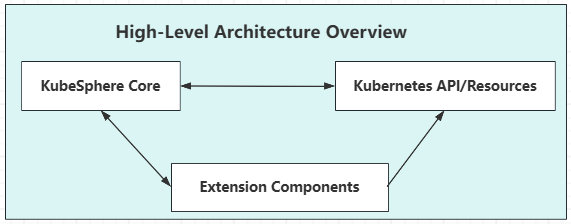
**KubeSphere**

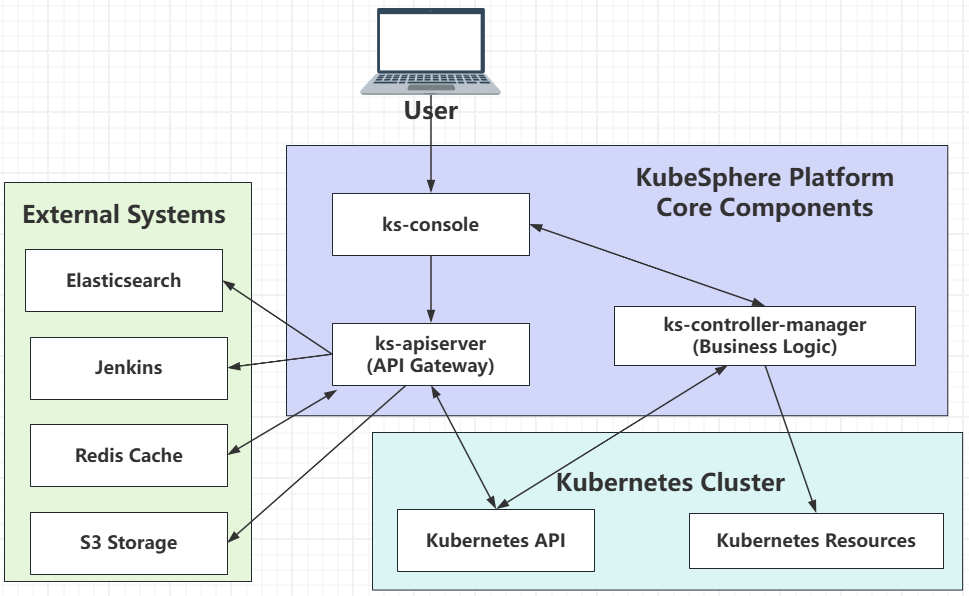
KubeSphere是一个基于 Kubernetes 构建的用于云原生应用管理的分布式作系统。KubeSphere 提供了一个全面的平台，该平台具有用户友好的界面、多租户支持以及专为跨多云、数据中心和边缘环境管理应用程序而设计的广泛功能。

# 一、KubeSphere 是什么？

KubeSphere 是一个用于云原生应用管理的分布式操作系统，使用 Kubernetes 作为其内核。它提供即插即用（plug-and-play）架构，允许第三方应用程序无缝集成到其生态系统中。该平台采用多租户设计，具有全栈自动化 IT 运营和简化的 DevOps 工作流程，所有这些都可以通过直观的 Web 界面访问。适用于 Kubernetes 多云、数据中心和边缘管理的容器平台。



## 1.1、技术架构图



## 1.2、平台功能

**可扩展架构**

旨在提供灵活性，支持基于插件的扩展和无缝集成。轻松定制和扩展功能，以满足不断变化的需求， 了解更多.

**🕸 部署 Kubernetes 集群**

支持在任何基础设施上部署 Kubernetes，支持在线安装和离线安装，了解更多。

**🔗 Kubernetes 多集群管理**

提供集中控制平台来管理多个 Kubernetes 集群，支持将应用程序发布到跨不同云供应商的多个 k8s 集群上。

**🤖 Kubernetes DevOps**

提供基于 GitOps 的 CD 方案，底层支持 Argo CD，可实时统计 CD 状态。结合主流 CI 引擎 Jenkins，让 DevOps 更加易用。了解更多。

