云原生大作业

团队成员：

- 许涛 171250579
- 詹志 171250609

分工：

- 许涛:SpringBoot应用编写，即第一步的所有工作
- 詹志:负责第二步的所有工作

第一步·构建SpringBoot应用

1. 构建SpringBoot应用 使用阿里云构建应用：https://start.aliyun.com
2. 选择限流算法 常见的限流算法有：计数器、漏桶和令牌桶算法。
   这里使用最简单的计数器，每次来一个请求计数器加一，如果计数大于限流值(最大可处理请求数，这里是100，即最大可以同时处理100个请求)
3. 编写注解类 编写注解类AccessLimit，参数是最大访问次数，可以理解同一时段最多访问 maxCount 次。

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface AccessLimit {
    int maxCount();
}
```

4. 拦截器AccessLimtInterceptor

```
@Component
public class AccessLimtInterceptor implements HandlerInterceptor {
    private int count=0;
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        if (handler instanceof HandlerMethod) {
            HandlerMethod hm = (HandlerMethod) handler;
            AccessLimit accessLimit = hm.getMethodAnnotation(AccessLimit.class);
            int maxCount = accessLimit.maxCount();
            if (count<maxCount){
                count++;
                System.out.println("count++ : "+count);
                return true;
            }else{
                System.out.println("429:Too many requests");
                response.setStatus(429); //429:Too many requests
                return false;
            }
        }
        return true;
    }
    @Override
```

```
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
        count--;
        System.out.println("count-- : "+count);
    }
}
```

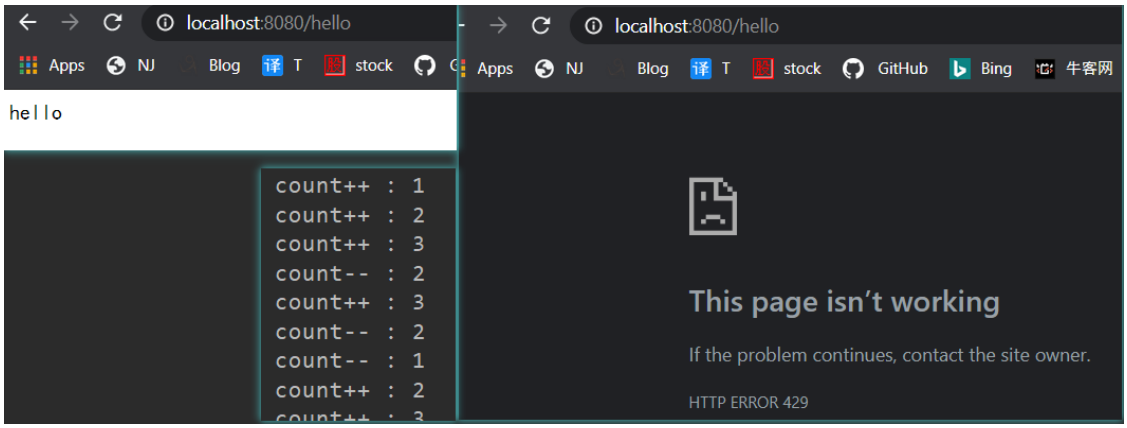
5. 拦截器配置 需要拦截/hello路由

```
@Configuration
public class IntercepterConfig  implements WebMvcConfigurer {
    @Autowired
    private AccessLimtInterceptor accessLimtInterceptor;
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(accessLimtInterceptor)
                .addPathPatterns("/hello");
    }
}
```

6. Controller中使用限流

```
@RestController
public class MyController {
    @GetMapping("/hello")
    @AccessLimit(maxCount = 100)
    public String hello(){
        return "hello";
    }
}
```

7. 效果验证 为了能看到效果，controller中处理每个请求时sleep 1秒，然后最大可以同时处理3个请求，于是可以看到效果：
在浏览器快速按刷新：



第二步 构建流水线，实现CI/CD

- 准备dockerfile 构建镜像：

```
FROM nat-harbor.daocloud.io/library/openjdk:8u232-jre-debian
ADD ./target/demo-0.0.1-SNAPSHOT.jar /app/demo-0.0.1-SNAPSHOT.jar
ADD runboot.sh /app/
WORKDIR /app
RUN chmod a+x runboot.sh
EXPOSE 8088
CMD /app/runboot.sh
```

- shell脚本

```
java ${JAVA_OPS} -Duser.timezone=Asia/Shanghai -
Djava.security.egd=file:/dev/./urandom -jar /app/demo-0.0.1-SNAPSHOT.jar
```

- 流水线

```
pipeline {
    agent none
    stages{
        stage('Clone to master') {
            agent {
                label 'master'
            }
            steps {
                echo "1.Git Clone Stage"
                git url: "https://github.com/jiangxizhanzhi/cn-final-
assignment.git"
            }
        }

        stage('Maven Build') {
            agent {
                docker {
                    image 'maven:latest'
                    args '-v /root/.m2:/root/.m2'
                }
            }
            steps {
                echo "2.Maven Build Stage"
                sh 'mvn test -Dtest -DfailIfNoTests=false'
                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/demo-0.0.1-SNAPSHOT.jar -t cn:${BUILD_ID} . '
                sh 'docker tag cn:${BUILD_ID}
harbor.edu.cn/cn202003/cn:${BUILD_ID}'
            }
        }
```

