

基于 k8s operator 设计实现通过 k8s 集群下发操作系统配置

西安电子科技大学 XenoWYC121 队

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。

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

3. 项目技术介绍

3.1 K8S 简介

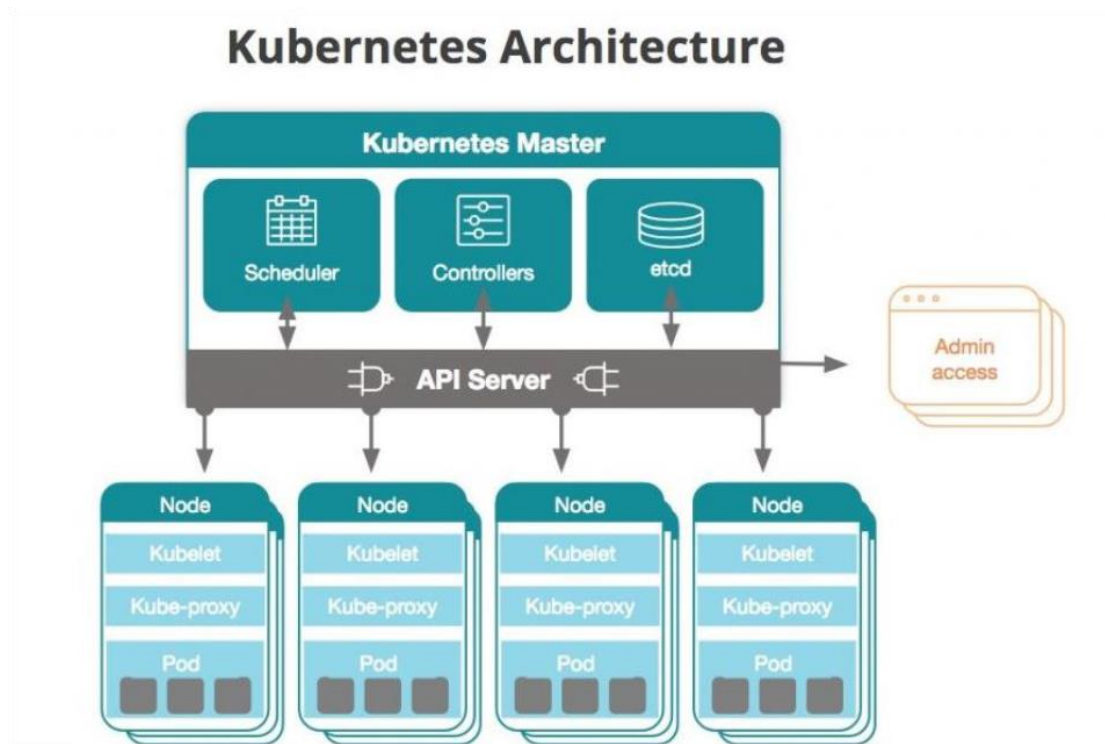
Kubernetes（简称 k8s）是 Google 在 2014 年 6 月开源的一个容器集群管理系统，使用 Go 语言开发，用于管理云平台中多个主机上的容器化的应用，Kubernetes 的目标是让部署容器化的应用简单并且高效，Kubernetes 提供了资源调度、部署管理、服务发现、扩容缩容、监控、维护等一整套功能，努力成为跨主机集群的自动部署、扩展以及运行应用程序容器的平台。它支持一系列容器工具，包括 Docker 等。

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。



3.2 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 (CICD)**：加速代码从提交到部署上线的过程。
 - CI: Continuous Integration, CD: Continuous Deployment

