云原生大作业

组员：211250037 刘洋

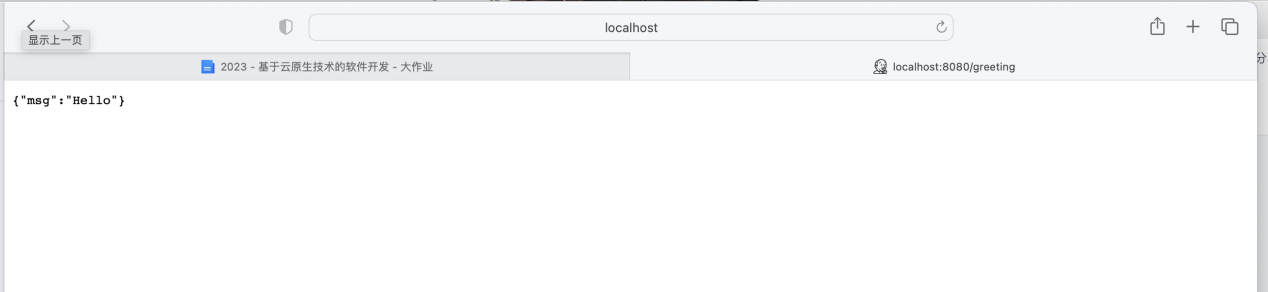
一．功能要求

1.实现一个 REST 接口（简单接口即可，比如 json 串 {"msg":"hello"}）





本地运行



2.  接口提供限流功能，当请求达到每秒 100 次的时候，返回 429（Too many requests）



用了Spring AOP来实现请求限流功能，并使用@annotation(RequestLimit)作为切入点表达式来匹配带有@RequestLimit注解的方法，然后在requestLimit方法中进行限流处理。

@Pointcut("@annotation(RequestLimit)")：定义一个切入点，用于匹配所有带有@RequestLimit注解的方法。

@Around("RequestLimit()")：定义一个环绕通知，该通知会在匹配到的方法执行前后触发。在环绕通知中，我们对请求进行限流处理。

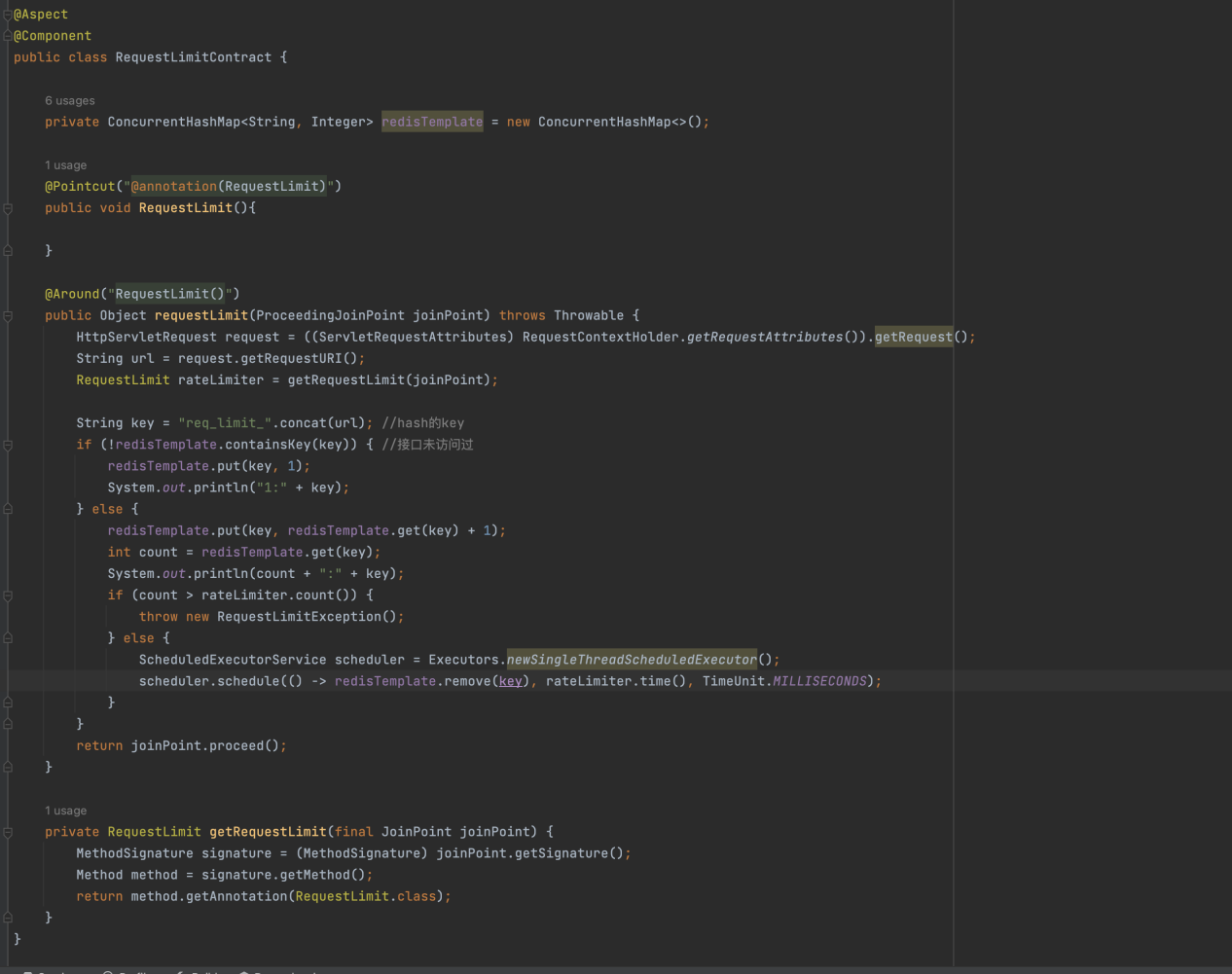
在requestLimit方法中，首先获取请求的URL和方法上的@RequestLimit注解，然后根据URL构造用于在redisTemplate中存储计数的键key。