```
        stage('Push') {
            agent{
                label 'master'
            }
            steps {
                echo "4.Push Docker Image Stage"
                sh "docker login  --username=cn202003 harbor.edu.cn -p
cn202003"
                sh "cat ~/.docker/config.json |base64 -w 0"
                sh "docker push harbor.edu.cn/cn202003/cn:${BUILD_ID}"
            }
        }
    }

}

node('slave') {
    container('jnlp-kubectl') {
        stage('Git Cone') {
            git url: "https://github.com/jiangxizhanzhi/cn-final-
assignment.git"
        }

        stage('YAML') {
            echo "5. Change YAML File Stage"
            sh 'sed -i "s#lastest#${BUILD_ID}#g" ./cn.yaml'
            sh 'pwd'
        }

        stage('Deploy') {
            echo "5. Deploy To K8S Stage"
            sh 'kubectl apply -f ./cn-service.yaml -n cn202003'
            sh 'kubectl apply -f ./cn.yaml -n cn202003'
        }
    }
}
```

- 我在流水线里唯一碰到的坑就是k8s 拉取我push到harbor的镜像构建pod时会出现ErrorImagePull之类的错误，这个小点讲述怎么解决：

    1. 在项目中补充mytocken.yaml,用来在k8s中构建secret

    ```
    apiVersion: v1
    data:
      .dockerconfigjson:
    ewoJImF1dGhzIjogewoJCSJoYXJib3IuZWR1LmNuIjogewoJCQkiYXV0aCI6ICJZMjR5TUR
    JdO1ETTZZMjR5TURJdO1ETTOiCgkJfSwKCQkiaHROcHM6Ly9pbmRleC5kb2NrZXIuaW8vdj
    EvIjogewoJCQkiYXV0aCI6ICJjR3QxYmpwc2FUSXdNREJqYUhWdSIKCQl9Cgl9LAoJIkh0d
    HBIZWFkZXJzIjogewoJCSJVc2VyLUFnZW50IjogIkRvY2tlci1DbGllbnQvMTguMDkuOC1j
    ZSAobGludXgpIgoJfQp9
    kind: Secret
    metadata:
      name: mytoken
    type: kubernetes.io/dockerconfigjson
    ```

    2. 在mytocken.yaml中.dockerconfigjson字段我是在流水线的push 这个stage中添加了一个step,然后查看jenkins流水线运行过程中打印出的结果就可以了

```
sh "cat ~/.docker/config.json |base64 -w 0"
```

3. 在stage (deploy) 中执行

```
sh 'kubectl create secret -f ./mytoken.yaml -n cn202003'
```

4. 由于不需要重复生成secret，所以我第一次创建后就把这个过程在流水线中删除了

- 生成service

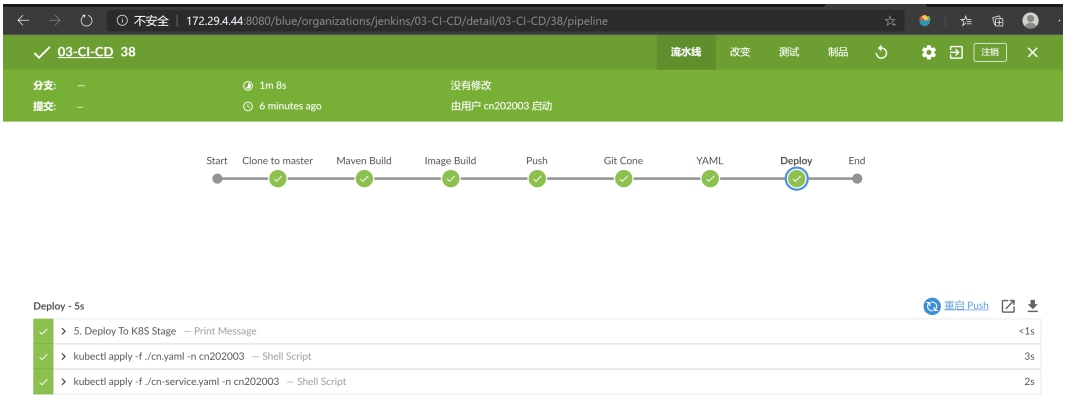
1. 添加cn-service.yaml脚本

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cn
  name: cn
  namespace: cn202003
spec:
  ports:
  - protocol: TCP
    port: 8088
    targetPort: 8088
  selector:
    app: cn
  type: NodePort
```