3.3 Docker 与 K8S 比较

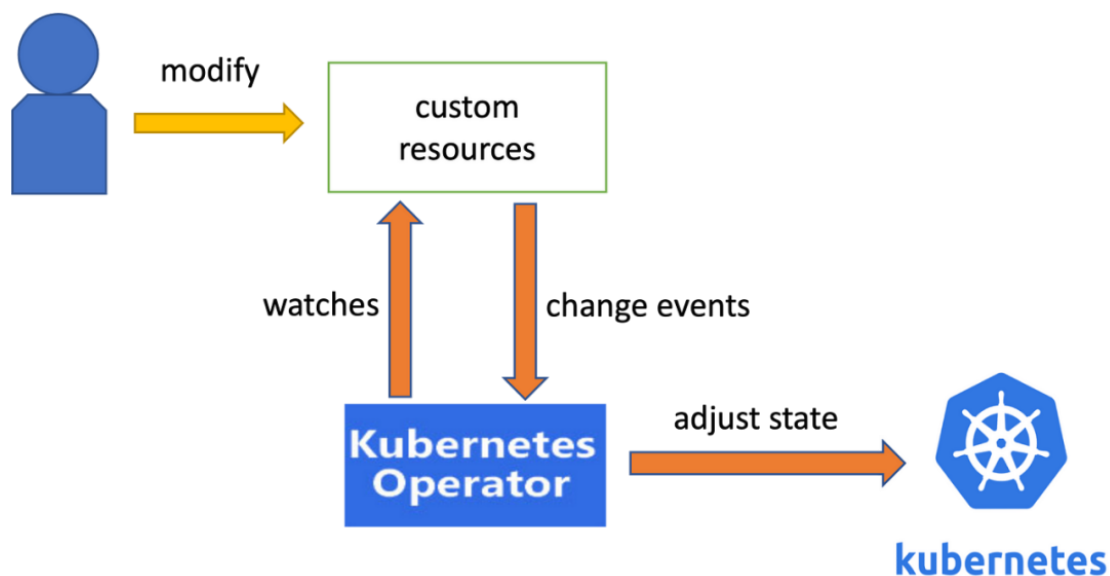
特性	Docker	Kubernetes
目的	用于打包和运行应用程序的容器化平台	用于管理和扩展容器化应用程序的容器编排平台
重点	创建、打包和运行单个容器	跨集群管理和编排大量容器
架构设计	具有简单 CLI 客户端-服务器架构	具有主节点和工作节点的复杂架构

