Spring Cloud + K8s 最佳实践

Spring Cloud + K8s 最佳实践

1. 为什么要微服务化?

高可用可拓展，快速迭代，可复用，把 K8s 当作操作系统。

2. 微服务带来的成本

数据一致性，分布式事务，复杂度增加，链路变长带来的时延

3. 如何拆分微服务?

单一职责

4. Spring Cloud 与 istio 的区别与联系

Spring Cloud + K8s 最佳实践

Spring Cloud 微服务应用最为核心的部分包括服务注册中心、配置中心、应用网关等等。

spring cloud 和 K8s 在核心功能上有很大一部分交集，比如 K8s 中 service 为我们提供了服务注册、负载均衡的能力，configmap 和 secret 为我们提供了配置管理相关的功能，我们还可以使用 ingress controller 作为项目的网关，

我会以一个简短的 demo 进行展开，给大家带来 Spring Cloud 微服务在 K8s 环境中的最佳实践，借助 Spring Cloud 社区开源的 spring-cloud-kubernetes 这个项目，我们可以将 K8s 当作 Spring Cloud 微服务应用的注册中心、配置中心，以此来简化微服务应用的整体架构，更好的和云原生环境进行融合。

spring-cloud-kubernetes

spring-cloud-kubernetes 是 Spring Cloud 社区为 K8s 环境，提供的开箱即用的服务发现、配置分发的方案。

该项目实现了的 Spring Cloud 中几个核心的接口，允许开发者在 Kubernetes 上构建和运行 Spring Cloud 应用。贴出具体的项目链接，大家课后可以仔细阅读相关文档。

https://github.com/spring-cloud/spring-cloud-kubernetes
https://spring.io/projects/spring-cloud-kubernetes

一、Starters

通过导入 spring-cloud-starter-kubernetes 可以为我们提供开箱即用的功能，服务发现 和 配置管理两部分内容。可以通过添加不同的依赖包，启用不同的功能特性，按需加载。

1. 该项目实现了 SpringCloud 中 Discovery Client 接口，可以从 K8s 的 service 中获取 endpoint 信息，实现服务注册与服务发现功能

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-kubernetes-fabric8</artifactId>
</dependency>
```

2. 可以从 K8s 的 ConfigMap、Secret 中动态加载应用配置，并实现 reload

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-kubernetes-fabric8-config</artifactId>
</dependency>
```

3. 一键启用所有特性

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-kubernetes-fabric8-all</artifactId>
</dependency>
```

二、服务发现

其中 `spring-cloud-starter-kubernetes-fabric8` 项目为 Kubernetes 提供了客户端服务发现的实现。可以从客户端按名称查询 Kubernetes 中的 service 关联的 endpoint（这里需要参考K8s中服务的相关概念），客户端如果运行在 K8s 集群中则可以直接访问这些 endpoint，还可以在此基础之上实现负载均衡。

我们可以通过以下命令查看服务的具体信息，查看 K8s 中 service 具体关联的 endpoint。

```
kubectl get services
kubectl get endpoint
```

K8s 通过 service 提供了服务发现（server side）的能力（请参阅：https://kubernetes.io/docs/concepts/services-networking/service/#discovering-services)。使用原生的 K8s 服务发现可确保与其他工具（如Istio）的兼容性，Istio是一种能够实现负载均衡、熔断器、故障切换等功能的服务网格工具，我的同事也会在本期课程中进行展开讲解。

服务的调用方只需要引用在集群中可以解析的域名，比如 kubernetes.default.svc。
通常格式如下：

`{service-name}.{namespace}.svc.{cluster}.local`

二、配置管理

通常我们会使用 application.yaml、application-profile.yaml 文件对 Spring Boot 应用进行配置，配置文件中包含了一些应用配置相关的键值对。

在 K8s 中我们可以直接使用 ConfigMap 挂载配置文件到运行的 pod 中，或者使用 `spring-cloud-starter-kubernetes-fabric8-config` 将对应的配置文件加载到应用程序中，配置文件还支持 reload 特性，可以在不重启情况下进行配置变更。

三、负载均衡

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-kubernetes-fabric8-
loadbalancer</artifactId>
</dependency>
```