**🔎 云原生可观测性**

支持多维度监控、事件和审计日志；内置多租户日志查询和收集，告警和通知，了解更多。

**🌐 基于 Istio 的微服务治理**

为分布式微服务应用程序提供细粒度的流量管理、可观测性和服务跟踪，支持可视化的流量拓扑，了解更多。

**💻 应用商店**

为基于 Helm 的应用程序提供应用商店，并在 Kubernetes 平台上提供应用程序生命周期管理功能，了解更多。

**💡 Kubernetes 边缘节点管理**

基于 KubeEdge 实现应用与工作负载在云端与边缘节点的统一分发与管理，解决在海量边、端设备上完成应用交付、运维、管控的需求，了解更多。

**🗃 支持多种存储和网络解决方案**

支持 GlusterFS、CephRBD、NFS、LocalPV ，并提供多个 CSI 插件对接公有云与企业级存储。

提供 Kubernetes 在裸机、边缘和虚拟化中的负载均衡器实现 OpenELB 。

提供网络策略和容器组 IP 池管理，支持 Calico、Flannel、Kube-OVN。

**🏢 多租户与统一鉴权认证**

具有基于角色的访问控制的逻辑隔离可确保跨多个租户的安全资源共享。支持细粒度的权限和配额管理，了解更多。

**🧠 GPU 工作负载调度与监控**

支持可视化创建 GPU 工作负载，支持 GPU 监控，同时还支持对 GPU 资源进行租户级配额管理。

# 核心组件

KubeSphere 由三个主要组件组成，它们协同工作以提供完整的容器平台：

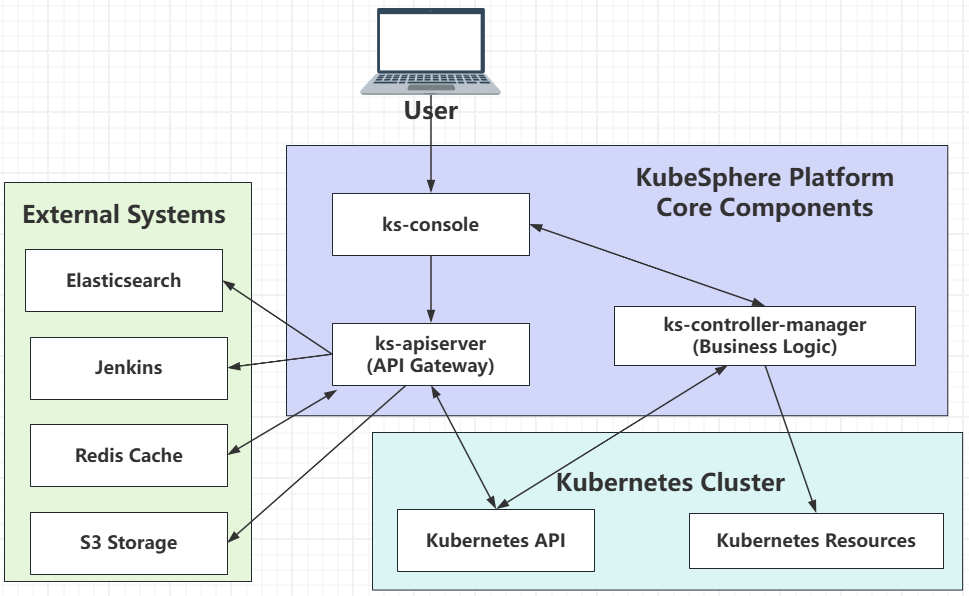
**ks-apiserver**：中央 API 网关，用于处理来自控制台和其他客户端的所有请求。它处理身份验证、授权以及与 Kubernetes API 和外部系统的通信。