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

3.4 K8S operator

3.4.1 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。

3.4.2 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。

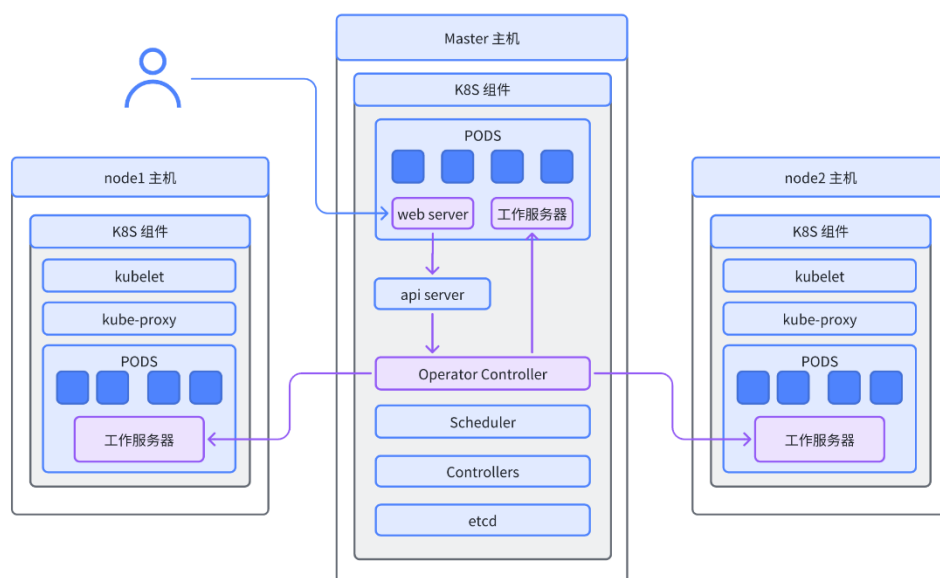
3.4.3 K8S operator 的优势

- 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。

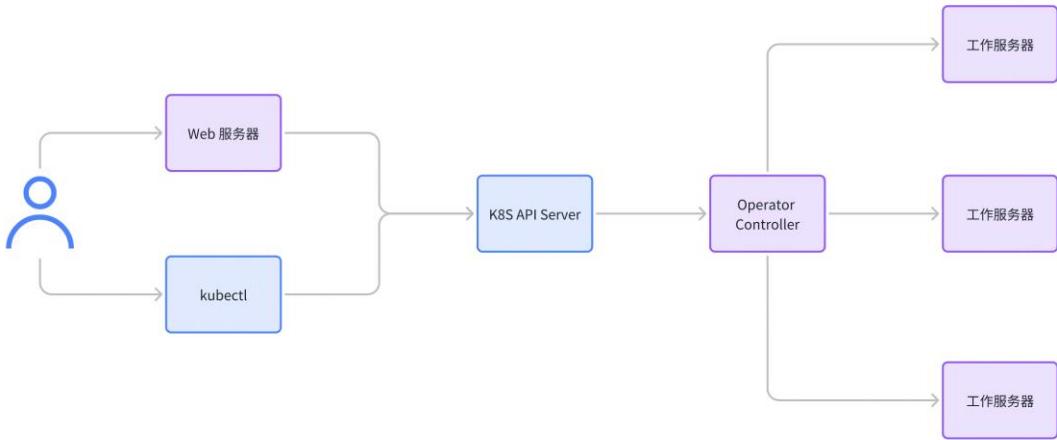
4. 项目设计

4.1 节点间通信方式

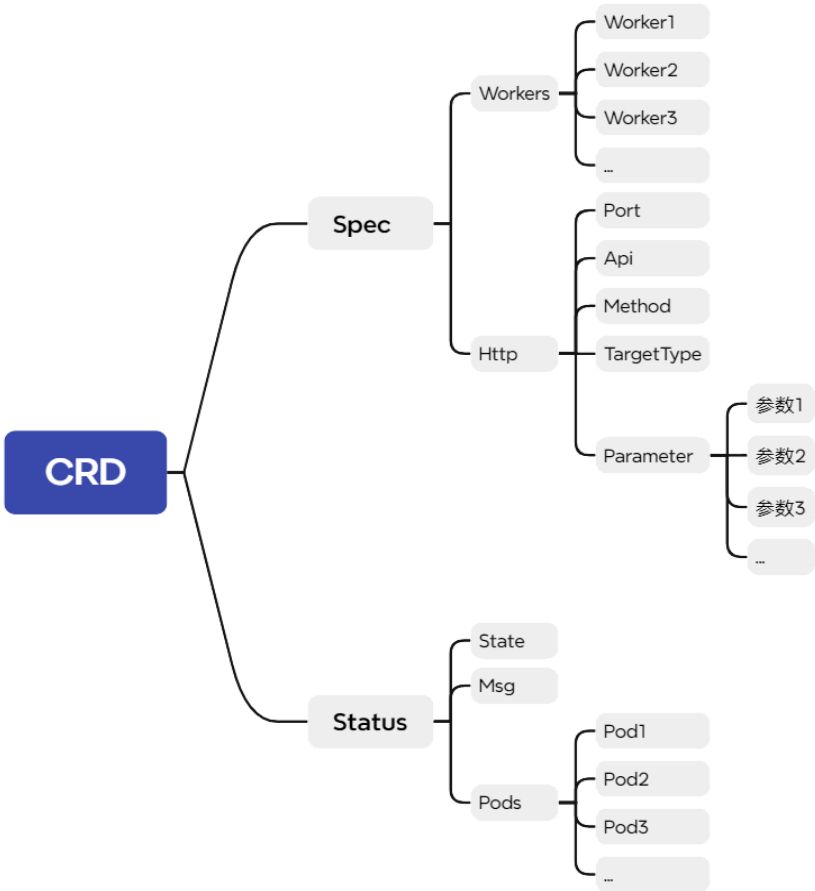
Master 与 worker 节点之间使用 HTTP 协议通信，由 K8S operator controller 作为客户端发送请求，worker 节点运行 HTTP 服务器负责接受请求、检查请求合法性、完成操作以及响应请求。工作服务器和 web 服务器组件以容器的形式运行于 Pod 中，简化了容器 OS 的部署过程，开箱即用。