该项目提供了基于 K8s service、endpoint 的负载均衡实现。

四、选主

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-kubernetes-fabric8-leader</artifactId>
</dependency>
```

该项目使用 K8s ConfigMap 实现了 Spring Integration 的选主 API。

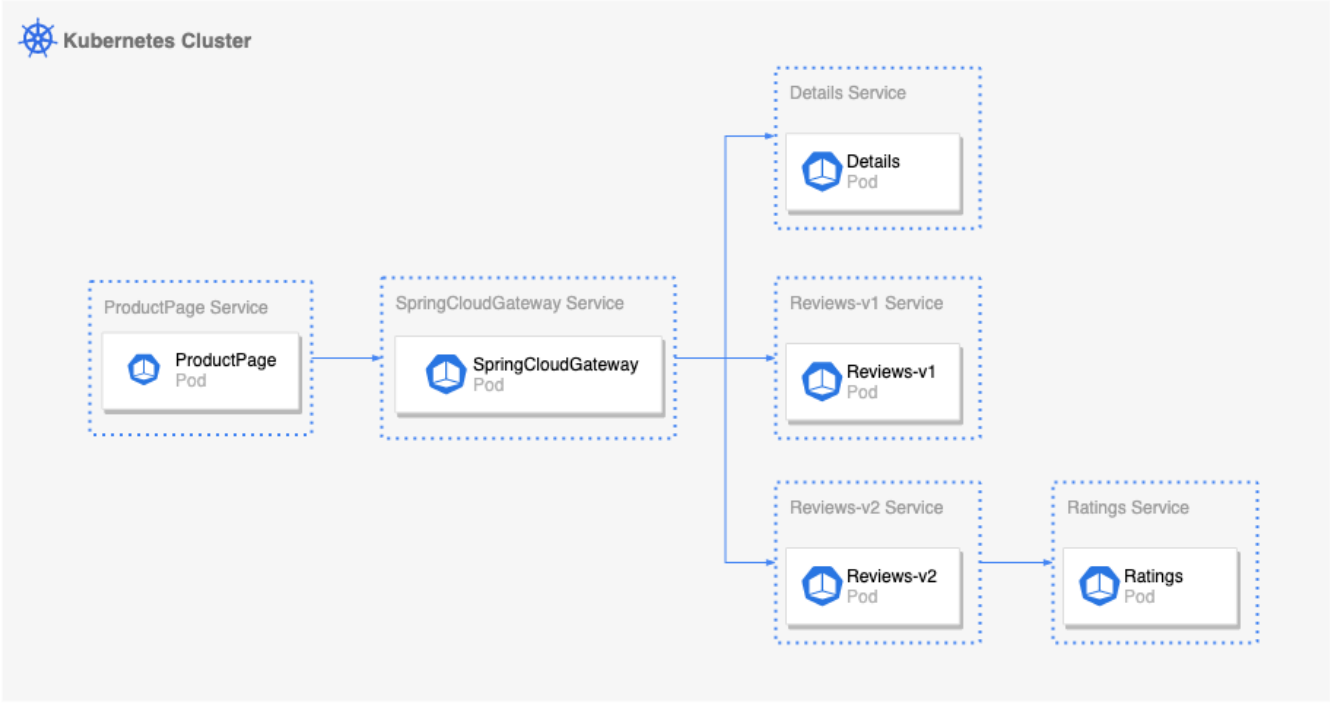
在多副本需要主备切换的情况下有非常大的帮助，不用从头去实现选主功能，极大的简化了开发工作。

---

spring-cloud-bookinfo

通过前面的介绍，相信大家对 spring-cloud-kubernetes 有了一个基本的了解，接下来我会以 springcloud bookinfo 为例做一些 demo。

springcloud bookinfo 是从 istio 社区 copy 的一个简单的微服务应用示例，整体架构如图