接着，检查redisTemplate中是否存在该键。如果不存在，说明该接口尚未访问过，则将计数设置为1，并打印日志。

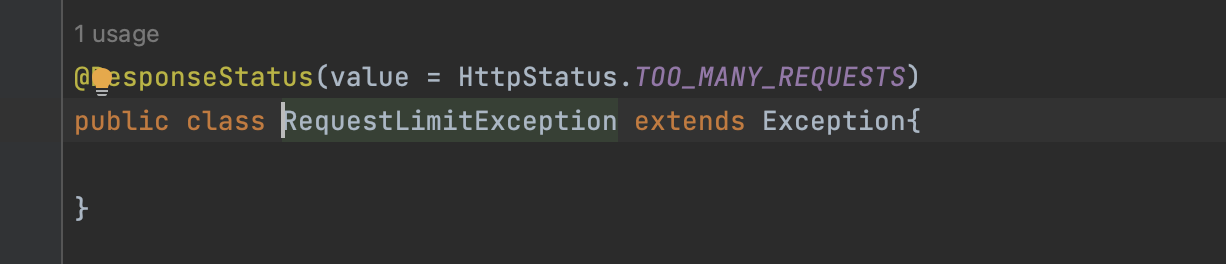
如果计数键已经存在，则将计数加1，并判断是否超过了限定的请求次数（rateLimiter.count()）。如果超过了限定次数，则抛出RequestLimitException异常。

否则，我们使用ScheduledExecutorService来创建一个定时任务，在指定的时间后移除redisTemplate中的计数键，从而实现计数的定时清除。这样可以避免存储一直增加导致内存占用过高。