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。



4.2 自定义资源设计



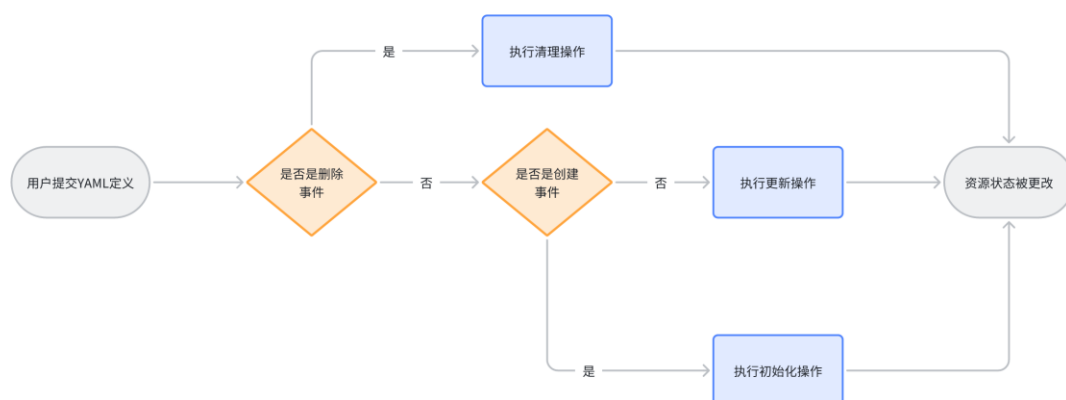
自定义资源需要以下字段：

- 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。

4.3 Operator Controller 设计（容器化之前）

4.3.1 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：

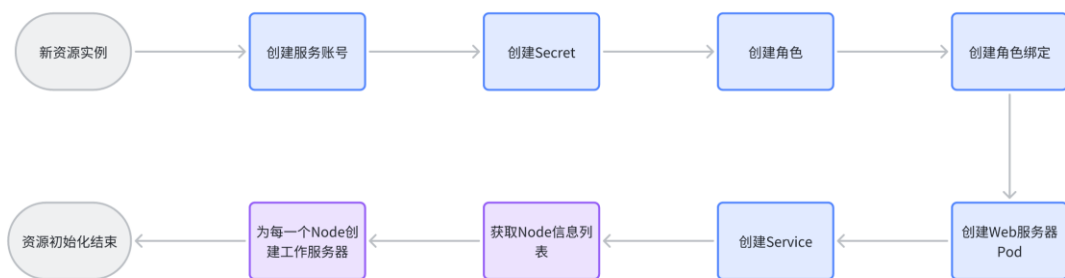


首先 Controller 将判断是否是删除事件触发了 Controller 执行，通过在 Controller 内部读取上下文的删除关键字即可判断。如果为删除事件，将会执行清理操作。