大家可以从 GitHub 上查看这个 demo 项目

https://github.com/kubesphere-sigs/spring-cloud-bookinfo

整个微服务应用中包含了 5 个组件

1. productpage 是一个由 react 开发的前端组件
2. gateway 是一个由 spring-cloud-gateway 提供的 API 网关服务
3. details 是一个 spring-cloud 微服务，提供了书籍详情 API
4. reviews-v1 提供了基础的书籍评论信息，review-v2 在 review-v1 的基础之上额外的提供了评分数据，依赖 ratings 服务
5. ratings 是一个 golang 开发的微服务组件

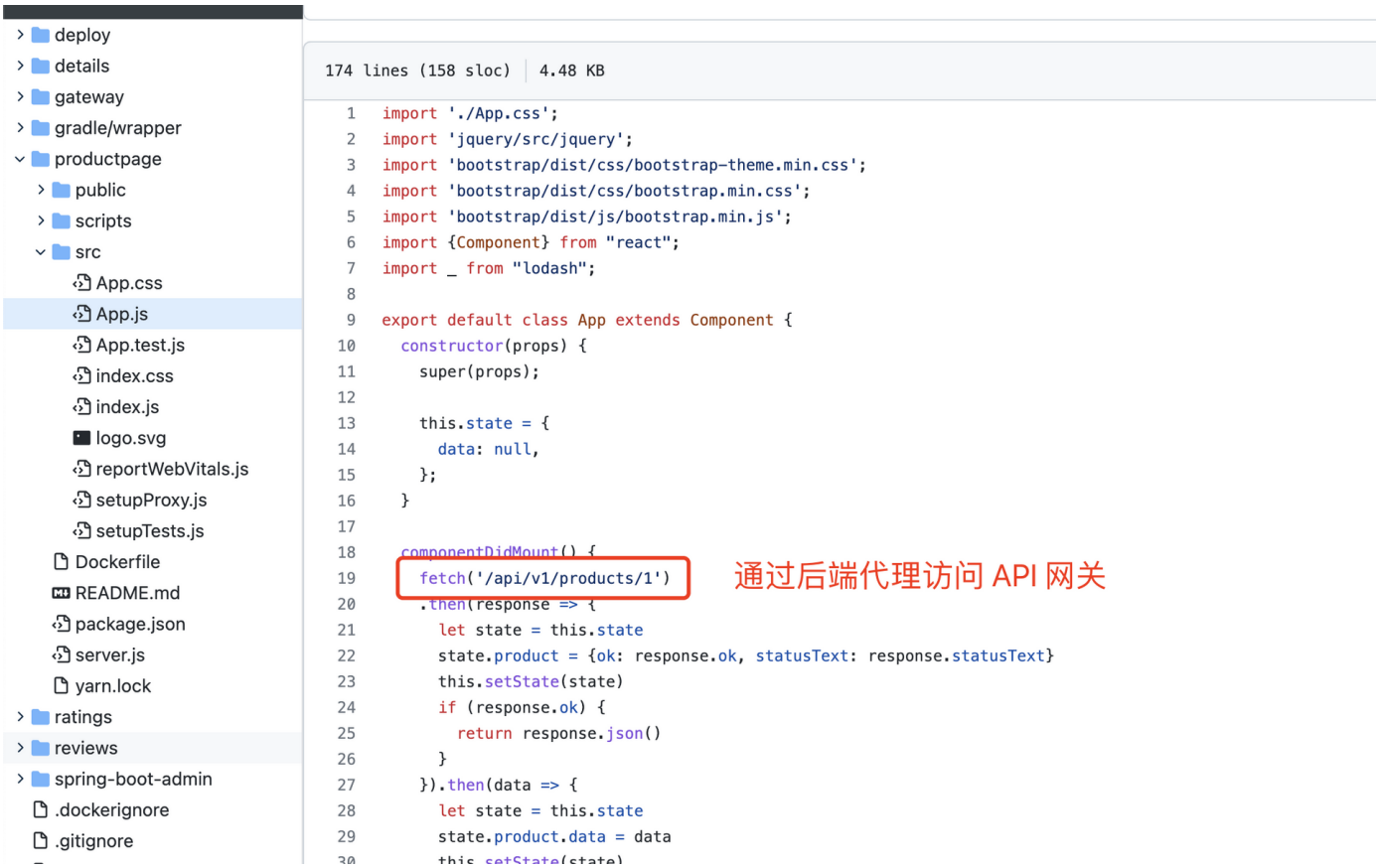
下面带着大家来将这个微服务应用部署到我们的 K8s 集群中