**ks-controller-manager**：管理 KubeSphere 特定的资源并运行各种控制器，以协调所需状态与系统的实际状态。这些控制器处理企业空间、用户、应用程序和其他 KubeSphere 资源。

**ks-console**：基于 Web 的用户界面，提供与系统交互的可视化方式。它与 ks-apiserver 通信以执行所有操作。

每个组件都已容器化并单独部署。这些组件的 Dockerfile 可在以下位置找到：

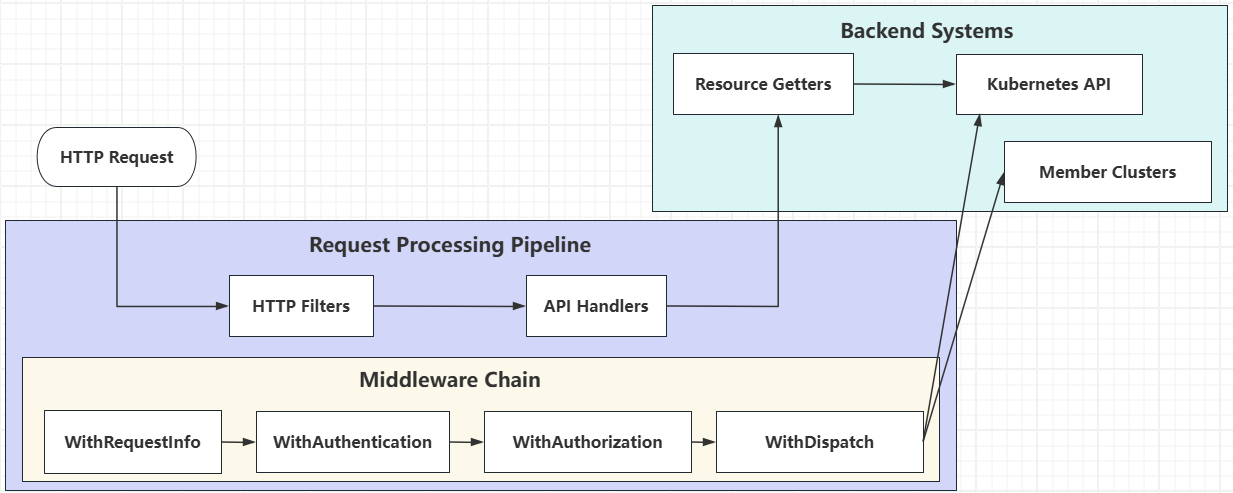
* build/ks-apiserver/Dockerfile
* build/ks-controller-manager/Dockerfile



## 2.1、ks-apiserver

ks-apiserver充当KubeSphere的中央网关。它处理所有传入请求，执行身份验证和授权，并将请求路由到相应的处理程序。ks-apiserver与 Kubernetes API 和各种外部系统进行通信。主要职责：

* 请求身份验证和授权
* 请求路由和处理
* API 扩展
* 与 Kubernetes 和外部系统通信
* 多集群请求委派

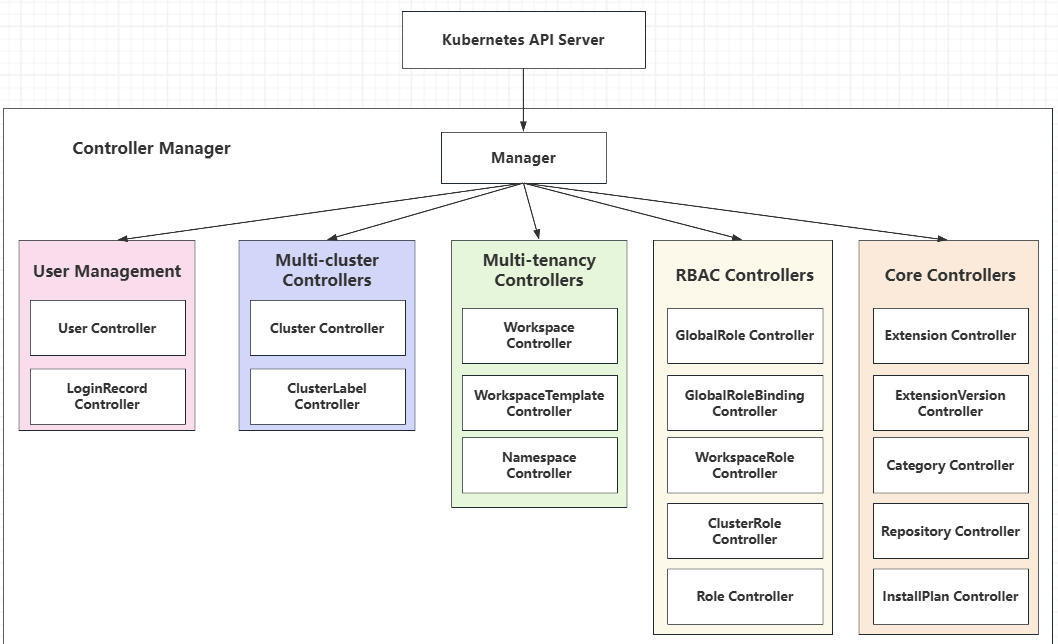


ks-apiserver初始化一个容器，该容器托管多个 Web 服务（API 组），每个服务处理特定的资源或功能。请求处理管道包括多个中间件组件：

1. WithRequestInfo：提取请求信息并将其添加到上下文中
2. WithAuthentication：通过各种方法对用户进行身份验证
3. WithAuthorization：检查用户是否有权执行请求的动作
4. WithDispatch：将请求路由到适当的处理程序
5. WithKubeAPIServer：在需要时代理 Kubernetes API
6. WithMultiCluster：将请求路由到多集群设置中的成员集群