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。

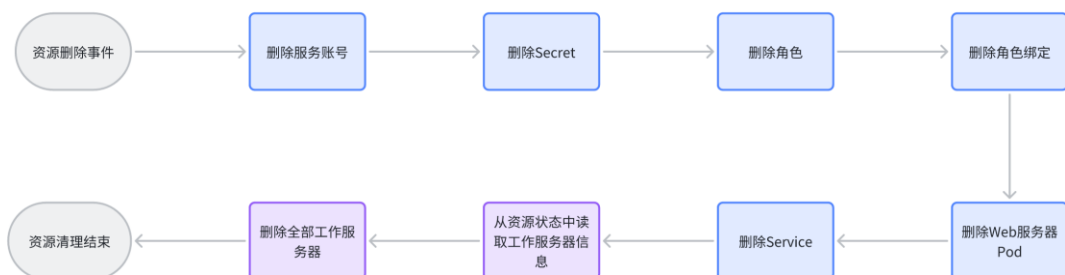
4.3.1.1 初始化操作



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。

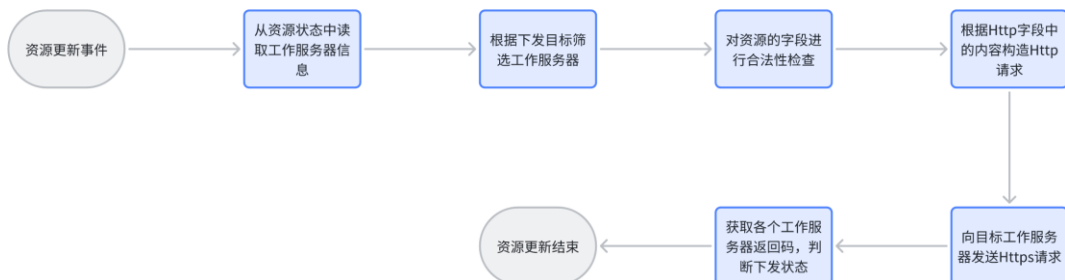
在初始化阶段的主要目标就是创建全部组件以及相关依赖项，蓝色方框代表了创建 Web 服务器以及相关依赖的操作，紫色方框代表了创建各个 Node 上的工作服务器的相关操作。

4.3.1.2 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。

4.3.1.3 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。

4.3.2 命令注入防止

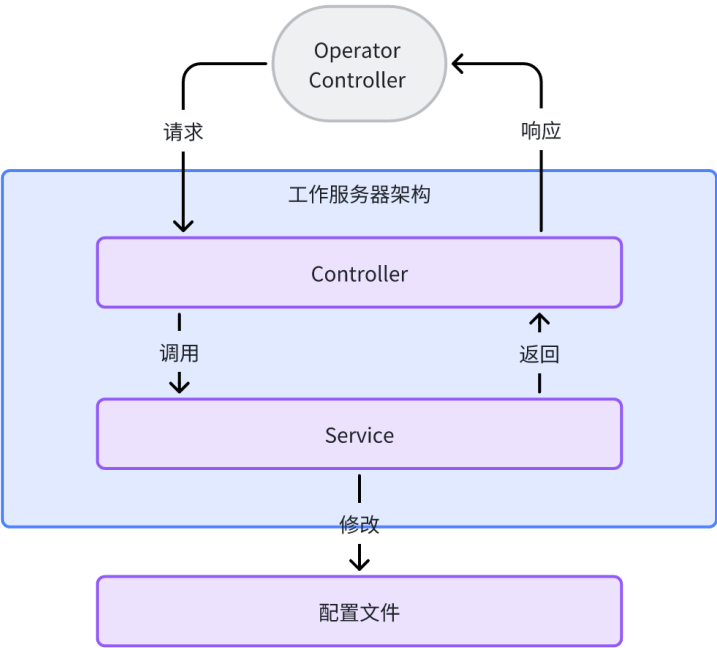
在 Operator Controller 和 HTTP 服务器端分别实现了对请求与参数的正则匹配，匹配的关键字包括常见 Linux 注入命令与特殊符号。通过对用户输入的预先处理，防止命令注入。