1. 创建 spring-cloud namespace
2. 对 default serviceaccount 进行授权，允许微服务组件从 K8s 中获取数据，实现服务发现，获取配置文件

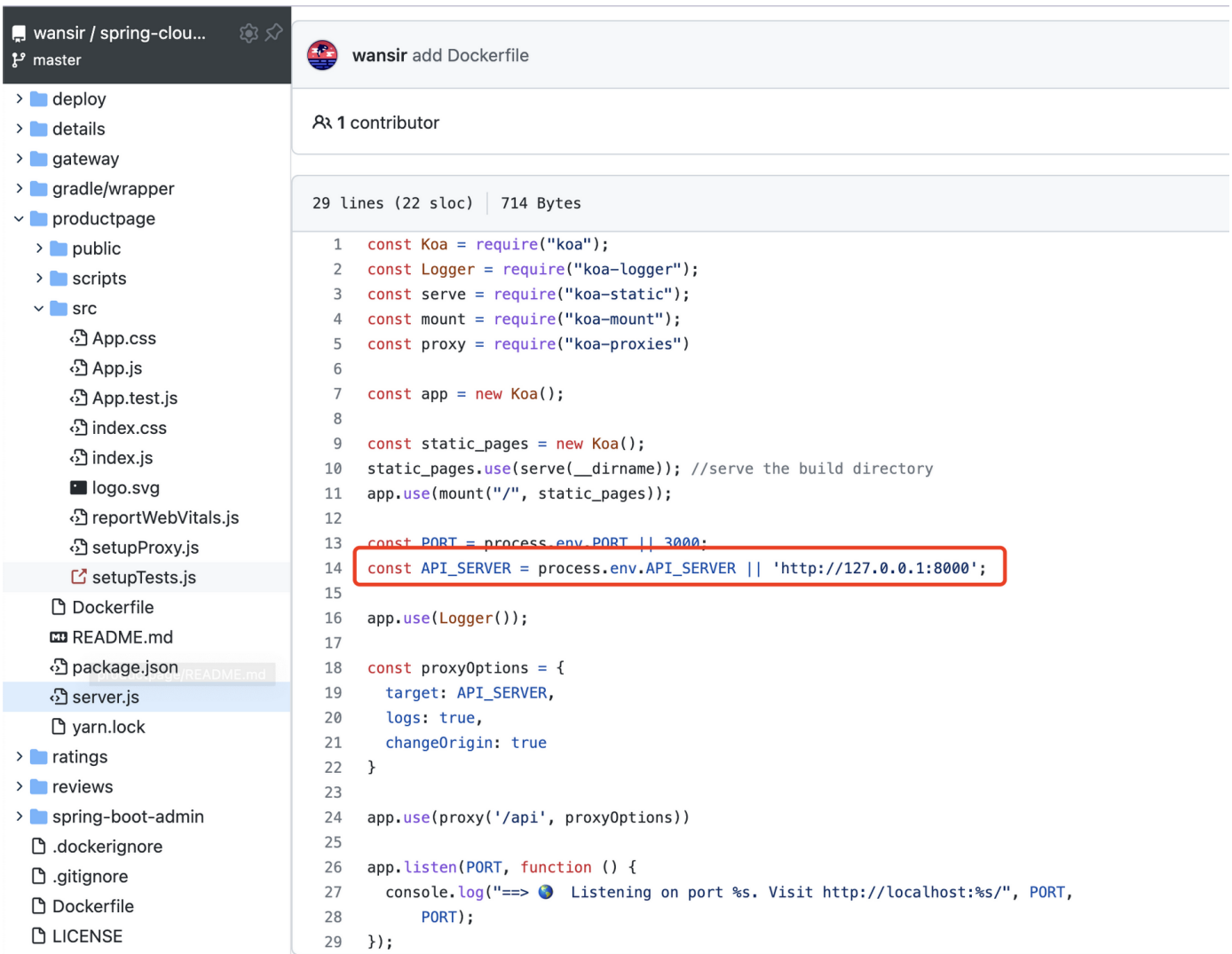
一、productpage

是一个 react 开发的前端项目，是本示例中的前端访问入口，先带大家了解一下项目结构

通过反向代理 访问 API 网关



通过环境变量配置 API 网关地址



事先已经构建好了镜像，我们通过 kubesphere 进行部署

1. 创建工作负载并通过环境变量指定网关地址
2. 创建服务并指定工作负载
3. 开启项目网关，创建应用路由对外提供访问
4. 打开前端访问页面

二、gateway

是一个 spring-cloud-gateway 应用，作为后端 API 的入口

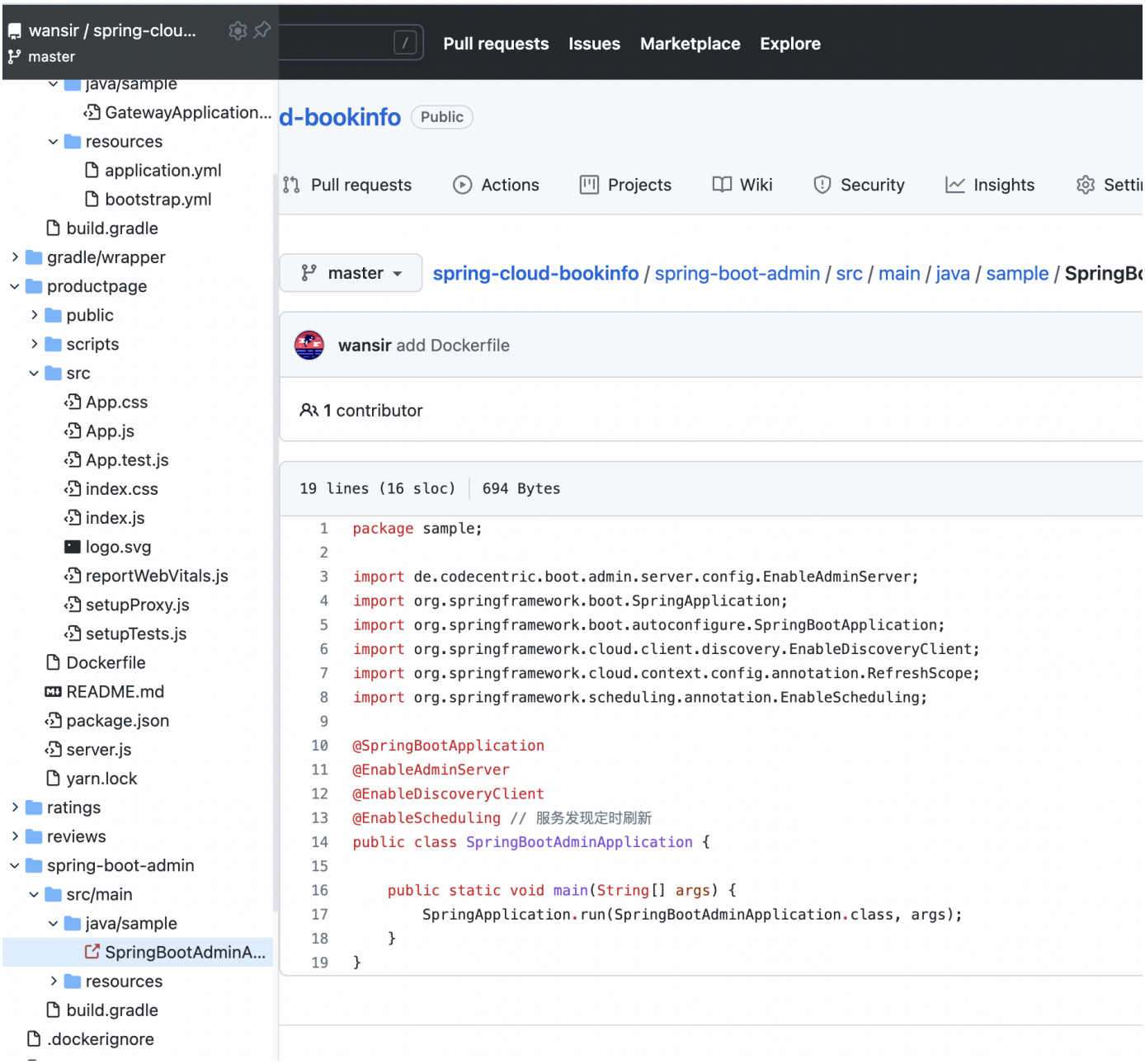
添加相关依赖并启用 spring-cloud-gateway

进行部署

1. 创建配置文件：注意开启管理相关的 API 访问
2. 创建工作负载
3. 创建服务
4. 创建应用路由
5. 打开访问页面

三、spring-boot-admin

可以借助 spring-boot-admin 管理我们的微服务应用，需要启用 discovery client 和定时任务



紧接着部署 spring-boot-admin

1. 创建配置文件
2. 创建工作负载
3. 创建服务
4. 创建应用路由
5. 打开访问页面，检查前面部署的 gateway 是否正常注册

四、details

details 提供了具体的书籍详情 API，我们可以通过 product id 获取书籍的详细信息

部署 details 应用

1. 创建配置文件

2. 创建工作负载

3. 创建服务

4. 打开 spring-boot-admin 检查服务是否正常注册

5. 通过 spring-boot-admin 配置路由规则

