

项目文档

西安电子科技大学 XenoWYC121

1. 目标描述

1.1 题目描述

容器Linux操作系统（简称为容器OS）是为容器运行而设计的轻量级linux操作系统，作为集群worker节点的host OS时，可以由k8s等容器编排系统进行管理，实现容器和容器OS的统一管理。为了提高容器OS的安全性和集群环境的一致性，容器OS通常不会包含ssh，但是在实际使用时用户可能仍需要对节点的某些配置进行修改，所以期望通过k8s集群将配置下发到各个worker节点上。k8s operator是k8s提供的一种扩展机制，可以自定义集群资源和控制器等。所以请基于k8s operator设计并实现一套完整的流程，实现通过k8s集群统一下发配置到worker节点上的功能。

1.2 题目要求

基本目标：

- 使用k8s的operator机制，设计crd和controller等相关组件及组件之间的通信方式
- 完成crd及controller等相关组件的代码开发
- 实现配置文件和命令通过k8s集群的master节点统一下发到集群的worker节点上执行，比如完成worker节点 host OS 上 docker 镜像仓库、代理等配置

加分项：

- 安全性：设计合理的机制防止命令注入等问题
- master节点独立配置：对于多master节点的高可用集群，支持对master节点下发配置，并且区别master节点和worker节点，下发不同的配置

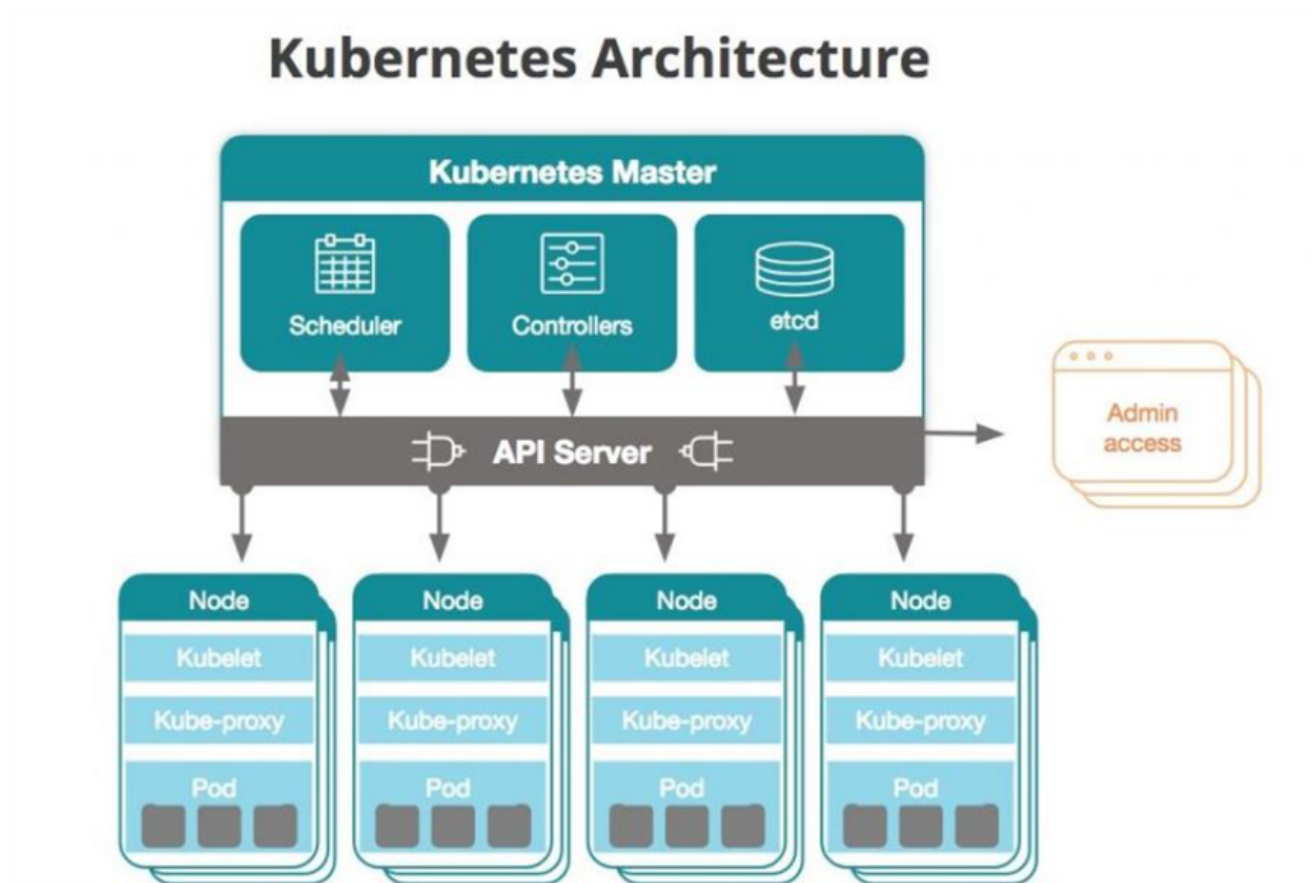
2. 比赛题目分析和相关资料调研

2.1 k8s 简介

Kubernetes（简称k8s）是Google在2014年6月开源的一个容器集群管理系统，使用Go语言开发，用于管理云平台中多个主机上的容器化的应用，Kubernetes的目标是让部署容器化的应用简单并且高效,Kubernetes提供了资源调度、部署管理、服务发现、扩容缩容、监控，维护等一整套功能。，努力成为跨主机集群的自动部署、扩展以及运行应用程序容器的平台。它支持一系列容器工具, 包括Docker等。

K8S的特点：

- 开源的容器编排平台，用于自动化部署、扩展和管理容器化应用程序；
- 提供了一种可靠和可扩展的方式来运行、管理和编排容器，以实现高可用性、弹性伸缩和故障；
- 旨在解决传统部署应用程序所面临的挑战，例如复杂的应用程序依赖关系、扩展性和高可用性要求；
- 通过使用声明式配置文件定义应用程序的期望状态，并确保系统根据这些配置自动进行调度和管理恢复。



2.2 k8s应用场景

- **微服务与无服务器计算架构**：每个服务均可封装在一个单独容器内，方便管理与扩展；
- **云原生应用**：Docker+K8S可方便实现云环境下的资源弹性伸缩、故障检测、负载均衡等功能；
- **开发环境标准化**：统一的Docker镜像搭建一致的开始环境，K8S用于编排容器；
- **持续集成与部署（CI/CD）**：加速代码从提交到部署上线的过程。
 - CI: Continuous Integration
 - CD: Continuous Deployment

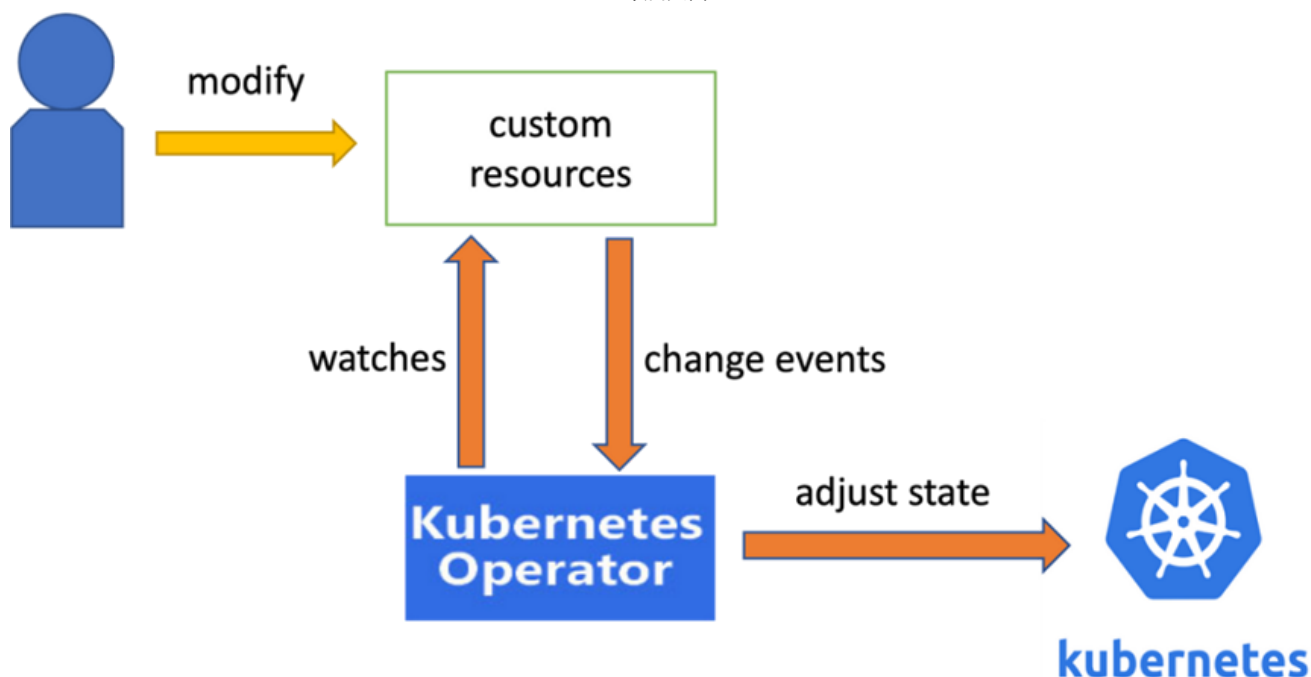
2.3 docker与k8s的比较

特性	Docker	Kubernetes
目的	用于打包和运行应用程序的容器化平台	用于管理和扩展容器化应用程序的容器编排平台
重点	创建、打包和运行单个容器	跨集群管理和编排大量容器
架构设计	具有简单CLI客户端-服务器架构	具有主节点和工作节点的复杂架构
可扩展性	适合少量节点的应用程序	旨在处理跨多个节点的大规模、复杂应用程序
编排	有限的内置编排功能	提供高级编排功能，例如自动扩展、自我修复和负载均衡
配置	用于构建容器镜像的Dockerfile	用于定义程序所需状态的YAML或者JSON配置文件
网络	为容器提供基本的网络能力	提供高级网络功能，例如服务发现和负载均衡
学习曲线	比较简单容易上手	由于复杂的架构与概念，学习曲线更陡峭
部署	专注于部署单个容器	管理跨集群的容器化应用程序的部署与扩展
一体化	可以独立使用或与其他工具集成使用	通常与docker结合使用作为容器运行时
适用场景	比较小的应用程序或开发环境	需要高可用性和可扩展性的大型生产级应用程序

