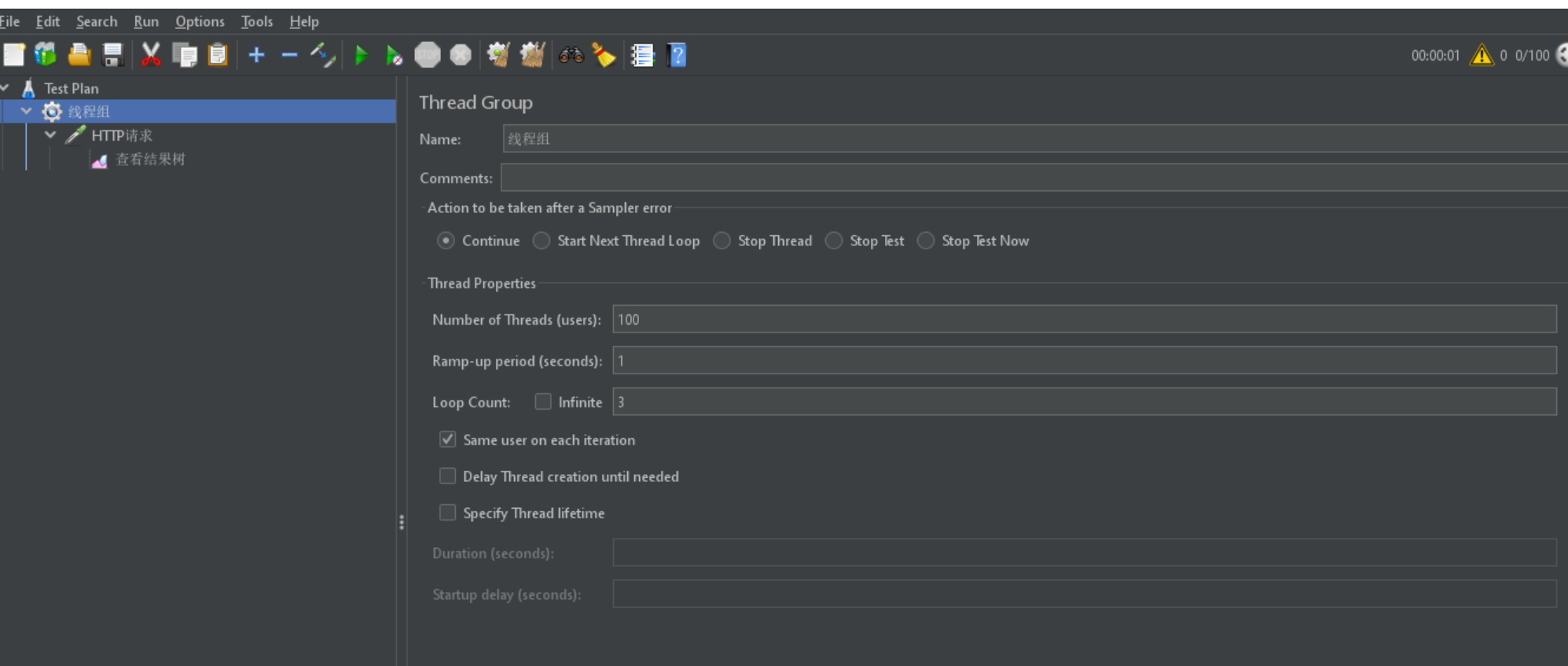
1. **Grafana 定制应用监控大屏**

****

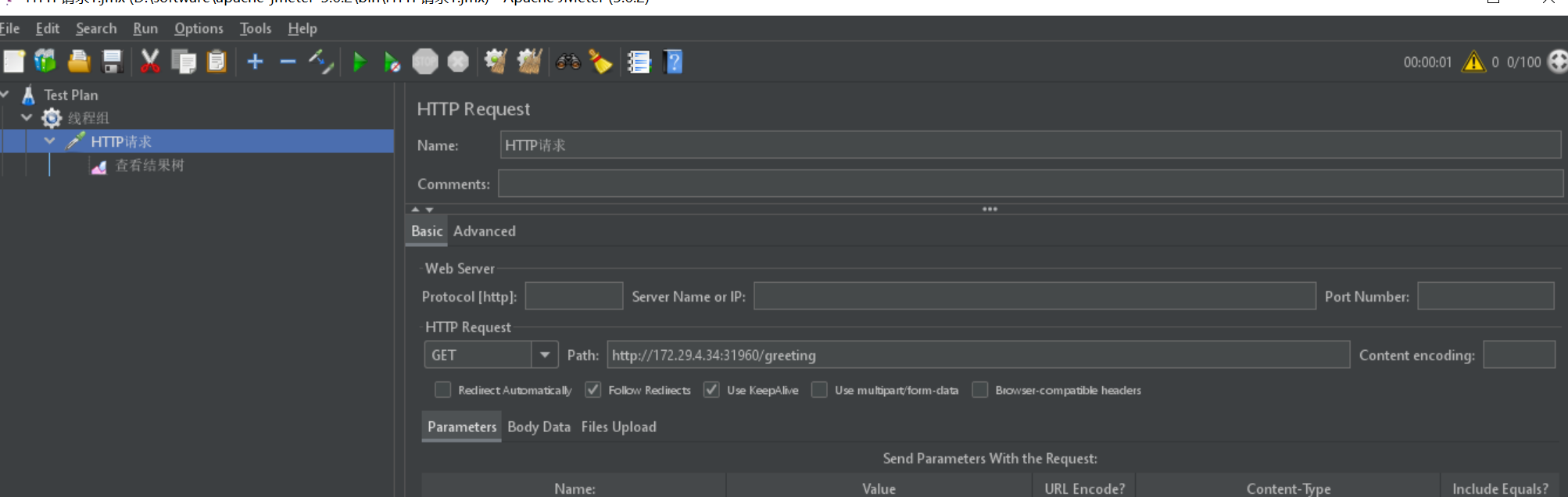
****

1. **压测并观察监控数据**
2. 压测工具JMeter配置如下：