```
        - id: details
          uri: lb://details
          predicates:
            - Path=/api/v1/products/*
```

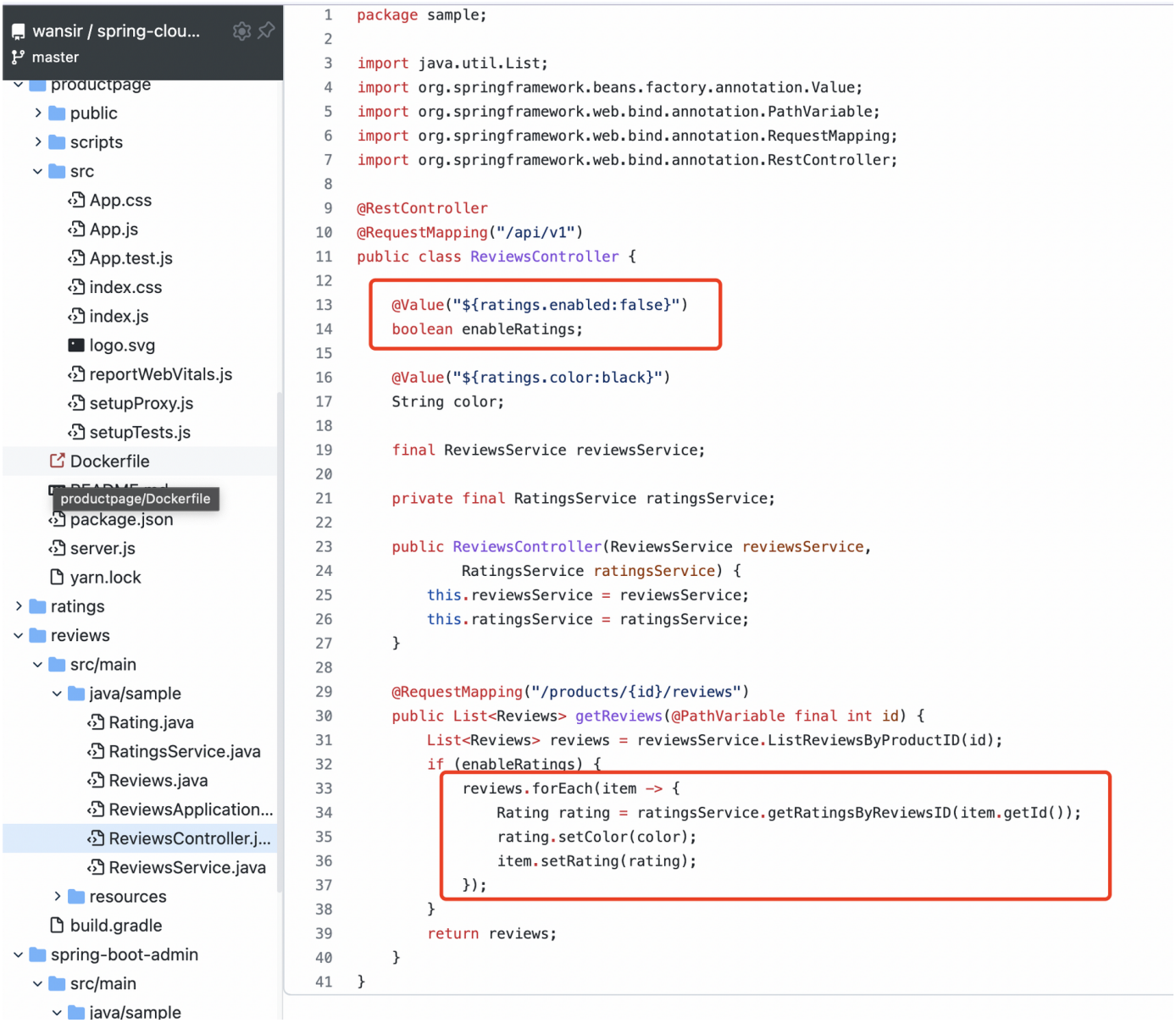
6. 检查路由规则是否生效