2.4 k8s operator

2.4.1 k8s operator 简介

Kubernetes Operator 基于 Kubernetes 的资源和控制器概念构建。它使用自定义资源（Custom Resource, CR）来表示和管理应用程序，同时通过自定义控制器（Custom Controller）来监控资源状态并执行管理操作。



当用户修改自定义资源时，operator可以监听到用户的修改，并根据资源的原始状态与用户的修改，调整自定义资源与K8S的状态。

2.4.2 k8s operator 工作原理

Operator 的工作原理可以概括为以下几个步骤：

1. 定义自定义资源（CRD）：Operator 首先需要定义一种或多种自定义资源，这些资源代表了要管理的应用程序或服务的配置和状态。
2. 实现自定义控制器：控制器是 Operator 的核心，负责监控指定的资源，当资源状态发生变化时，控制器会根据资源的当前状态和期望状态来调整，确保应用或服务处于正确的状态。
3. 自动化操作逻辑：控制器中会编码应用管理的业务逻辑，如升级、备份、恢复等操作，这些逻辑以前可能需要人工介入执行，现在可以自动完成。

2.4.3 k8s operator的优势

1. 自动化管理：Operator 可以自动进行应用部署、更新、备份和恢复等复杂操作，减少了人工错误和操作成本。
2. 深度集成：Operator 深度理解其管理的应用，能够提供更智能的管理，如故障自动恢复、自动横向扩展等。
3. 易于扩展：通过添加新的 Operator，用户可以轻松扩展 Kubernetes 集群的管理能力，支持更多类型的应用或服务。

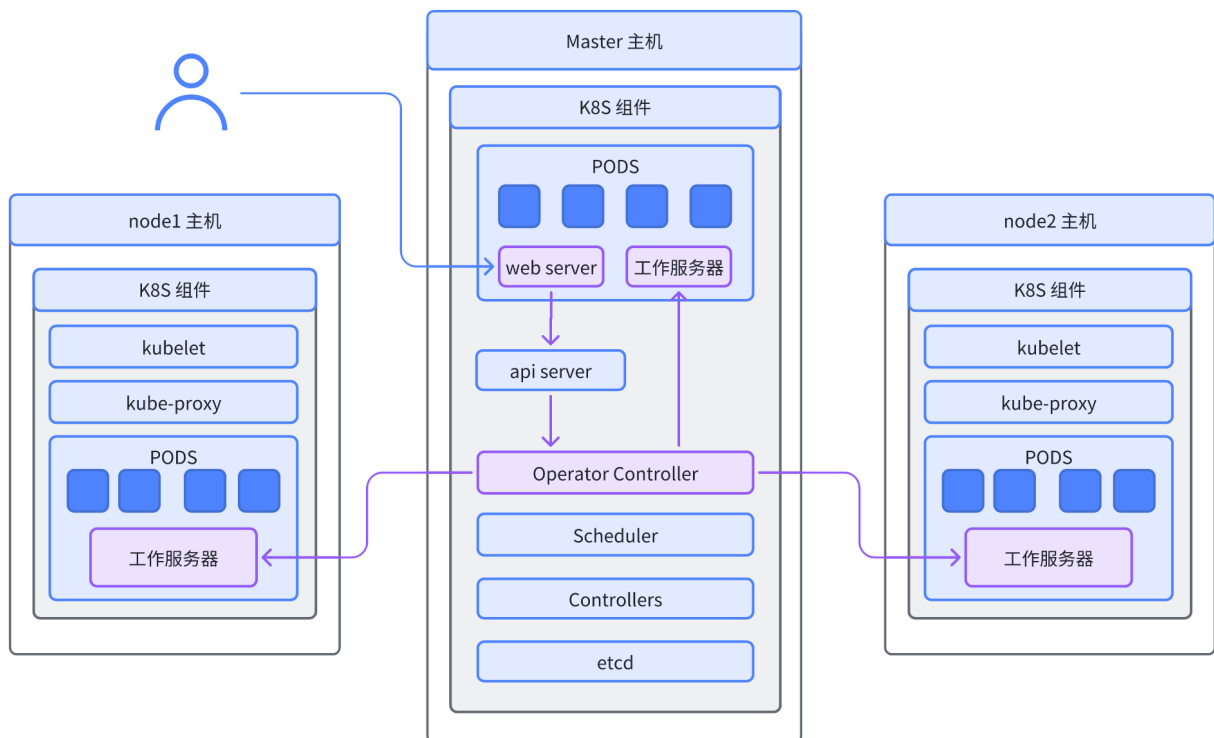
3. 系统框架设计

Master与worker节点之间使用HTTP协议通信，由K8S operator controller作为客户端发送请求，worker节点运行HTTP服务器负责接受请求、检查请求合法性、完成操作以及响应请求。web server运行web界面，使配置过程图形化，更加方便。

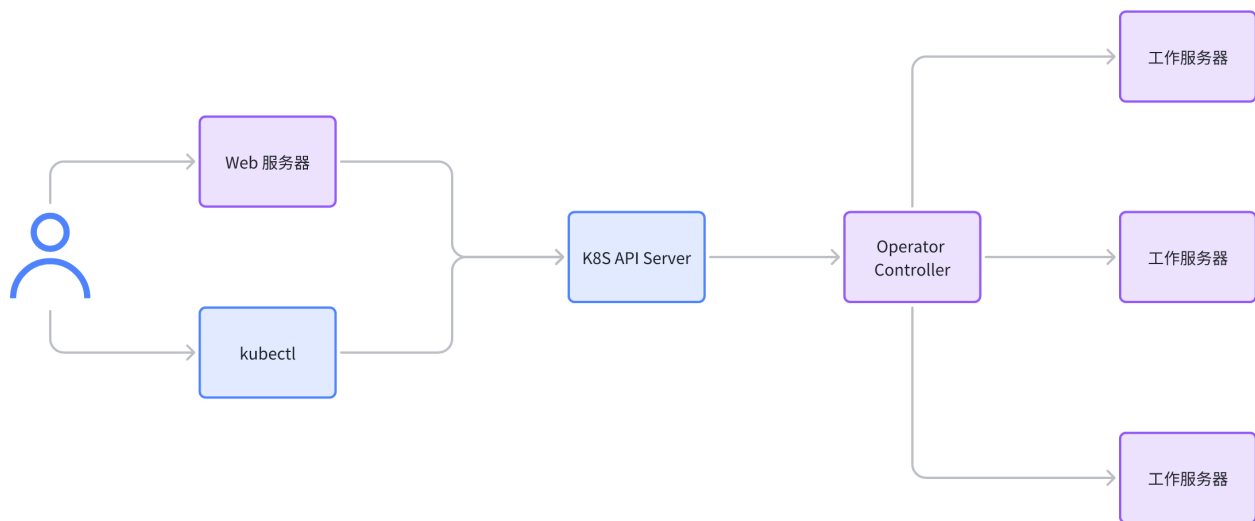
由于需要对集群中的工作节点物理机本身进行操作，所以HTTP服务器需要运行于物理机上，不能运行与k8s pod中。