线程组配置：

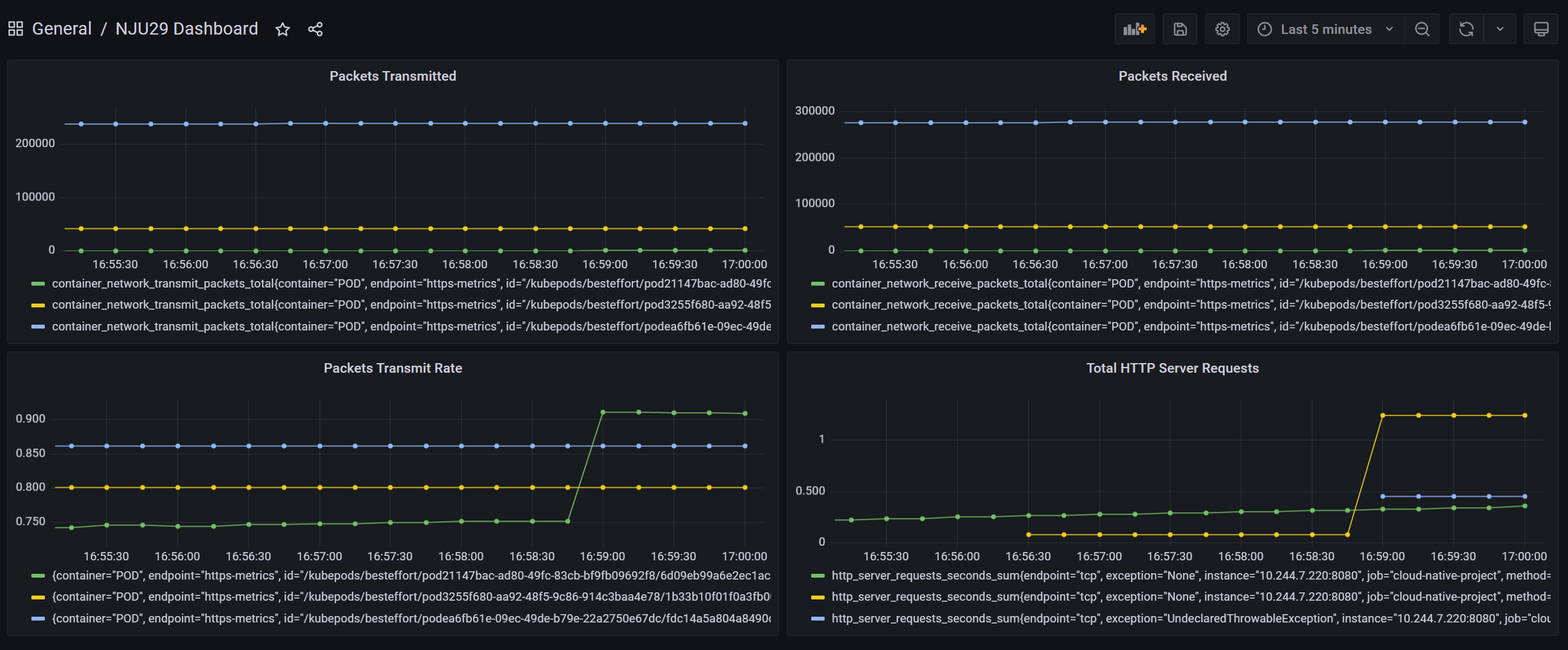


HTTP请求配置：



1. 压力测试

扩容前：

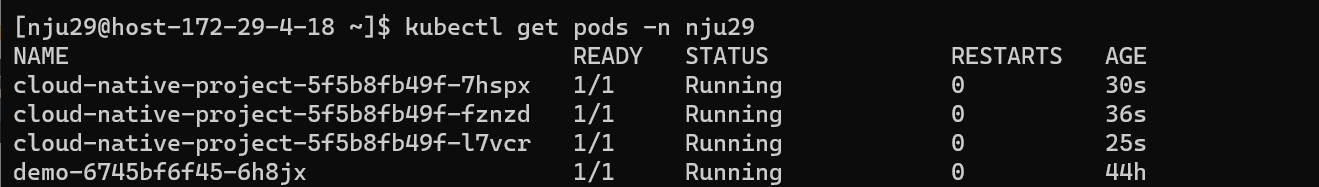


Total HTTP Server