## 2.2、ks-controller-manager

控制器管理器包含 KubeSphere 特定资源的业务逻辑。它运行多个控制器来监视资源并协调其状态。



控制器管理器注册并运行许多控制器，包括：

* 用于扩展管理的核心控制器
* 用于访问控制的 RBAC 控制器
* 用于工作区和命名空间管理的多租户控制器
* 用于集群联合的多集群控制器
* 用户管理控制器
* 应用程序生命周期控制器

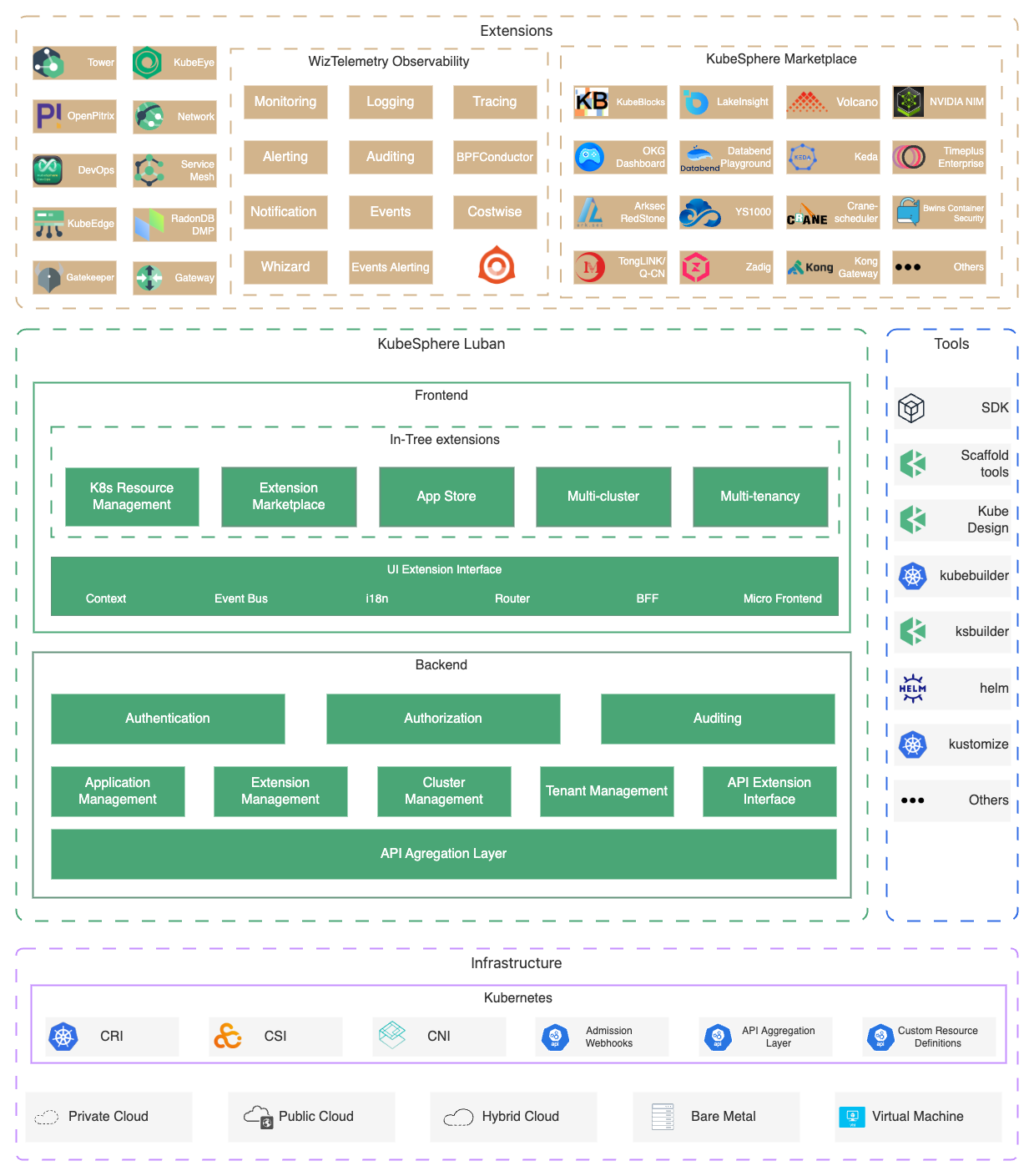
每个控制器都会监视特定资源并协调其状态，从而为各种 KubeSphere 功能实现业务逻辑。