```
/api/v1/products/1
```

7. 检查 productpage 中书籍详情是否正常显示

五、reviews-v1

reviews 应用提供书籍评论相关的 API，可以通过配置开启是否展示评分



我们先部署 v1 版本的 reviews 后端服务，默认不展示书籍评分

1. 创建配置文件： 注意禁用评分功能
2. 创建工作负载
3. 创建服务
4. 打开 spring-bootadmin 检查服务是否正常注册
5. 通过 spring-boot-admin 配置路由规则

```
        - id: reviews-v1
          uri: lb://reviews-v1
          predicates:
            - Path=/api/v1/products/*/reviews
```

6. 检查路由规则是否生效

```
/api/v1/products/1/reviews
```

五、ratings

review-v2 为书籍提供更详细的评论信息，包含评分数据，依赖 ratings 提供评分相关的数据

ratings 是一个简单的 golang 开发的应用， spring cloud kubernetes 可以为非 spring-cloud 应用提供负载均衡

1. 创建工作负载
2. 创建服务
3. 通过 spring-boot-admin 配置路由规则

```
        - id: ratings
          uri: lb://ratings
          predicates:
            - Path=/api/v1/reviews/*/ratings
```

4. 检查路由规则是否生效

六、reviews-v2

通过 spring-cloud-gateway 的 WeightRoutePredicateFactory 实现简单的灰度发布，根据不同的版本配置流量权重

创建 v2 版本的 reviews 服务

1. 创建配置文件： 注意启用 ratings
2. 创建工作负载
3. 创建服务

4. 通过 spring-boot-admin 检查服务是否正常注册
5. 配置路由规则，v1、v2 各占一半的权重

```
        - id: reviews-v1
          uri: http://reviews-v1
          predicates:
            - Path=/api/v1/products/*/reviews
            - Weight=reviews, 50
        - id: reviews-v2
          uri: http://reviews-v2
          predicates:
            - Path=/api/v1/products/*/reviews
            - Weight=reviews, 50
```

6. 打开 productpage ，刷新界面，预期 review-v1/review-v2 的概率各占一半

总结

spring-cloud-kubernetes 简化了 K8s 环境下 spring-cloud 应用的组件依赖，提供了开箱即用的服务发现和配置管理能力，为 java 微服务应用提供了良好的基础。