Request中429在纵坐标上体现为0.455，右下角蓝色线段。

手动进行扩容：将replicas修改为3，重新流水线部署

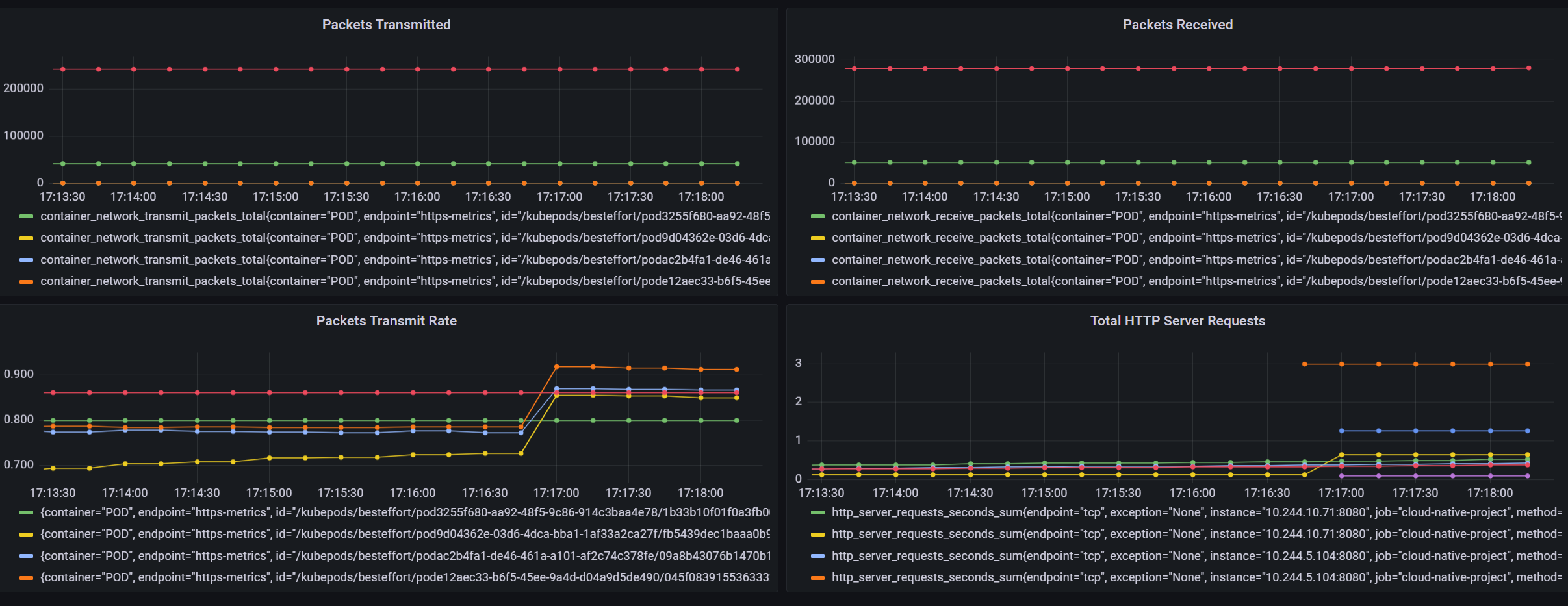


查看扩容后pod



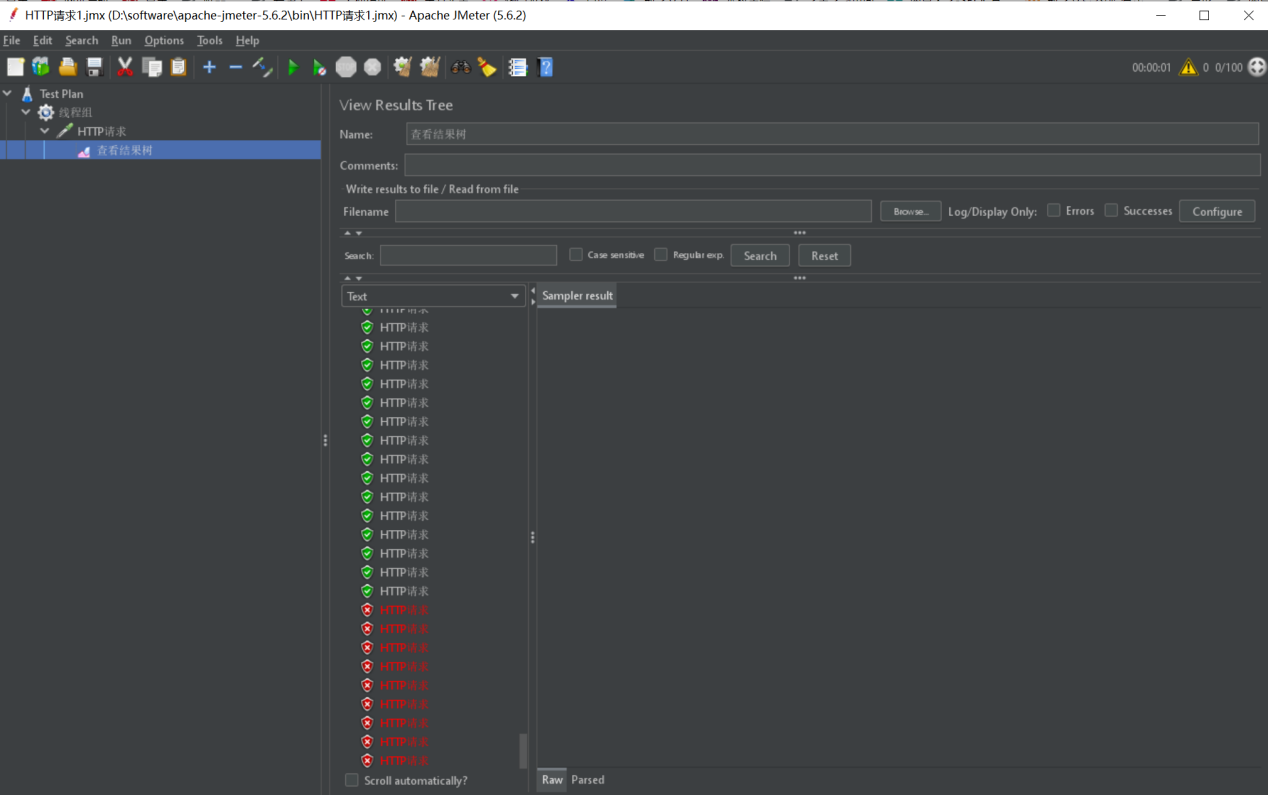