## 2.3、ks-console

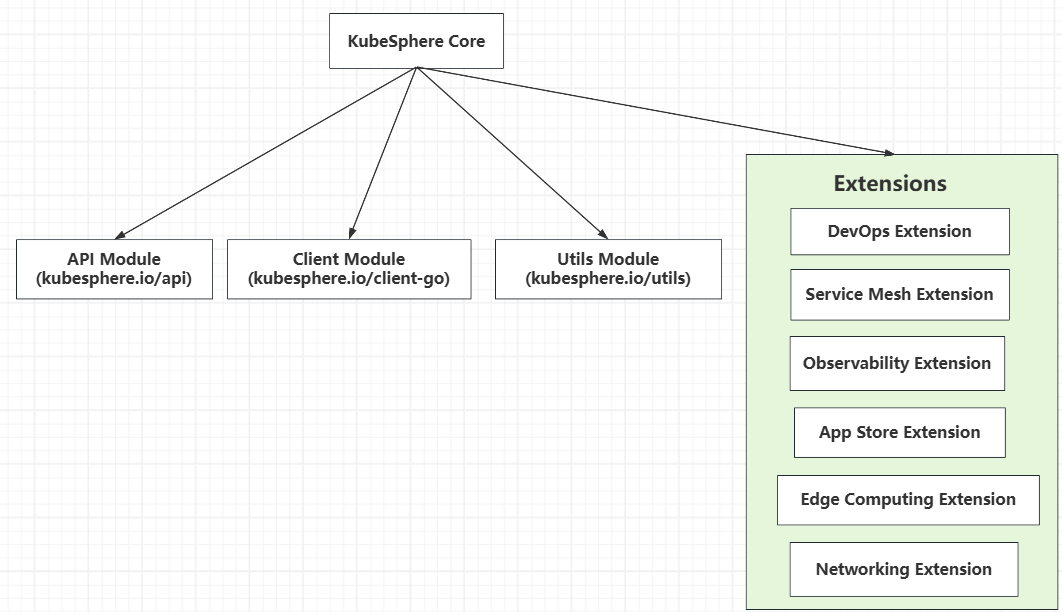
控制台为 KubeSphere 提供了基于 Web 的用户界面。它与 API 服务器通信以显示资源和执行作。

# 架构设计

KubeSphere 4.x，采用了微内核 + 扩展组件的架构（代号 LuBan）。其中内核部分（KubeSphere Core）仅包含系统运行的必备基础功能，将独立的功能模块拆分通过扩展组件（Extensions）的形式提供。用户可在系统运行时动态地管理扩展组件，借助扩展能力，KubeSphere 可以支持更多的应用场景，满足不同用户的需求。



## 3.1、模块结构图



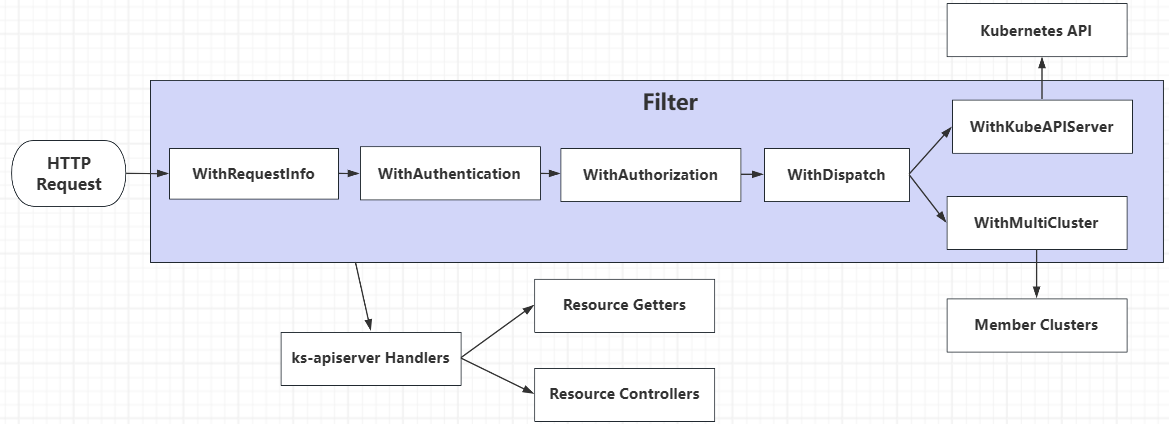
KubeSphere 的代码库分为几个模块：

|  |  |  |
| --- | --- | --- |
| **模块** | **描述** | **路径** |
| API | API 定义和自定义资源定义 | staging/src/kubesphere.io/api/ |
| client-go | 为 KubeSphere API 生成的客户端 | staging/src/kubesphere.io/client-go/ |
| utils | 常用实用程序和帮助程序函数 | staging/src/kubesphere.io/utils/ |
| cmd | 每个组件的入口点 | cmd/ |