3.1 架构图

3.1.1 系统架构（Controller容器化之前）



3.1.2 主要流程

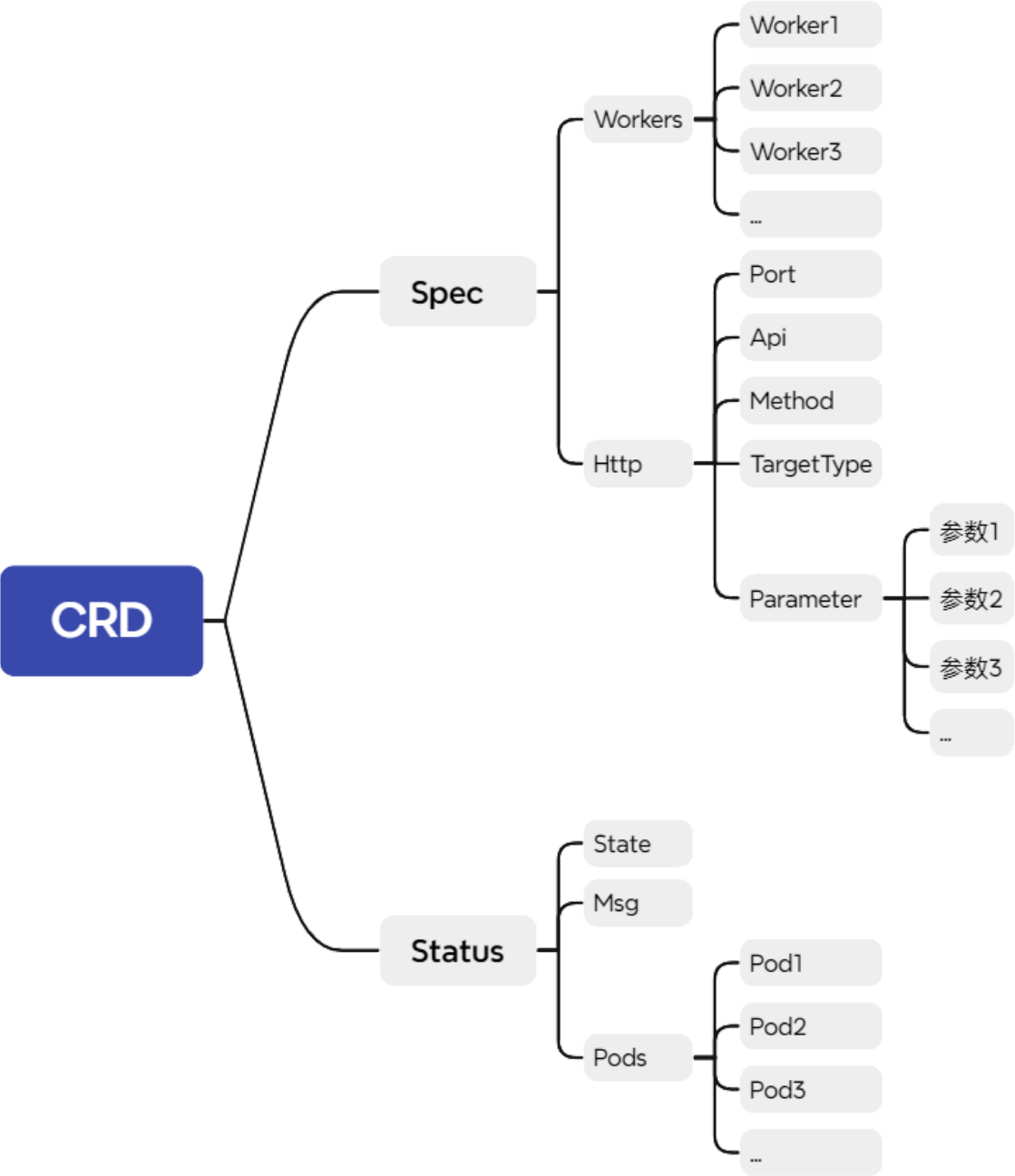


用户使用web服务器或是kubectl命令行工具向K8S API Server提交资源定义，API Server将调用Controller处理Operator资源。Operator Controller接收到定义后，解析目标、筛选目标、检查合法性、构造http请求，最后将请求发送给相应的工作服务器，完成配置下发。

3.2 自定义资源设计

自定义资源需要以下字段：

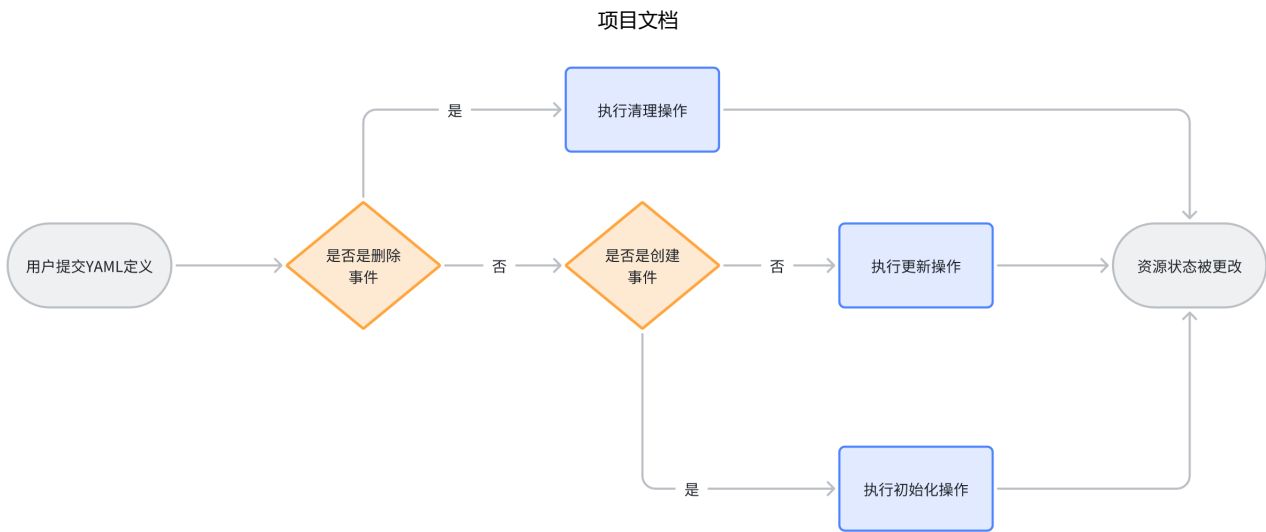
- Spec：描述了资源的期望状态——希望资源所具有的特征。当创建Kubernetes资源时，必须提供对象的规约，用来描述该资源的期望状态，以及关于对象的一些基本信息。
 - Workers：代表下发操作的节点（不区分Master与Node）。
 - Http：代表了请求的方法、参数与路径，worker在受到请求后，会执行相对应的操作。
 - TargetType：Http字段中的特殊字段，代表了下发操作的节点类型，可以是Master也可以是Node。
- Status：存储了资源当前的状态，由Operator维护，由于此资源用于下发操作，不需要硬性的状态信息，本字段用于存储操作的执行结果。



3.3 operator controller 设计

3.3.1 Controller主要流程

在阶段二中，我们对Controller功能进行了增强，导致其执行流程与阶段一相比有较大的改变。阶段二中的整体执行流程如下所示：

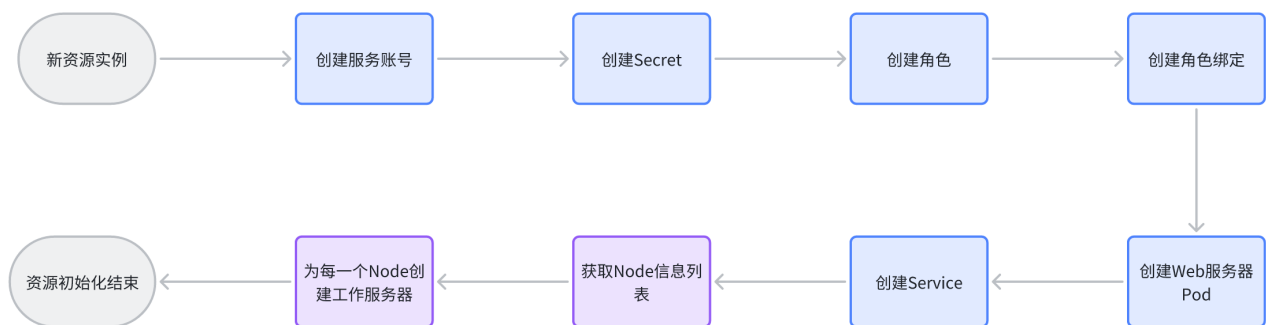


首先Controller将判断是否是删除事件触发了Controller执行，通过在Controller内部读取上下文的删除关键字即可判断。如果为删除事件，将会执行清理操作。

如果不是删除事件，Controller将会继续判断是否为创建时间，此时可以通过读取对象的Status状态，状态为空时可以认为是在创建资源，将会执行初始化操作。

最后在既不是删除也不是创建事件时，则可以认为是更新事件，此时只需完成更新操作，完成配置的下发任务即可。

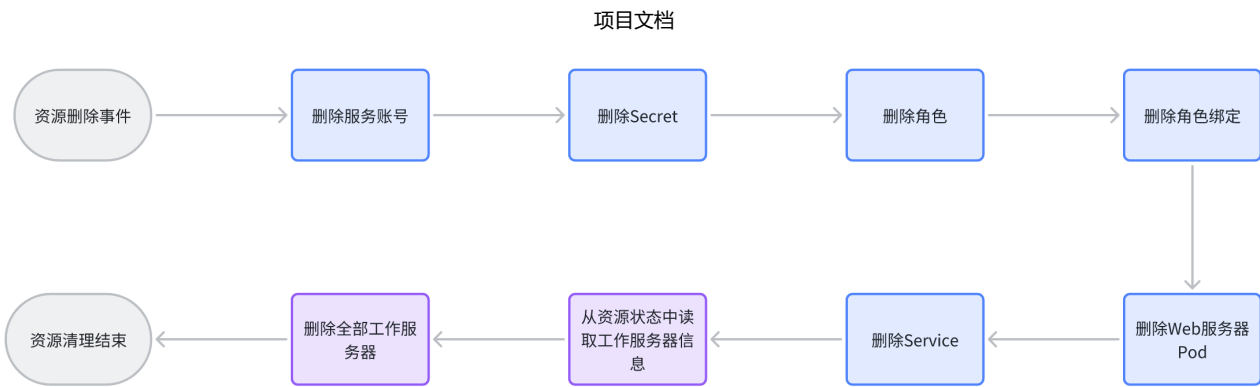
3.3.2 初始化操作



以上所有的操作完成后都存在一步结果判断过程，在此处省略。

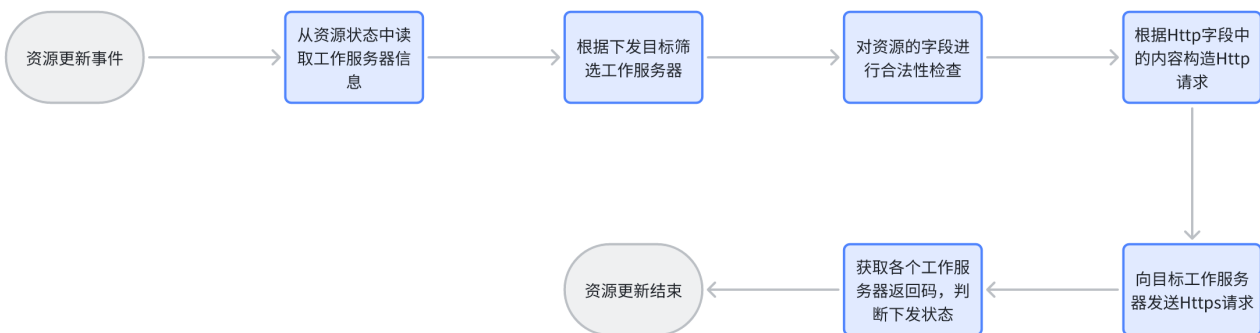
在初始化阶段的主要目标就是创建全部组件以及相关依赖项，蓝色方框代表了创建Web服务器以及相关依赖的操作，紫色方框代表了创建各个Node上的工作服务器的相关操作。