同样的JMeter配置进行压力测试：

监控结果如下：



Total HTTP Server Request中429在纵坐标上体现为0.0883，右下角最下方紫色线段。

JMeter结果数结果：

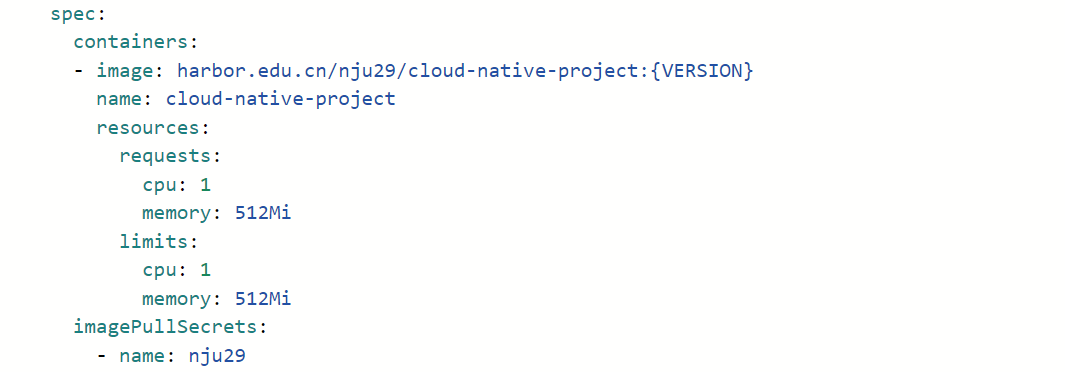