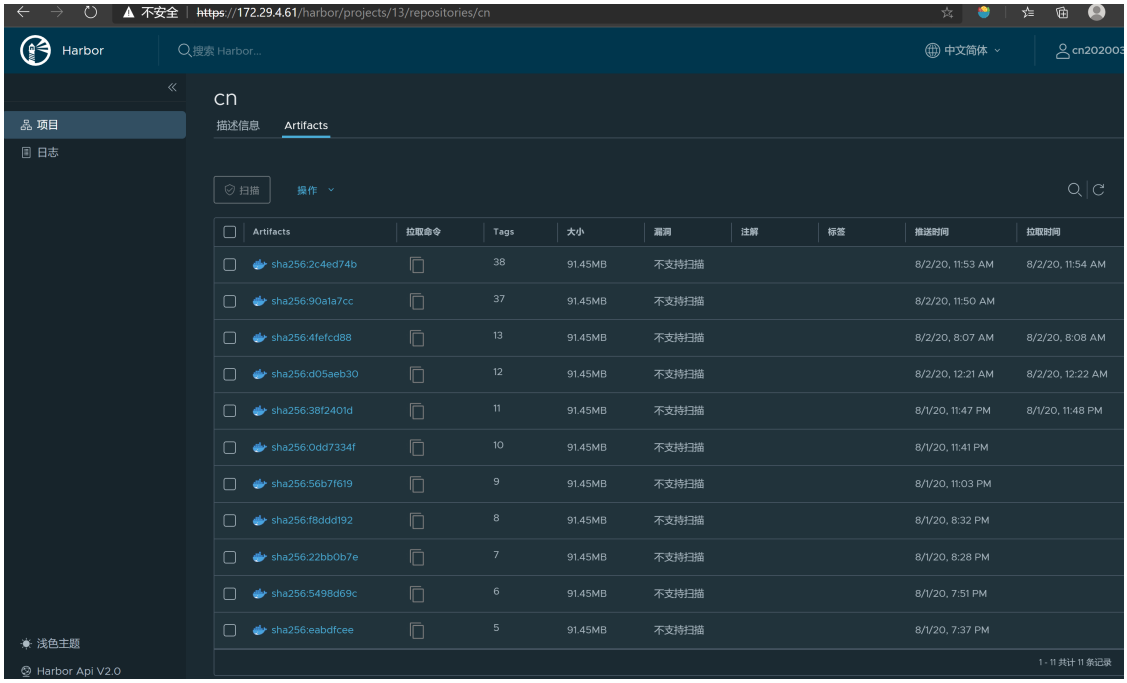
2. 在流水线中stage('Deploy') 添加step:

```
sh 'kubectl apply -f ./cn-service.yaml -n cn202003'
```

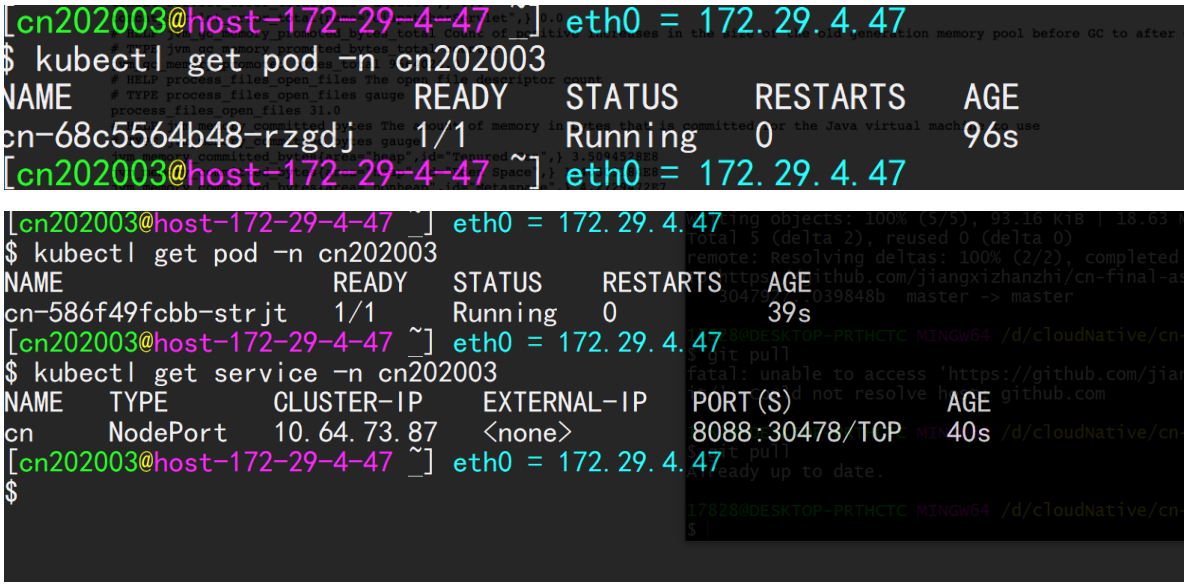
- 结果图：
  - 流水线运行成功：



- 镜像成功推送到harbor私人仓库



- k8s成功部署了pod 和service



- 访问部署的web程序，由于service 显示部署在172.29.4.47:30478，我们用浏览器访问

hello