3.3.3 清理操作



以上所有的操作完成后都存在一步结果判断过程，在此处省略。清理操作的流程基本上就是初始化操作的逆操作，删除全部组件及其依赖的过程。

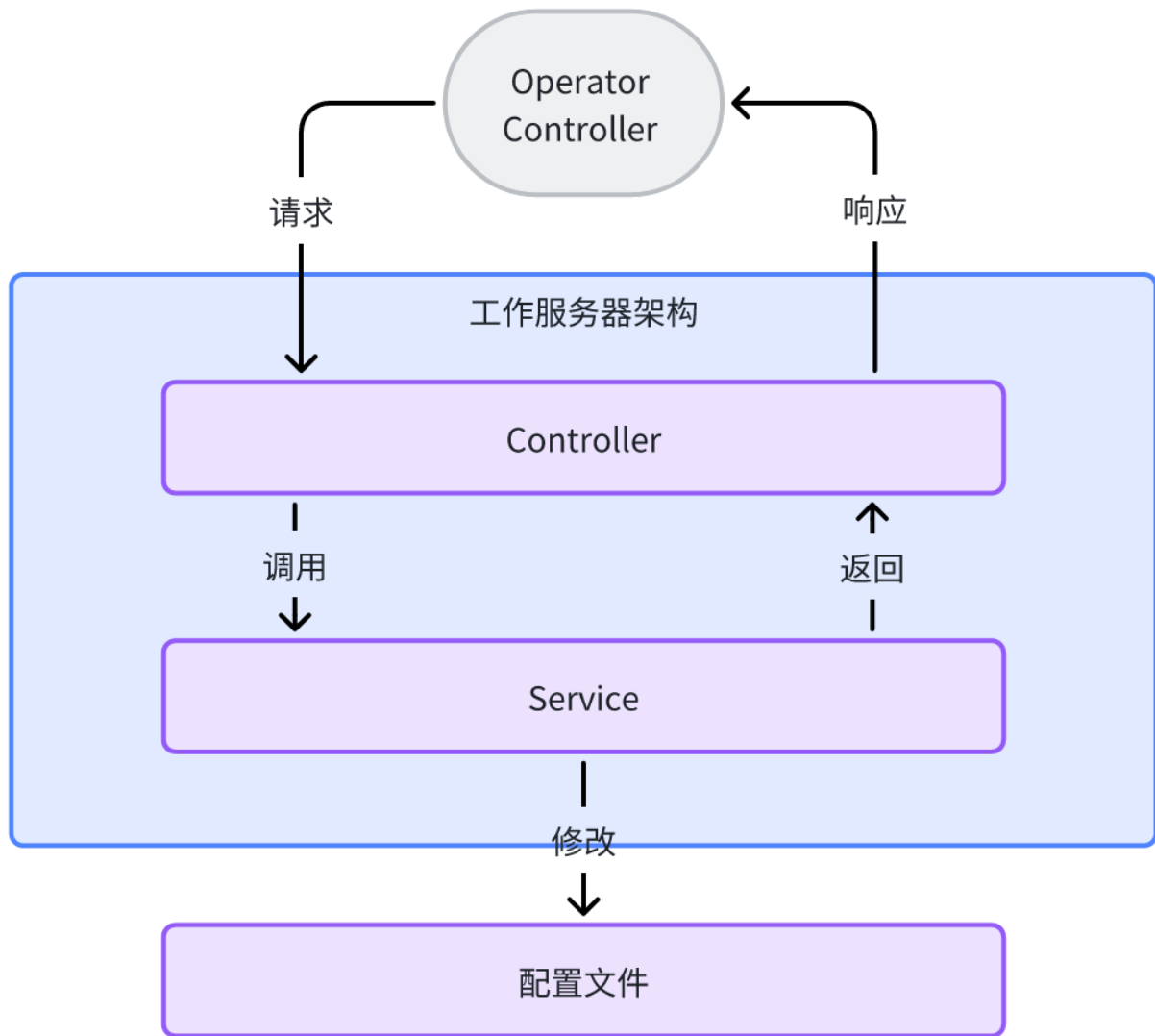
3.3.4 更新操作



以上所有的操作完成后都存在一步结果判断过程，在此处省略。更新操作就是下发配置的主要流程。在获取到工作服务信息后，首先根据下发配置的节点类型筛选工作服务器，之后进行合法性检查、构造http请求，最后发送给目标并获取执行结果。

3.4 工作服务器设计（容器化之前）

工作服务器为Http服务器，负责接收Controller发送来的Http请求，请求中包含所需下发的配置，工作服务器解析后修改主机上的配置文件，完成修改过程。



3.5 安全性:

命令注入防止

在Operator Controller和HTTP服务器端分别实现了对请求与参数的正则匹配，匹配的关键字包括常见Linux注入命令与特殊符号。通过对用户输入的预先处理，防止命令注入。

组件间通信

组件于组件之间使用HTTPS协议通信。确保数据传输时的安全性，防止数据被窃取或篡改。

防火墙配置

通过查看k8s api server日志，至少有以下ip 需要与 api server进行通信：

- Node IP

- Pod IP
- Cluster IP

因此在配置防火墙时，需要将以下的IP添加至白名单中：

- 集群中各个节点的IP地址
- Pod IP的CIDR（集群创建时指定）
- Cluster IP的CIDR（由api server的启动参数—service-cluster-ip-range配置）

关于防火墙的相关配置信息见文档《防火墙配置脚本》

Network Policy

使用K8S内置的Network Policy资源，限制外界对工作服务器的访问。

Network Policy是一种用于控制Pod之间以及Pod与其他网络端点之间通信的策略。它允许用户定义规则来指定哪些Pod可以互相通信，哪些不能。Network Policy的工作机制类似于防火墙规则，能够基于标签选择Pod，并设置允许或拒绝的通信流量。

为各个工作服务器pod设置network policy，使其只能被Controller pod访问，外界无法访问。

3.6 节点独立配置

自定义资源的Spec.Http.TargetType字段，定义了本条请求下发的主机类型（Master、Node或者All），只有对应类型的主机才会接受到该请求。

3.7 Operator功能：节点docker镜像配置

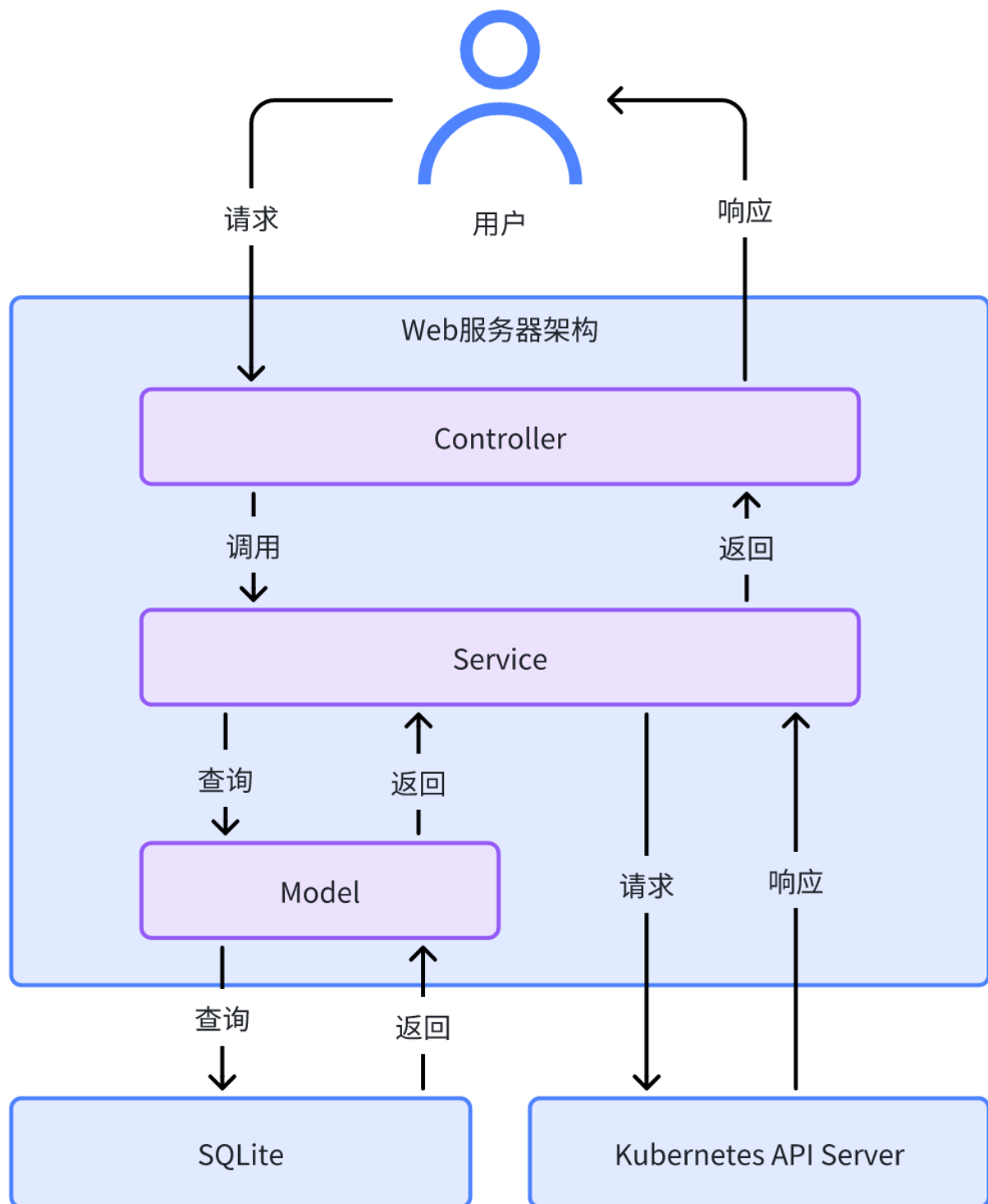
在自定义资源中填写docker镜像配置的请求，Controller会将请求下发给各个符合条件的节点的HTTP服务器，受到请求的节点会修改docker镜像。