由原先接近一半429变为少数请求几个429，多次测量甚至429的数量会更少。

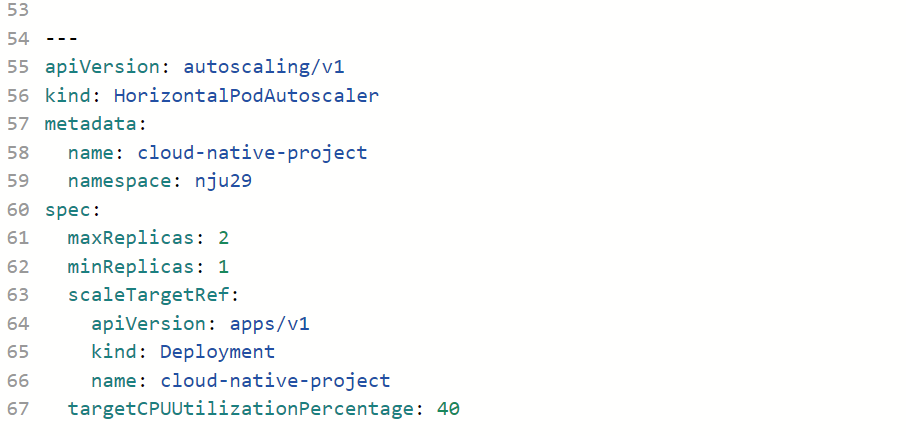
使用kubectl scale会报Error from server (Forbidden)，提示权限不足，所以直接修改yaml文件，重新部署。