# 请求处理管道

当请求进入 KubeSphere 时，它会经过 ks-apiserver 中的一系列过滤器和处理程序：

## 4.1、API请求流程图

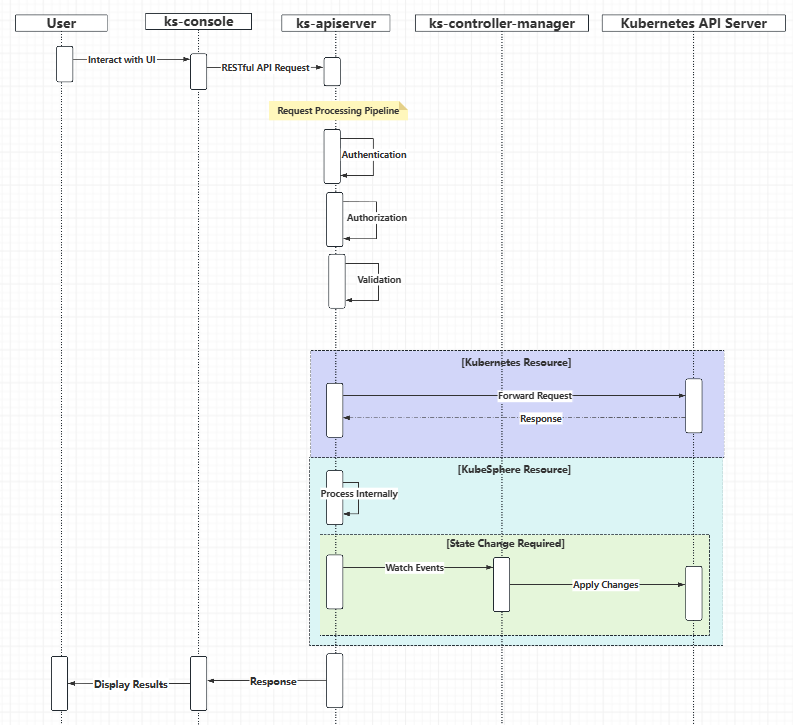


请求处理管道包括：

* WithRequestInfo：提取请求信息并将其添加到上下文中
* WithAuthentication：使用配置的提供程序对用户进行身份验证
* WithAuthorization：检查用户是否有权执行作
* WithDispatch：将请求路由到相应的处理程序
  + WithKubeAPIServer：用于 Kubernetes API 请求
  + WithMultiCluster：用于成员集群请求

## 4.2、请求处理流程

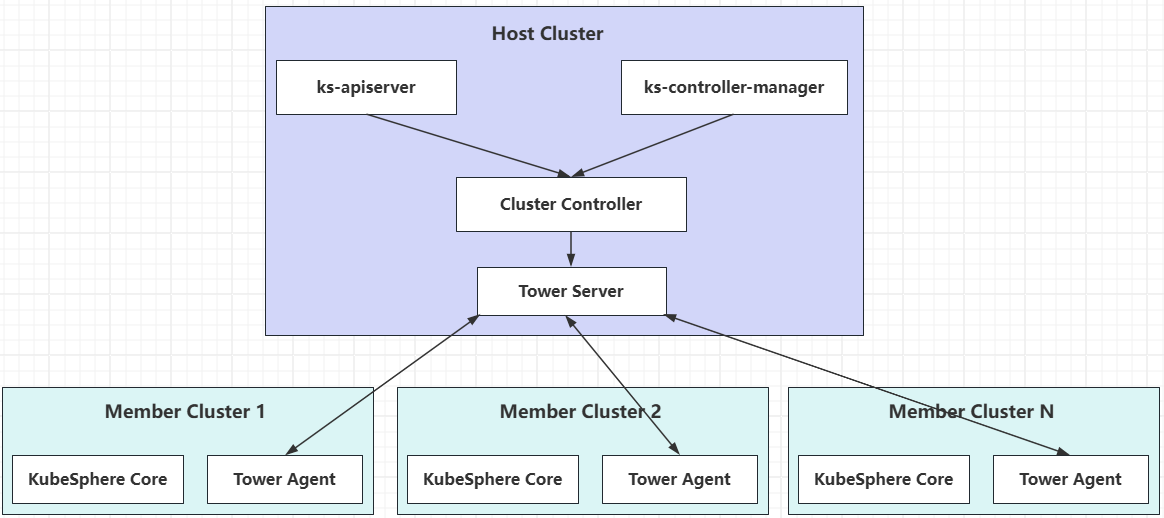
KubeSphere 遵循标准的请求-响应模式，请求通过一系列中间件组件进行处理，用于身份验证、授权和验证，然后由相应的控制器处理。