3.8 Operator功能：节点代理配置

在自定义资源中填写代理配置的请求，Controller会将请求下发给各个符合条件的节点的HTTP服务器，受到请求的节点会修改代理信息。

3.9 额外功能：web服务器（容器化之前）

使用web界面调用k8s api server 提供的RESTful 接口，完成上述所有功能，提供更良好的交互体验。需要查看K8S RESTful 接口文档、用户认证、用户授权相关内容。



3.10 额外功能实现：RESTful 接口调用

通过查看k8s官方API文档，K8S官方内置资源资源的API一般为：

```
/api/v1/namespaces/{namespace}/{resource}/{name}
```

K8s第三方资源（如operator）的API一般为：

```
/apis/{domain}/{version}/namespaces/{namespace}/{resource}/{name}
```

通过以上API在拥有权限的情况下就可以完成对k8s内部资源的操作。

3.11 额外功能实现：k8s用户认证

K8s内部拥有两种类型的用户，一种是普通用户，另一种是服务账号。普通用户被假定为由外部独立服务管理。管理员分发私钥，用户存储（如 Keystone 或 Google 帐户），甚至包含用户名和密码列表的文件。无法通过 API 调用的方式向集群中添加普通用户。

相对的，service account 是由 Kubernetes API 管理的帐户。它们都绑定到了特定的 namespace，并由 API server 自动创建，或者通过 API 调用手动创建。Service account 关联了一套凭证，存储在 Secret，这些凭证同时被挂载到 pod 中，从而允许 pod 与 kubernetes API 之间的调用。

API 请求被绑定到普通用户或 service account 上，或者作为匿名请求对待。这意味着集群内部或外部的每个进程，无论在工作站上输入 kubectl 的人类用户到节点上的 kubelet，到控制平面的成员，都必须在向 API Server 发出请求时进行身份验证，或者被视为匿名用户。

3.12 额外功能实现：k8s用户授权

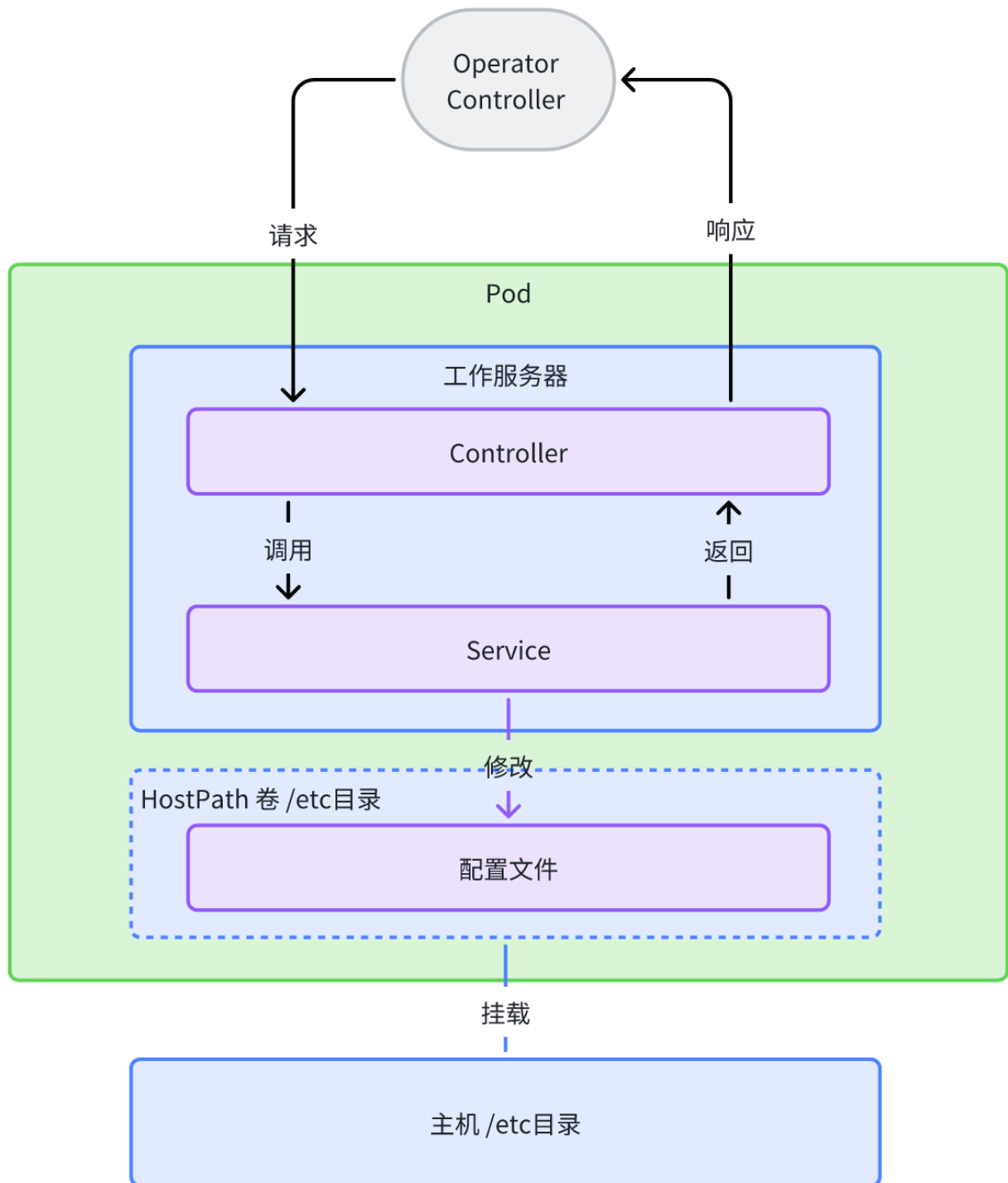
K8s 提供了多种用户授权方式最常见由RBAC与ABAC。通过配置RBAC将权限授予相应的用户后，即可执行相应的操作完成资源管理（详情见文档：《K8S web权限管理与用户认证》）