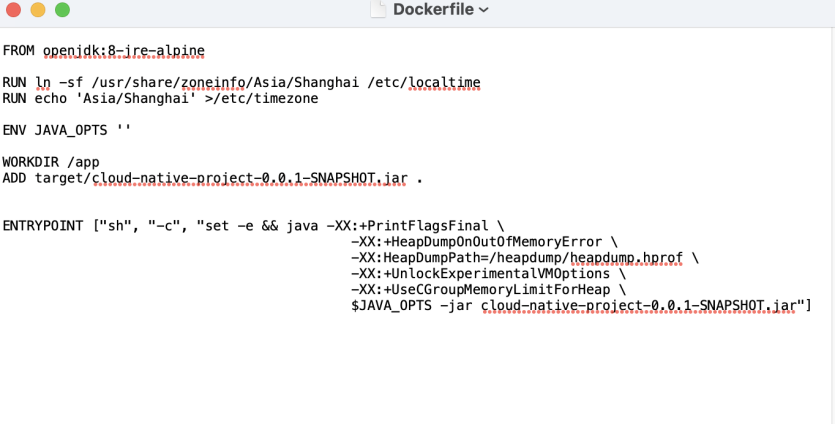
最后，我们调用joinPoint.proceed()方法来继续执行原始方法，确保请求继续正常处理。