# 多集群管理

KubeSphere 支持通过主机-成员架构管理多个 Kubernetes 集群，提供集中控制平面：

## 5.1、多集群架构图



1. Host Cluster：管理多个集群的中央控制平面
2. Member Clusters ：由主机集群管理的 Kubernetes 集群
3. Tower：用于安全集群通信的组件

* Tower 代理在成员集群上运行
* 建立与主机群集的安全连接
* 为成员集群管理启用代理连接

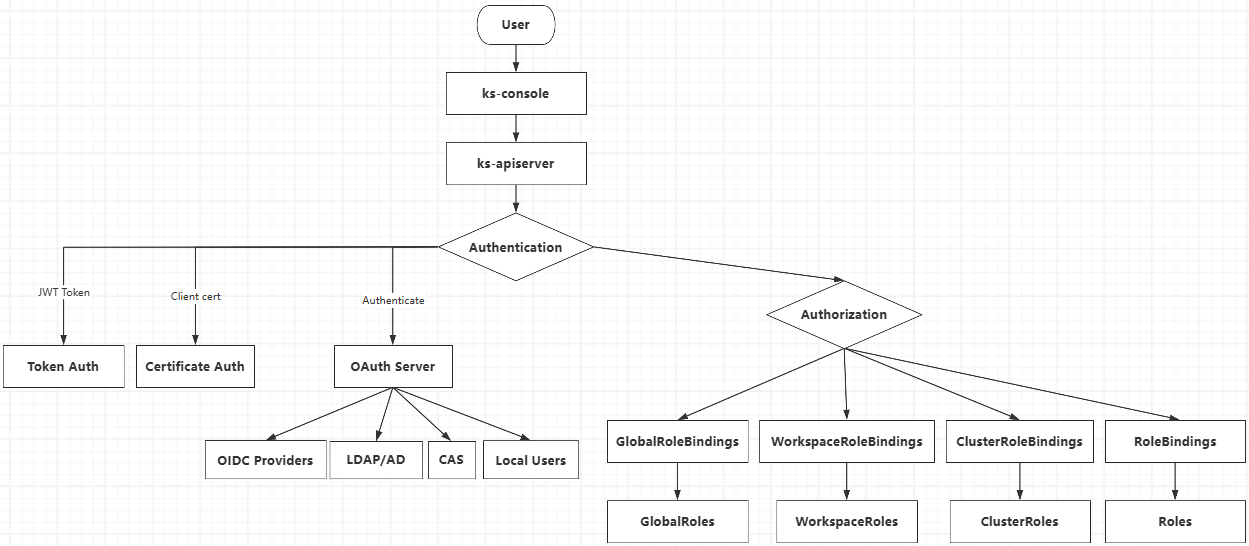
主机集群使用集群控制器运行中央控制平面，集群控制器管理成员集群的生命周期。Tower 代理在成员集群上运行，以建立与主机的安全连接。

* 集群配置验证
* 可配置的控制器重新同步周期
* 轻量级成员集群安装
* 改进了集群状态更新

# 身份验证和授权

KubeSphere 提供了全面的身份验证和授权系统。

## 6.1、身份验证流程图



系统支持多种身份验证方法：

* **OAuth**：与外部身份提供商集成
* **JWT 令牌** ：安全 API 访问
* **Client Certificates** ：服务到服务身份验证

授权通过分层 RBAC 系统处理，该系统具有全局、工作区、集群和命名空间级角色和角色绑定。

## 6.1、认证

API 服务器支持多种身份验证方法：

* 身份提供商的 OAuth/OpenID Connect 集成
* 基于 JWT 令牌的身份验证
* X.509 客户端证书身份验证
* 基本身份验证

身份验证是使用请求处理管道中的验证器链实现的。

## 6.2、鉴权

KubeSphere 使用分层 RBAC 系统进行授权：

* Global Roles：平台范围的权限
* Workspace Roles ：工作区中的权限
* Cluster Roles ：集群内的权限
* Namespace Roles ：命名空间中的权限