3.13 组件容器化

容器化的组件无需额外配置，简化安装过程，只需拉取镜像即可直接部署使用。

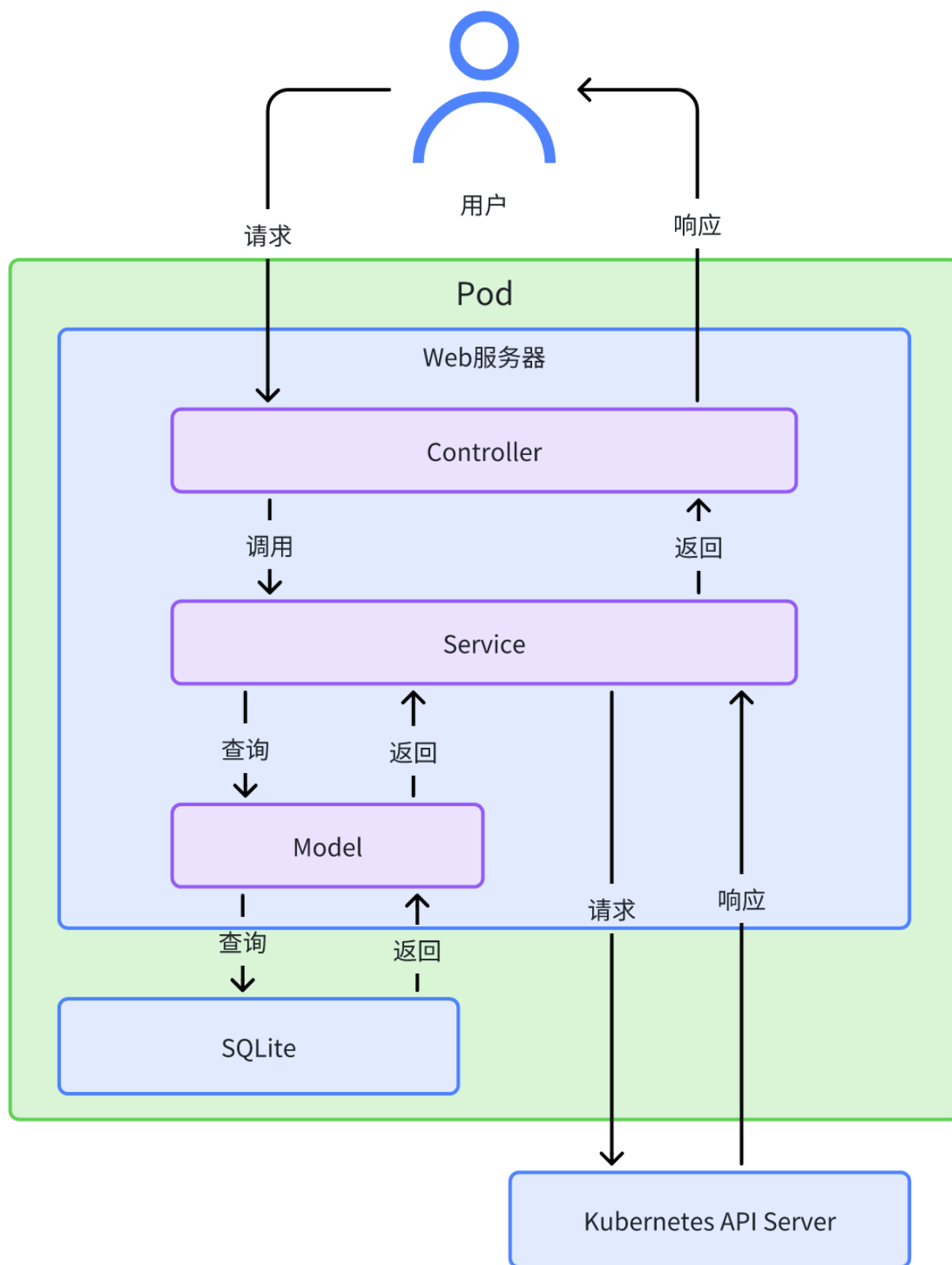
3.13.1 工作服务器容器化

系统中的工作服务器以容器的形式运行于K8S集群的Pod中，使用nodeName字段将工作服务器的Pod限制到各个主机（防止其调度到其他主机），依靠HostPath机制将主机etc目录挂载到pod内部的镜像中，以此达成对于容器OS配置的修改。



3.13.2 Web服务器容器化

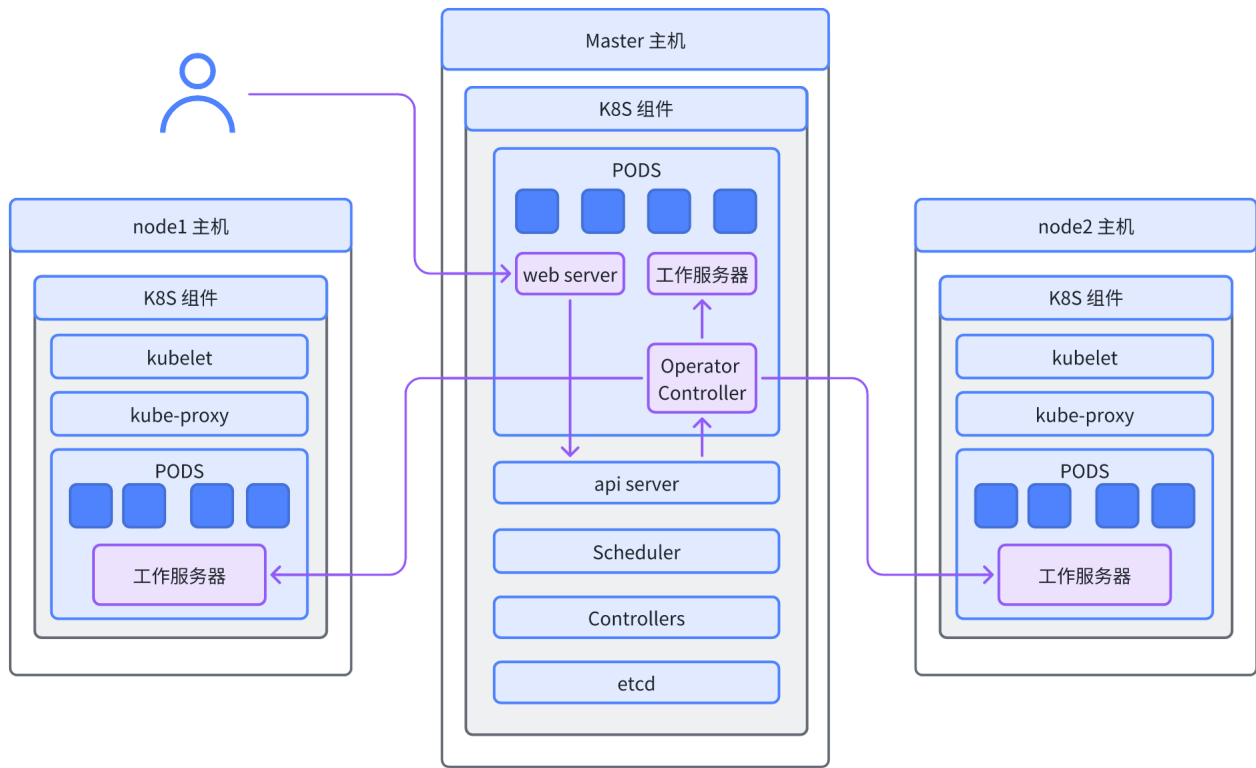
系统中的Web服务器以容器的形式运行于K8S集群的Pod中，使用Service资源将Pod内部端口映射到master节点，使得外部可以访问。



3.13.3 Controller 容器化

Controller 也可以放入Pod中运行，但是需要为Pod分配特定的服务账号，并为此服务账号授予Controller需要的所有权限。

容器化之后的架构图如下所示：



3.14 增强Operator控制器

增强Operator控制器功能，使其在资源创建阶段能够自动创建依赖，在资源更新阶段能管理依赖，在资源删除阶段能够自动删除所需依赖，不再需要用户手动管理。

4. 开发计划

4.1 项目范围

- 设计并实现自定义资源定义（CRD）。
- 开发k8s operator控制器。
- 实现配置下发机制。
- 确保配置下发过程的安全性。
- web server 界面编写
- 编写文档和测试用例。

4.2 软件资源

- 编程语言：Go、python
- 开发工具：Git、Docker、Goland、Pycharm
- 集群管理工具：kubeadm、kubectl

4.3 项目时间表

- 里程碑1：完成需求分析和项目规划（1周）
- 里程碑2：完成技术调研和选型（1周）
- 里程碑3：完成k8s集群部署（1周）
- 里程碑4：完成CRD定义和控制器框架搭建（2周）
- 里程碑5：实现配置下发机制和安全性设计（2周）
- 里程碑6：完成web界面编写（1周）
- 里程碑7：完成核心组件容器化（1周）
- 里程碑8：完成测试和文档编写（2周）
- 里程碑9：项目提交和准备答辩（1周）

4.4 项目交付物

- 源代码：包含CRD定义、operator控制器和配置下发逻辑。
- 用户手册：详细说明如何使用配置下发功能。
- 操作文档：提供部署和使用operator的详细步骤。

5. 比赛过程中的重要进展

进展1: 部署k8s集群

- 完成k8s集群部署，为后续过程做好准备。

进展2: 定义CRD

- 创建自定义资源定义（CRD），用于描述需要下发的配置信息。

进展3: 开发operator控制器

- 实现operator的控制器逻辑，包括监听CRD的创建、更新和删除事件。
- 控制器负责解析CRD中的配置信息，并将其下发到指定的worker节点。

进展4: 实现配置下发机制

- 开发配置下发逻辑，确保配置能够安全、高效地下发到各个节点。
- 支持配置文件的差异化，根据不同节点需求下发不同的配置。

进展5: 安全性与权限控制

- 配置服务账号，确保系统内部组件可以访问k8s集群。
- 实现RBAC（基于角色的访问控制），确保只有授权用户可以下发配置。
- 配置Network Policy，确保外界不可访问组件Pod。

进展6: 开发web界面

- 实现web界面，以图形化方式下发配置。

进展7: 部署与文档编写

- 编写部署脚本，方便在k8s集群中部署operator。
- 编写用户手册和操作文档，指导用户如何使用operator下发配置。

进展8: 核心组件容器化

- 使用Docker工具以及Dockerfile脚本将工作服务器、web服务器容器化。
- 核心组件（controller、web 服务器、工作服务器）现在可以运行在K8S集群的Pod中，进一步简化配置过程。

进展9: 增强operator控制器功能

- 现在Operator使用的各种资源由Operator本身负责管理，无需手动管理
- Operator负责创建各个工作服务器Pod
- Operator负责创建web服务器本身以及其所需的各种资源

6. 系统测试情况

6.1 测试环境

- 硬件环境：
 - Master节点：CPU 4核心，内存4GB，硬盘20GB
 - Worker节点：CPU 4核心，内存4GB，硬盘20GB
- 软件环境：
 - kubernetes版本：v1.26
 - 操作系统：RockyLinux 9.4
 - 网络插件：Calico
 - 编程语言：go 1.22