当 value 大于 100 时返回前端的异常



1. DevOps 要求
2. 为该项目准备 Dockerfile，用于构建镜像



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

- 原始端口：8080

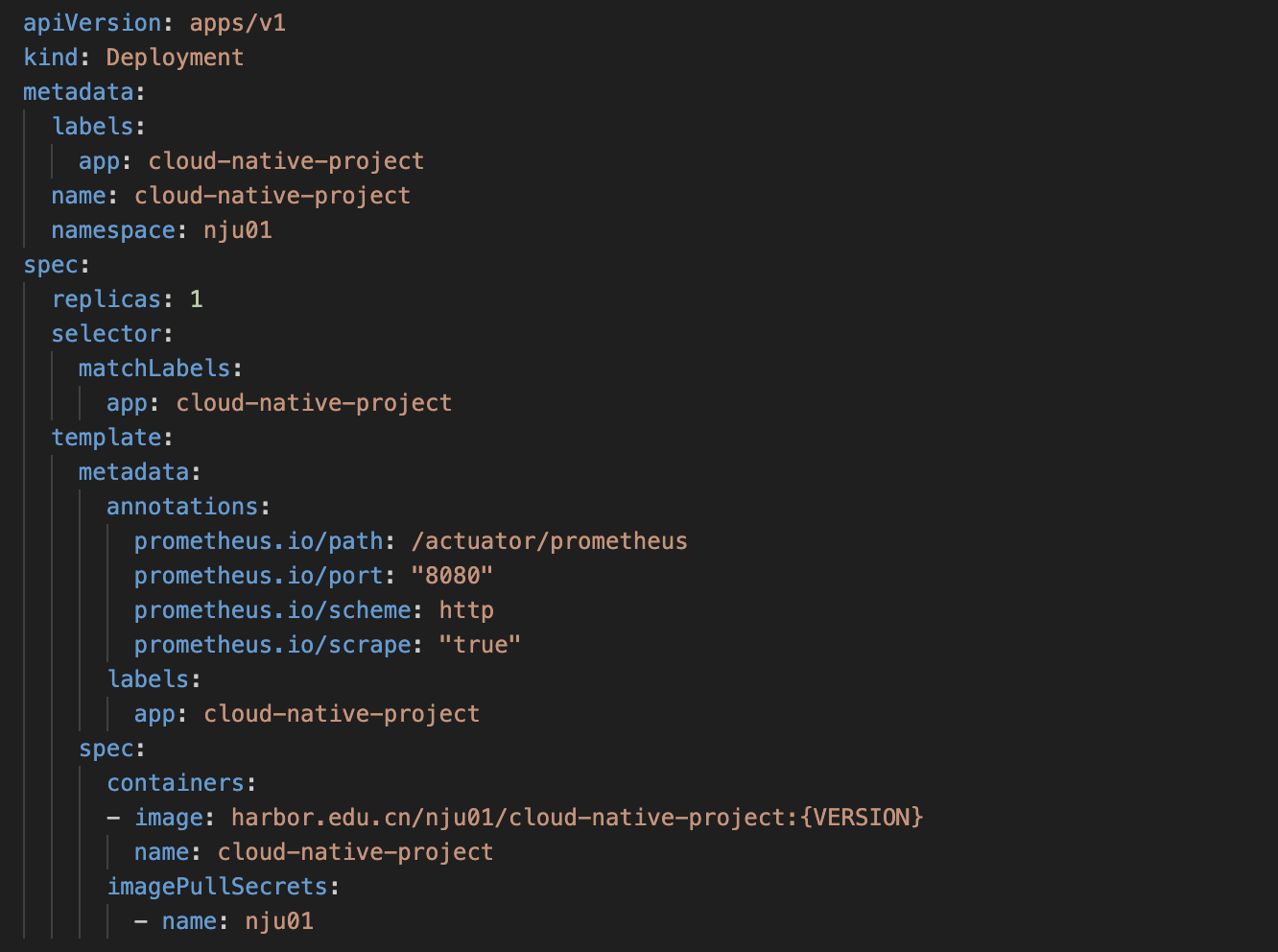
- 分配端口：30000

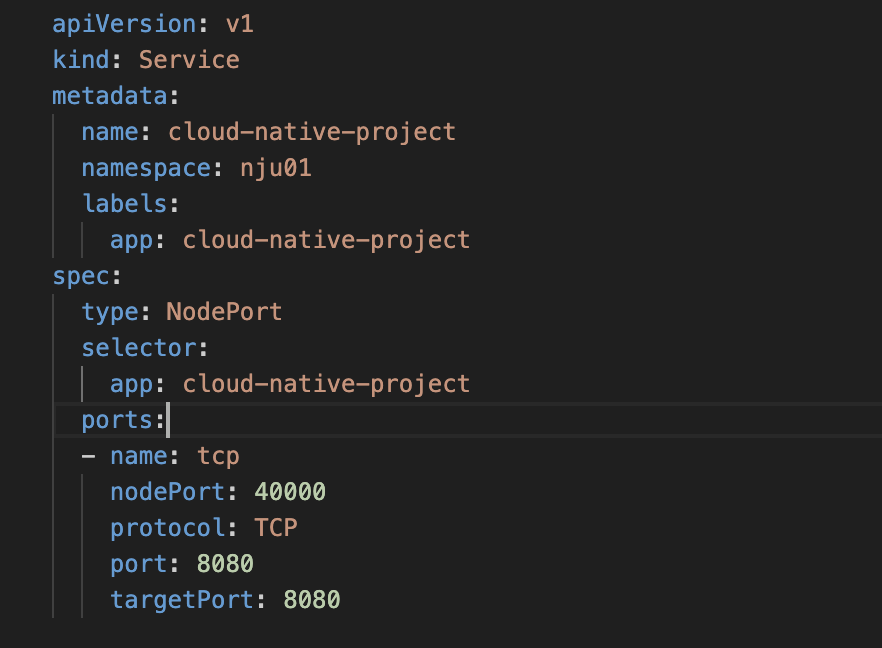
- 项目：cloud-native-project

- 类型：NodePort

- 镜像：harbor.edu.cn/nju01/cloud-native-project:{VERSION}

- Namespace：nju01





1. 编写 Jenkins 持续集成流水线，实现**代码构建**/**单元测试**/**镜像构建**功能（需要写至少一个单元测试）

pipeline {

agent none

stages {

stage('Clone to master') {

agent {

label 'master'

}

steps {

echo "1. Git Clone Stage"

git url: "https://github.com/211250037/Cloud\_Native\_Project.git"

}

}

stage('Maven Build') {

agent {

docker {

image 'maven:latest'

args '-v /root/.m2:/root/.m2'

}

}

steps {

echo "2. Maven Build Stage"

sh 'mvn -B clean 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-project-0.0.1-SNAPSHOT.jar -t cloud-native-project:${BUILD\_ID} . '

sh 'docker tag cloud-native-project:${BUILD\_ID} harbor.edu.cn/nju01/cloud-native-project:${BUILD\_ID}'

}

}

stage('Push') {

agent {

label 'master'

}

steps {

echo "4. Push Docker Image Stage"

sh "docker login --username=nju01 harbor.edu.cn -p nju012023"

sh "docker push harbor.edu.cn/nju01/cloud-native-project:${BUILD\_ID}"