4.3.3 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。

4.4 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。



4.5 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。

4.6 Operator 功能：节点 docker 镜像配置

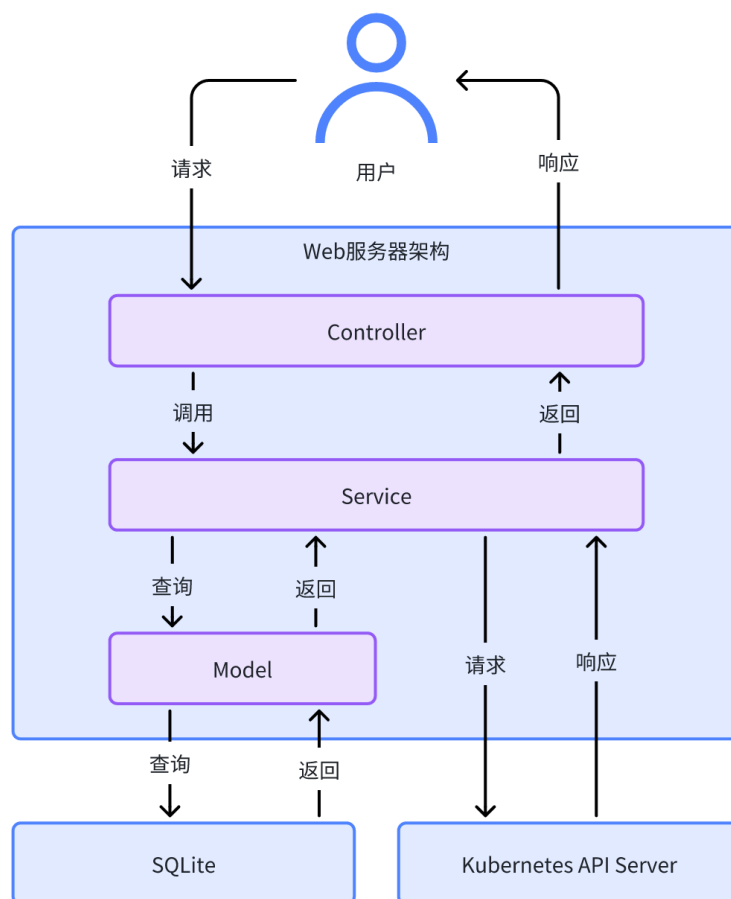
在自定义资源中填写 docker 镜像配置的请求，Controller 会将请求下发给各个符合条件的节点的工作服务器，受到请求的节点会修改 docker 镜像。

4.7 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工作服务器，受到请求的节点会修改代理信息。

4.8 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。



4.9 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。

4.10 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。

4.11 额外功能实现：k8s 用户授权

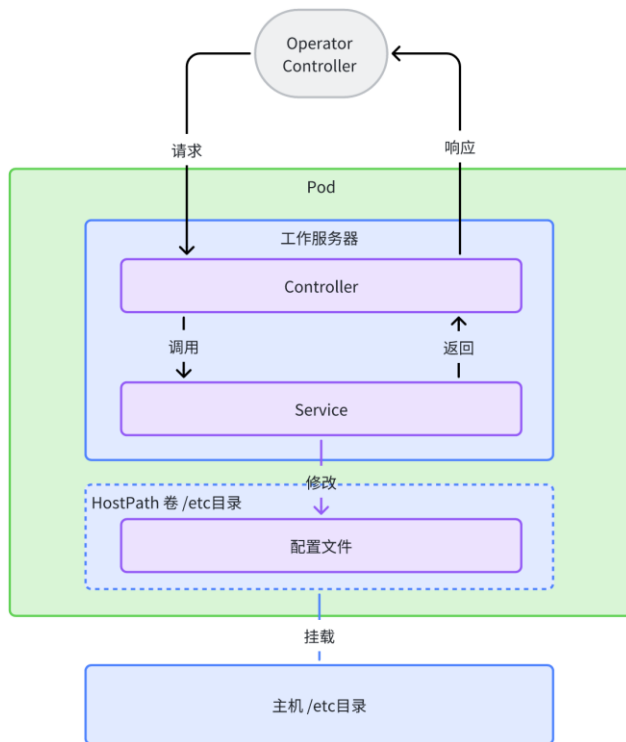
K8s 提供了多种用户授权方式最常见由 RBAC 与 ABAC。通过配置 RBAC 将权限授予相应的用户后，即可执行相应的操作完成资源管理（详情见文档：《K8S web 权限管理与用户认证》）