1. **Auto Scale**

水平自动扩容，同上面一样，使用kubectl autoscale也被禁止，只能修改yaml文件，首先为deployment配置加上一些资源限制，如下图所示：



添加HorizontalPodAutoscaler配置，如下图所示：

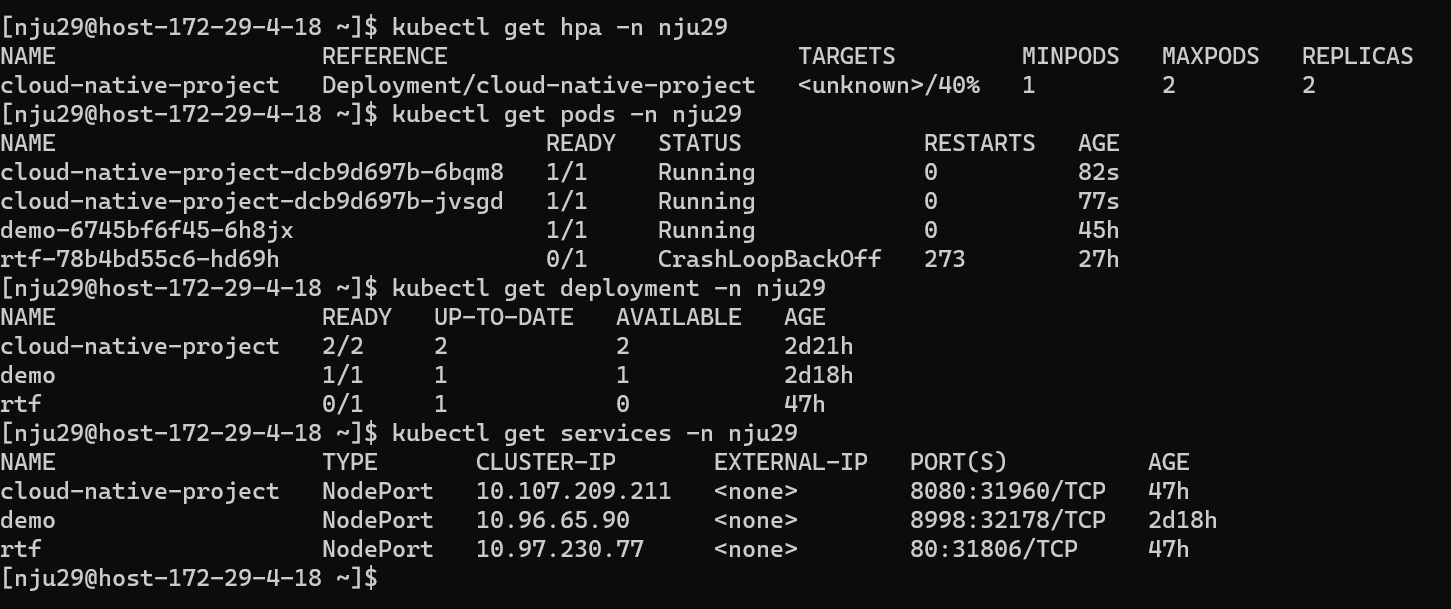


重新部署流水线

修改前信息为：pod数量为4



修改后：



由于为了看出效果，将最大Replicas设置为了2，现在可以看到重新部署流水线后cloud-native-project的pod为2，Auto Scale部分实现。