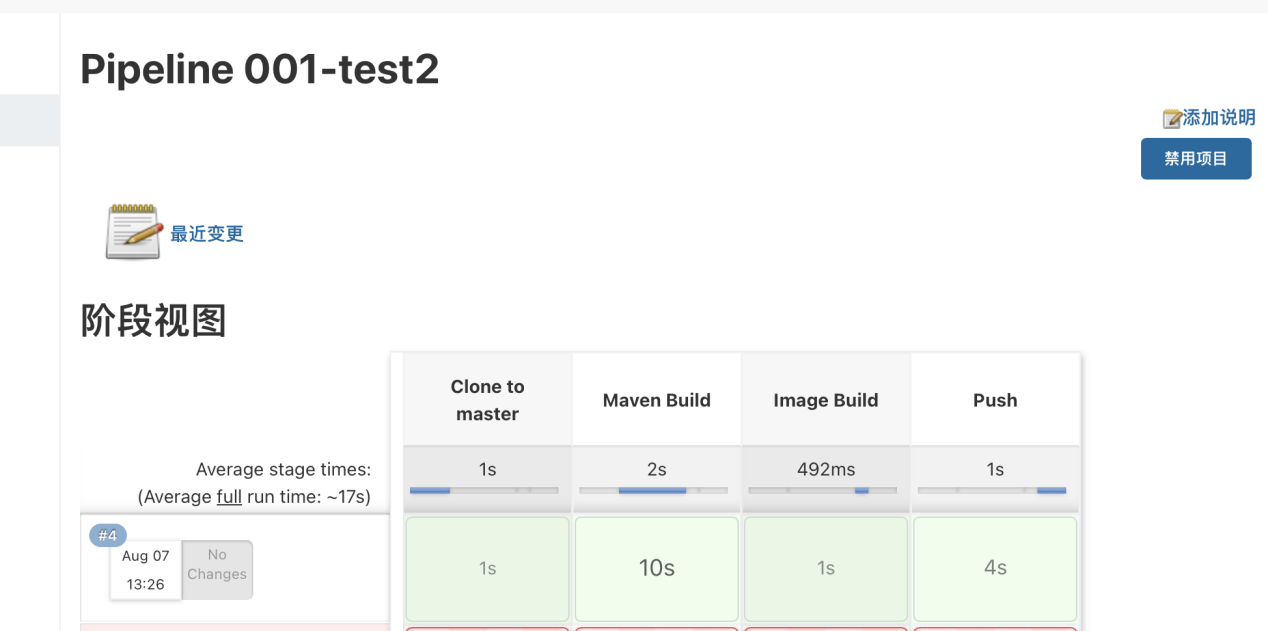
}

}

}

}

运行结果



1. 编写 Jenkins 持续部署流水线，实现部署到 Kubernetes 集群的功能，该流水线的触发条件为持续集成流水线执行成功

node('slave') {

container('jnlp-kubectl') {

stage('connect'){

sh 'curl "http://p.nju.edu.cn/portal\_io/login" --data "username=211250037&password=ly151503"'

}

stage('Git Clone') {

git url: "https://github.com/211250037/Cloud\_Native\_Project.git"

}

stage('YAML') {

echo "5. Change YAML File Stage"

sh 'sed -i "s#{VERSION}#${BUILD\_ID}#g" ./jenkins/scripts/cloud-native-project.yaml'

}

stage('Deploy') {

echo "6. Deploy To K8s Stage"

sh 'kubectl apply -f ./jenkins/scripts/cloud-native-project.yaml -n nju01'

sh 'kubectl apply -f ./jenkins/scripts/cloud-native-project-serviceMonitor.yaml'

}

stage('RTF Test'){

echo "RTF Test Stage"