4.12 易用性提升：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。

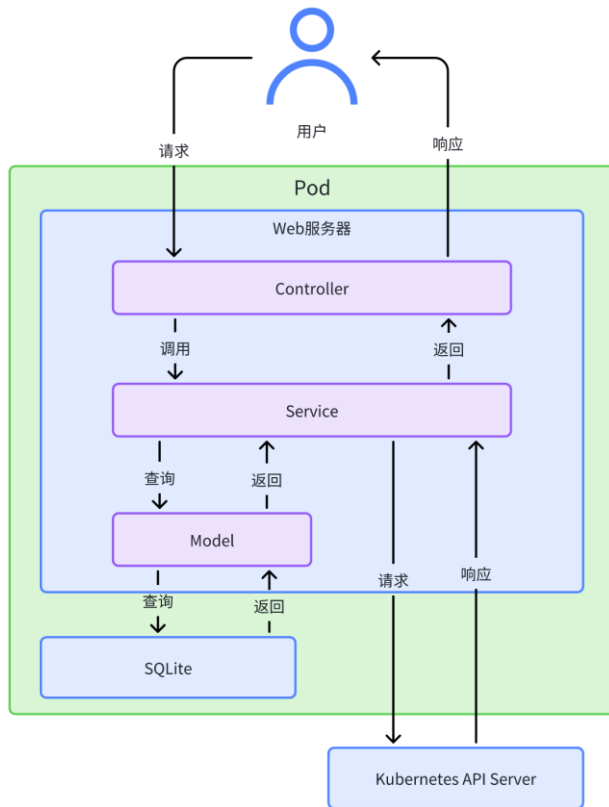
4.12.1 工作服务器容器化

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。



4.12.2 Web 服务器容器化

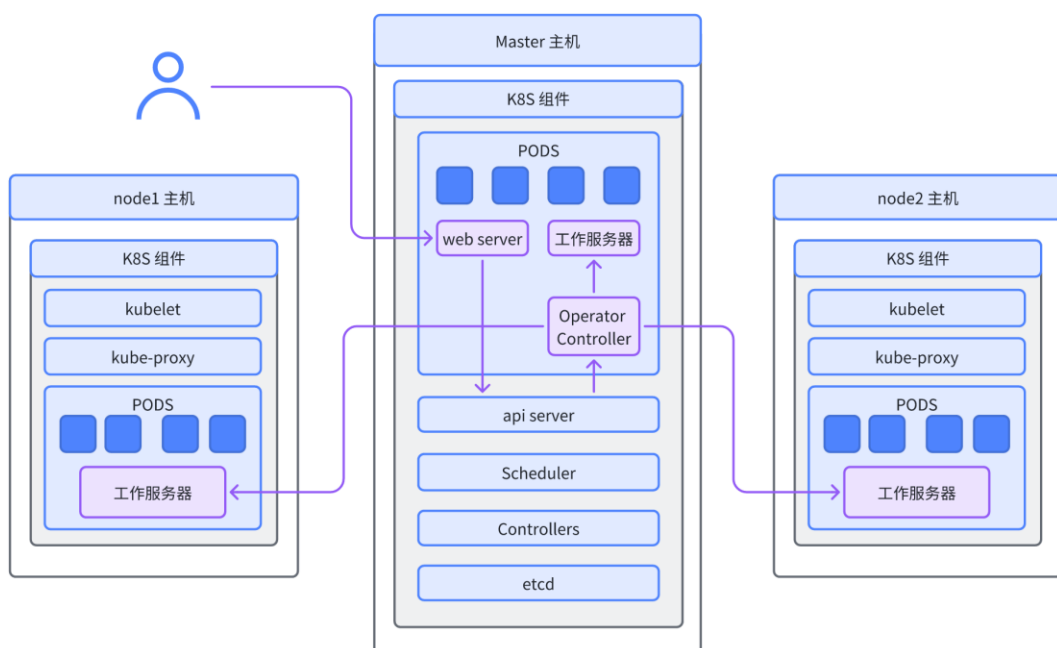
系统中的 Web 服务器以容器的形式运行于 K8S 集群的 Pod 中，使用 Service 资源将 Pod 内部端口映射到 master 节点，使得外部可以访问。



4.12.3 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：



4.13 易用性提升：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。

4.14 网络安全：

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）

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。