- **测试工具：**
 - kubectl：用于与k8s集群交互
 - Git：用户代码版本控制
 - 浏览器：用于测试web界面

6.2 测试用例

6.2.1 功能测试

- **用例1：**创建CRD并验证其在集群中的状态。
- **用例2：**下发单个配置文件到指定worker节点。
- **用例3：**下发多个配置文件到不同worker节点。
- **用例4：**更新已下发的配置文件。
- **用例5：**删除配置文件并验证节点上的配置是否被清除。
- **用例6：**创建Operator对象，并验证Controller是否创建依赖资源
- **用例7：**删除Operator对象，并验证Controller是否删除依赖资源

6.2.2 安全性测试

- **用例8：**未经授权的用户不应下发配置。

6.3 测试结果：

6.3.1 功能测试结果

- **用例1：**成功创建CRD，状态正常。
- **用例2：**配置文件成功下发到指定节点。
- **用例3：**多个配置文件成功下发到不同节点。
- **用例4：**已下发的配置文件成功更新。
- **用例5：**配置文件删除后，节点上的配置也被清除。
- **用例6：**Operator对象创建后，依赖资源也被创建
- **用例7：**Operator对象删除后，依赖资源也被删除

6.3.2 安全性测试结果

- **用例8：**未经授权的用户无法下发配置，系统安全性得到验证。

7. 遇到的主要问题和解决方法

7.1 k8s部署过程中遇到的问题

node加入master之后get nodes 失败

```
[root@node2 ~]# kubectl get nodes
E0317 20:37:09.770437 1887 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial tcp [::1]:8080: connect: connection refused
E0317 20:37:09.771613 1887 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial tcp [::1]:8080: connect: connection refused
E0317 20:37:09.771987 1887 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial tcp [::1]:8080: connect: connection refused
E0317 20:37:09.773738 1887 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial tcp [::1]:8080: connect: connection refused
E0317 20:37:09.775429 1887 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial tcp [::1]:8080: connect: connection refused
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

问题原因： 没有将admin配置文件导入到当前用户下

解决方案： 运行如下命令，将配置文件导入

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

pods sandbox create failed

问题原因： 网络问题

解决方案： 更换镜像源或使用科学上网

calico镜像pull缓慢或出错

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	calico-kube-controllers-57b57c56f-zssn6	1/1	Running	0	55m
kube-system	calico-node-728kq	1/1	Running	0	55m
kube-system	calico-node-n7v9v	0/1	Init:2/3	0	5m55s
kube-system	coredns-5bbd96d687-f8cg8	1/1	Running	0	32h
kube-system	coredns-5bbd96d687-tgg2s	1/1	Running	0	32h
kube-system	etcd-master	1/1	Running	4 (56m ago)	32h
kube-system	kube-apiserver-master	1/1	Running	4 (56m ago)	32h
kube-system	kube-controller-manager-master	1/1	Running	7 (56m ago)	32h
kube-system	kube-proxy-8jgn6	1/1	Running	2 (56m ago)	32h
kube-system	kube-proxy-cz8rp	1/1	Running	0	9m17s
kube-system	kube-scheduler-master	1/1	Running	7 (56m ago)	32h


```
Normal Sched 6m40s default-scheduler Successfully assigned kube-system/calico-node-n7v9v to node2
Normal Pull 6m39s kubelet Container image "docker.io/calico/cni:v3.25.0" already present on machine
Normal Created 6m39s kubelet Created container upgrade-ipam
Normal Started 6m39s kubelet Started container upgrade-ipam
Normal Pulled 6m39s kubelet Container image "docker.io/calico/cni:v3.25.0" already present on machine
Normal Created 6m39s kubelet Created container install-cni
Normal Started 6m39s kubelet Started container install-cni
Normal Pulling 6m39s kubelet Pulling image "docker.io/calico/node:v3.25.0"

Warning Unhealthy 11m kubelet Readiness probe failed: calico/node is not ready: BIRD is not ready: Error querying BIRD: unable to connect to BIRDv4 socket: dial unix /var/run/bird/birdctl: connect: no such file or directory
Warning Unhealthy 11m (x2 over 11m) kubelet Readiness probe failed: calico/node is not ready: BIRD is not ready: Error querying BIRD: unable to connect to BIRDv4 socket: dial unix /var/run/calico/birdctl: connect: connection refused
Warning Unhealthy 11m kubelet Readiness probe failed: 2023-03-17 15:00:07.231 [INFO][186] confd/health.go 180: Number of node(s) with BGP peering established = 0
calico/node is not ready: BIRD is not ready: BGP not established with 192.168.111.200,192.168.111.201
```

问题原因： 网络原因

解决方案： 更换镜像源

将containerd 源从 registry.cn-hangzhou.aliyuncs.com/google_containers/pause 换成 registry.aliyuncs.com/k8sxio/pause ,

7.2 开发过程中遇到的问题

k8s RESTful接口官方文档不清晰

解决方案： 结合官方文档进行测试整理，总结出文档《K8S RESTful API 使用》与《K8S Restful API返回状态码》。

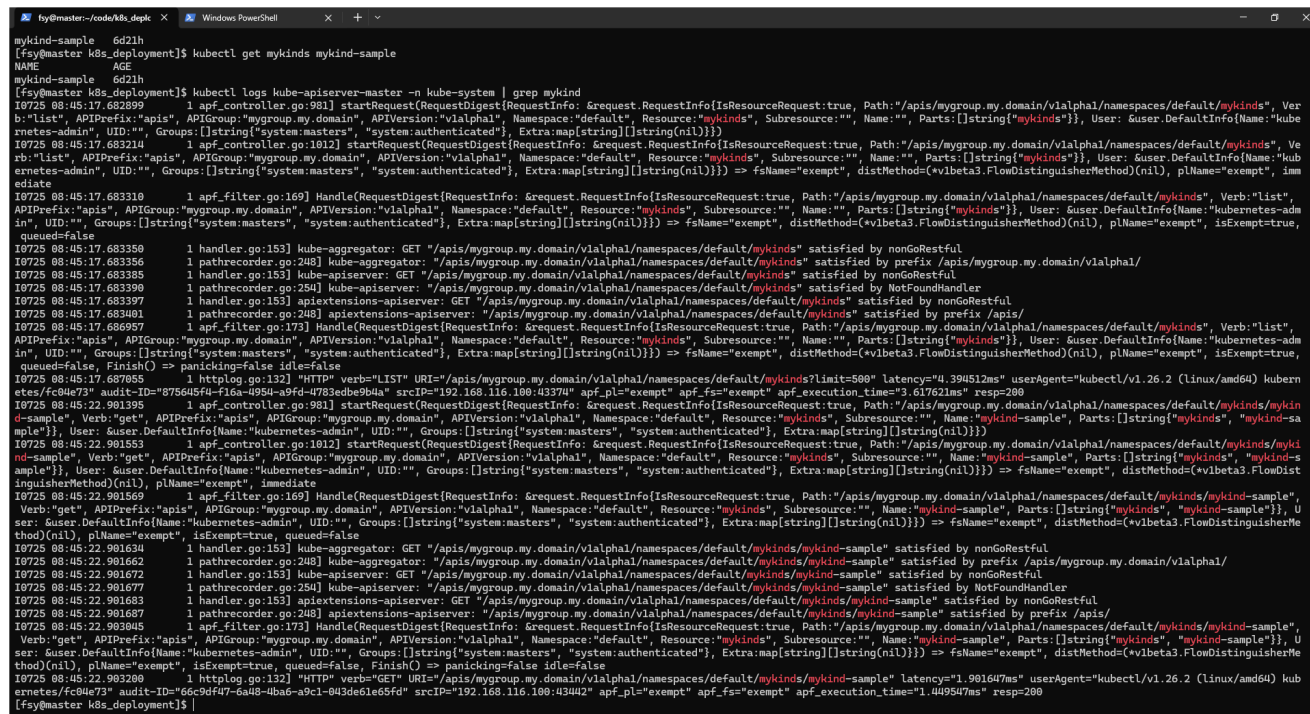
k8s operator 第三方接口RESTful接口地址不清晰

解决方案：

k8s中所有的操作都是调用api server完成，首先使用kubectl管理operator，查看k8s api server 日志文件，即可找到operator接口的地址信息。

调整api server的日志输出级别

在 api server的启动选项中添加 `-v 7` 参数即可看见客户端请求的完整信息，可以提取出 RESTful接口地址。