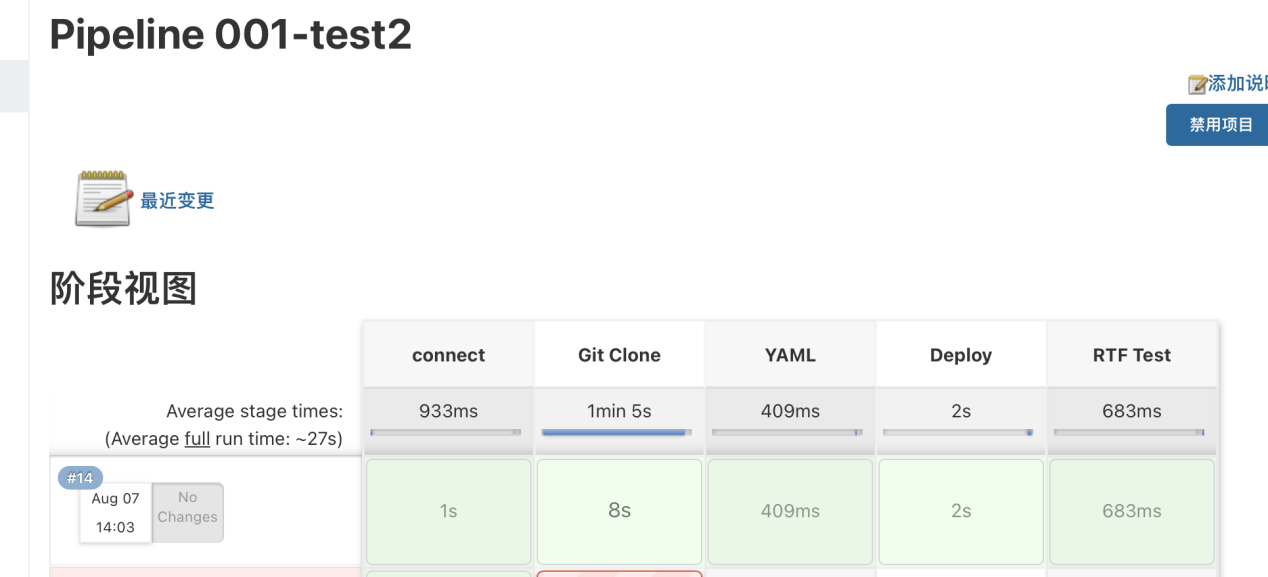
sh 'kubectl apply -f ./jenkins/scripts/rtf.yaml -n nju01'

}

}

}

运行结果



持续测试流水线

- 代码

stage('RTF Test'){

echo "RTF Test Stage"

sh 'kubectl apply -f ./jenkins/scripts/rtf.yaml -n nju01'

} - 脚本

\*\*\* Settings \*\*\*

Library requests

\*\*\* Test Cases \*\*\*

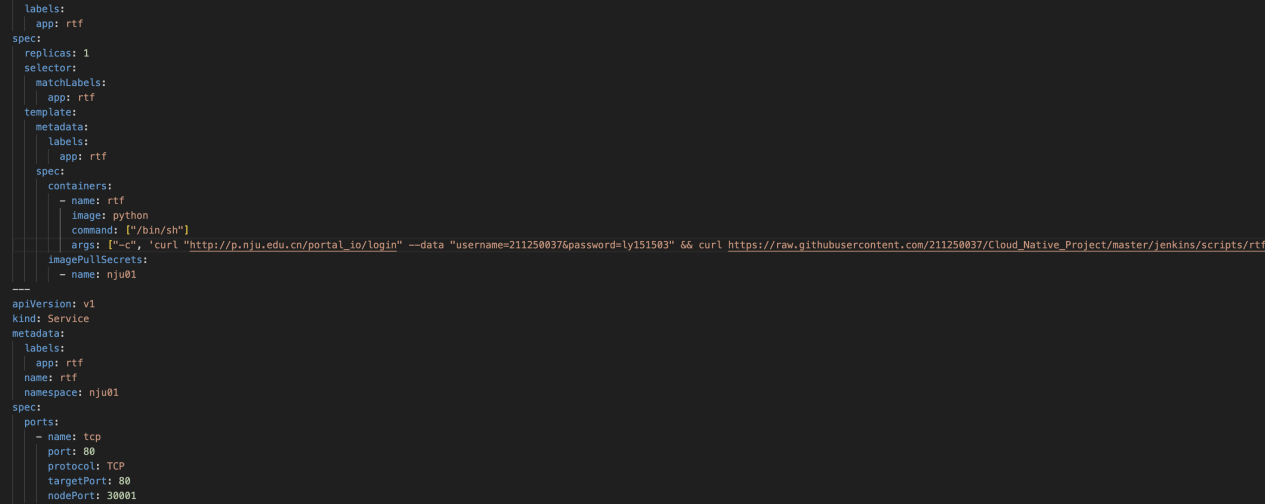
case 1

${res} requests.get http://172.29.4.47:30000/greeting

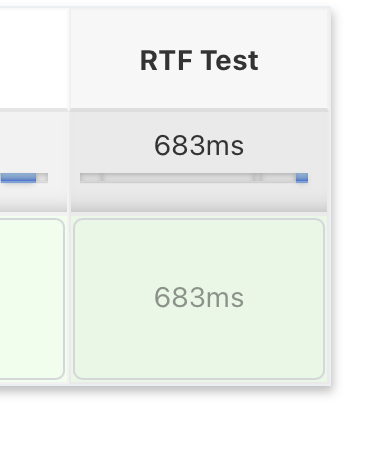
should contain ${res.text} Hello

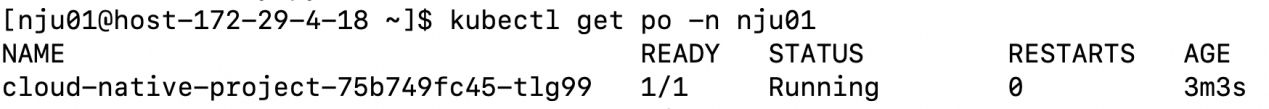
- 目的

将 python 部署到 k8s 上并且运行 rtf 测试脚本



运行结果

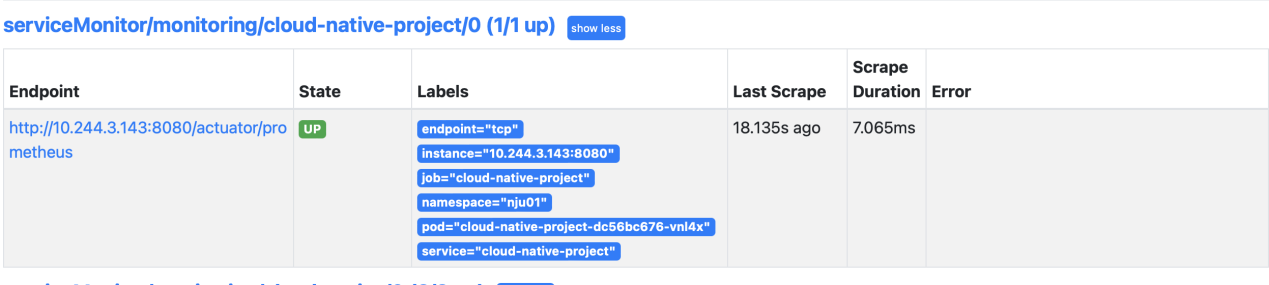


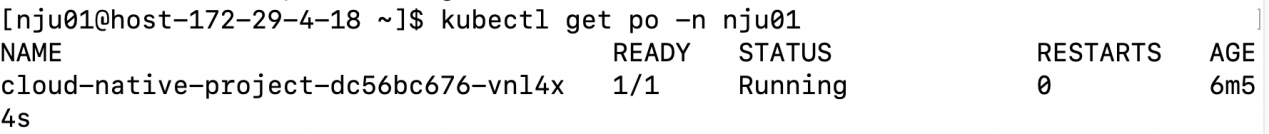


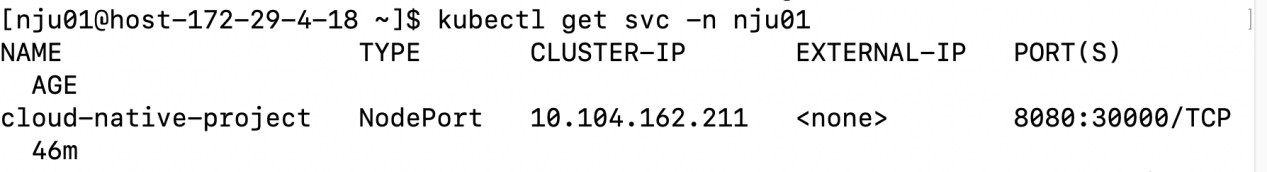
三． 扩容场景

1.  为该 Java 项目提供 Prometheus metrics 接口，可以供 Prometheus 采集监控指标









1. 在 Grafana 中的定制应用的监控大屏（CPU/内存/JVM）