```

mykind-sample 6d21h
[fsy@master k8s_deployment]$ kubectl get mykinds mykind-sample
NAME
mykind-sample 6d21h
[fsy@master k8s_deployment]$ kubectl logs kube-apiserver-master -n kube-system | grep mykind
19725 08:45:17.682899 1 apf.controller.go:981 startRequest(RequestDigest[RequestInfo: &request.RequestInfo{IsResourceRequest:true, Path: "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds", Ver
b:"list", APIPrefix:"apis", APIGroup:"mygroup.my.domain", APIVersion:"v1alpha1", Namespace:"default", Resource:"mykinds", Subresource:"", Name:"", Parts:[[]string{"mykinds"}], User: &user.DefaultInfo{Name:"kub
ernetes-admin", UID:"", Groups:[[]string{"system:masters", "system:authenticated"}, Extra:map[string][]string(nil)}]) => fsName="exempt", distMethod=(v1beta3.FlowDistinguisherMethod)(nil), pName="exempt", im
mediate
19725 08:45:17.683310 1 apf.filter.go:169 Handle(RequestDigest[RequestInfo: &request.RequestInfo{IsResourceRequest:true, Path: "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds", Verb:"list",
APIPrefix:"apis", APIGroup:"mygroup.my.domain", APIVersion:"v1alpha1", Namespace:"default", Resource:"mykinds", Subresource:"", Name:"", Parts:[[]string{"mykinds"}], User: &user.DefaultInfo{Name:"kubernetes-adm
in", UID:"", Groups:[[]string{"system:masters", "system:authenticated"}, Extra:map[string][]string(nil)}]) => fsName="exempt", distMethod=(v1beta3.FlowDistinguisherMethod)(nil), pName="exempt", isExempt=true,
queued=false
19725 08:45:17.683350 1 handler.go:153 kube-aggregator: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds" satisfied by nonGoRestful
19725 08:45:17.683356 1 pathrecorder.go:248 kube-aggregator: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds" satisfied by prefix /apis/mygroup.my.domain/v1alpha1/
19725 08:45:17.683385 1 handler.go:153 kube-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds" satisfied by nonGoRestful
19725 08:45:17.683390 1 pathrecorder.go:254 kube-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds" satisfied by NotFoundHandler
19725 08:45:17.683397 1 handler.go:153 apiextensions-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds" satisfied by nonGoRestful
19725 08:45:17.683401 1 pathrecorder.go:248 apiextensions-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds" satisfied by prefix /apis/
19725 08:45:17.686957 1 apf.filter.go:173 Handle(RequestDigest[RequestInfo: &request.RequestInfo{IsResourceRequest:true, Path: "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds", Verb:"list",
APIPrefix:"apis", APIGroup:"mygroup.my.domain", APIVersion:"v1alpha1", Namespace:"default", Resource:"mykinds", Subresource:"", Name:"", Parts:[[]string{"mykinds"}], User: &user.DefaultInfo{Name:"kubernetes-adm
in", UID:"", Groups:[[]string{"system:masters", "system:authenticated"}, Extra:map[string][]string(nil)}]) => fsName="exempt", distMethod=(v1beta3.FlowDistinguisherMethod)(nil), pName="exempt", isExempt=true,
queued=false, Finish() => panicking=false idle=false
19725 08:45:17.687055 1 httplog.go:132 "HTTP" verb="LIST" URI="/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds?limit=500" latency="4.394512ms" userAgent="kubectl/v1.26.2 (linux/amd64) kubern
etes/fc0b4e73" audit-ID="875645f4-f16a-495d-a9fd-4783edbe9b4a" srcIP="192.168.116.100:43374" apf.pl="exempt" apf.fs="exempt" apf.execution_time="3.617621ms" resp=200
19725 08:45:22.901395 1 apf.controller.go:981 startRequest(RequestDigest[RequestInfo: &request.RequestInfo{IsResourceRequest:true, Path: "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykin
d-sample", Verb:"get", APIPrefix:"apis", APIGroup:"mygroup.my.domain", APIVersion:"v1alpha1", Namespace:"default", Resource:"mykinds", Subresource:"", Name:"mykind-sample", Parts:[[]string{"mykinds", "mykind-sa
mple"}], User: &user.DefaultInfo{Name:"kubernetes-admin", UID:"", Groups:[[]string{"system:masters", "system:authenticated"}, Extra:map[string][]string(nil)}]) => fsName="exempt", distMethod=(v1beta3.FlowDist
inguisherMethod)(nil), pName="exempt", immediate
19725 08:45:22.901553 1 apf.controller.go:1012 startRequest(RequestDigest[RequestInfo: &request.RequestInfo{IsResourceRequest:true, Path: "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/myki
nd-sample", Verb:"get", APIPrefix:"apis", APIGroup:"mygroup.my.domain", APIVersion:"v1alpha1", Namespace:"default", Resource:"mykinds", Subresource:"", Name:"mykind-sample", Parts:[[]string{"mykinds", "mykind-s
ample"}], User: &user.DefaultInfo{Name:"kubernetes-admin", UID:"", Groups:[[]string{"system:masters", "system:authenticated"}, Extra:map[string][]string(nil)}]) => fsName="exempt", distMethod=(v1beta3.FlowDist
inguisherMethod)(nil), pName="exempt", immediate
19725 08:45:22.901569 1 apf.filter.go:169 Handle(RequestDigest[RequestInfo: &request.RequestInfo{IsResourceRequest:true, Path: "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample",
Verb:"get", APIPrefix:"apis", APIGroup:"mygroup.my.domain", APIVersion:"v1alpha1", Namespace:"default", Resource:"mykinds", Subresource:"", Name:"mykind-sample", Parts:[[]string{"mykinds", "mykind-sample"}], U
ser: &user.DefaultInfo{Name:"kubernetes-admin", UID:"", Groups:[[]string{"system:masters", "system:authenticated"}, Extra:map[string][]string(nil)}]) => fsName="exempt", distMethod=(v1beta3.FlowDistinguisherMe
thod)(nil), pName="exempt", isExempt=true, queued=false
19725 08:45:22.901634 1 handler.go:153 kube-aggregator: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample" satisfied by prefix /apis/mygroup.my.domain/v1alpha1/
19725 08:45:22.901662 1 pathrecorder.go:248 kube-aggregator: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample" satisfied by nonGoRestful
19725 08:45:22.901672 1 handler.go:153 kube-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample" satisfied by nonGoRestful
19725 08:45:22.901677 1 pathrecorder.go:254 kube-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample" satisfied by NotFoundHandler
19725 08:45:22.901683 1 handler.go:153 apiextensions-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample" satisfied by nonGoRestful
19725 08:45:22.901697 1 pathrecorder.go:248 apiextensions-apiserver: GET "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample" satisfied by prefix /apis/
19725 08:45:22.903045 1 apf.filter.go:173 Handle(RequestDigest[RequestInfo: &request.RequestInfo{IsResourceRequest:true, Path: "/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample",
Verb:"get", APIPrefix:"apis", APIGroup:"mygroup.my.domain", APIVersion:"v1alpha1", Namespace:"default", Resource:"mykinds", Subresource:"", Name:"mykind-sample", Parts:[[]string{"mykinds", "mykind-sample"}], U
ser: &user.DefaultInfo{Name:"kubernetes-admin", UID:"", Groups:[[]string{"system:masters", "system:authenticated"}, Extra:map[string][]string(nil)}]) => fsName="exempt", distMethod=(v1beta3.FlowDistinguisherMe
thod)(nil), pName="exempt", isExempt=true, queued=false, Finish() => panicking=false idle=false
19725 08:45:22.903260 1 httplog.go:132 "HTTP" verb="GET" URI="/apis/mygroup.my.domain/v1alpha1/namespaces/default/mykinds/mykind-sample" latency="1.901647ms" userAgent="kubectl/v1.26.2 (linux/amd64) kub
ernetes/fc0b4e73" audit-ID="66c0d4f7-6a08-4b46-a9c1-0430e61e65fd" srcIP="192.168.116.100:43404" apf.pl="exempt" apf.fs="exempt" apf.execution_time="1.449547ms" resp=200
[fsy@master k8s_deployment]$
  
```

Containerd 镜像导入

描述：

此次项目中部署的K8S集群使用Containerd作为容器化引擎。尝试从本地导入镜像时，K8S集群无法查找到本地导入到镜像。

解决方案：

containerd中存在命名空间机制，k8s不会搜索默认命名空间中的镜像，需要将镜像导入到k8s.io命名空间，具体示例如下：

```
sudo ctr -n k8s.io images import [image].tar
```

8. 分工和协作

- 题目分析：付盛原
- 架构设计：付盛原
- 系统开发：付盛原
- 系统测试：付盛原
- 文档撰写：付盛原
- 视频录制与剪辑：付盛原
- 技术支持：刘高翔、李航老师

9. 提交仓库目录和文件描述

9.1 /

项目根目录，存放有全部项目文档、演示视频、源代码、Dockerfile、Controller Pod定义、operator工具自动生成的代码。

9.2 /api

存放有operator CRD资源的go语言定义以及由operator工具自动生成的代码。

9.3 /cmd

程序入口，由operator工具自动生成

9.4 /config

operator配置文件，部分由operator工具生成。另一部分用于定义k8s资源实例，需要手动编写。

9.5 /hack

由operator工具自动生成

9.6 /internal

operator的核心代码，包含controller以及其他自行编写的工具类代码。

9.7 /web

web server前后端源码

9.8 /worker_server

用于接收controller请求的服务器源码

9.9 /项目文档

与项目相关的全部文档

9.10 /演示视频

项目功能演示视频

9.11 /shell

配置系统的相关脚本

10. 比赛收获

10.1 技术能力提升

- **深入学习k8s技术**：通过此次比赛，团队成员对Kubernetes（k8s）的核心概念和技术细节有了更深入的理解，特别是在CRD和operator的开发方面。
- **掌握Operator模式**：学会了如何使用k8s operator来扩展集群功能，这对于未来在容器化环境中进行自定义资源的管理非常有帮助。

10.2 行业事业拓宽

- **了解行业趋势：**通过比赛，团队对容器化和云原生技术的前沿趋势有了更清晰的认识。

10.3 总结

我们深入学习了k8s技术，掌握了Operator模式，增强了在开发过程中对安全性的重视，并提高了信息交流的效率，解决了协作中的冲突和问题。在比赛过程中，我们学会了如何根据比赛规则和时间线制定有效的参赛策略，如何在压力下保持冷静，高效工作，对自己的职业发展路径有了更明确的规划，并在团队合作中学会了相互尊重和信任，增强了团队精神。