授权机制评估每个级别的权限，以确定用户是否可以执行作。

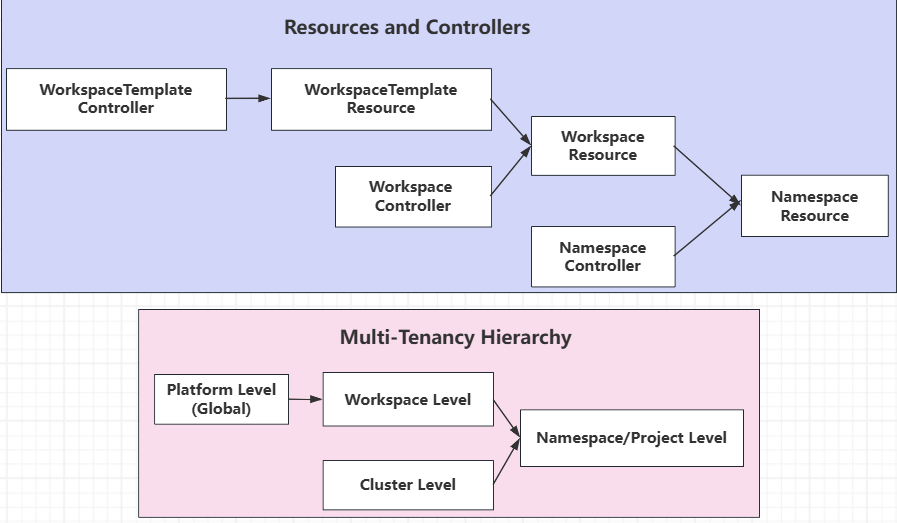
pkg/apiserver/apiserver.go 236-290

# 多租户模型

KubeSphere 实现了多级租户模型来确保隔离和资源管理。

## 7.1、租户模型分层

KubeSpherede 的分层多租户模型，分为四个级别：

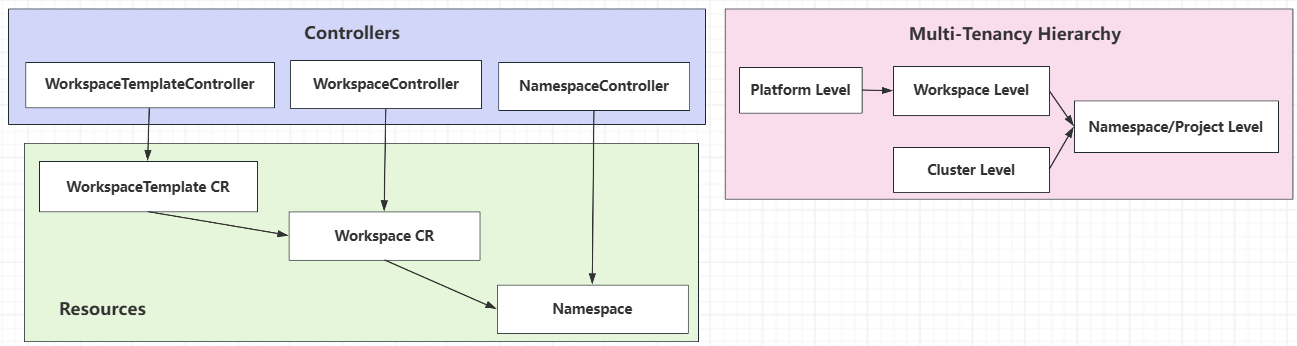


1. Platform Level：全局管理
2. Workspace Level ：项目和资源的逻辑分组
3. Cluster Level ：管理多个 Kubernetes 集群
4. Namespace/Project Level：Kubernetes 命名空间中的资源

租户管理在 tenant 运算符 （pkg/models/tenant/tenant.go） 中实现，该运算符处理工作区模板、工作区和命名空间。工作区对相关项目（命名空间）进行分组，并提供访问控制边界。命名空间标有工作区标签 （kubesphere.io/workspace） 以将它们与工作区关联：

|  |
| --- |
| // labelNamespaceWithWorkspaceName adds a kubesphere.io/workspace=[workspaceName] label to namespace  func labelNamespaceWithWorkspaceName(namespace \* corev1.Namespace, workspaceName string) \* corev1.Namespace {  if namespace.Labels == nil {  namespace.Labels = make(map[string] string, 0)  }  namespace.Labels[tenantv1beta1.WorkspaceLabel] = workspaceName  return namespace  } |

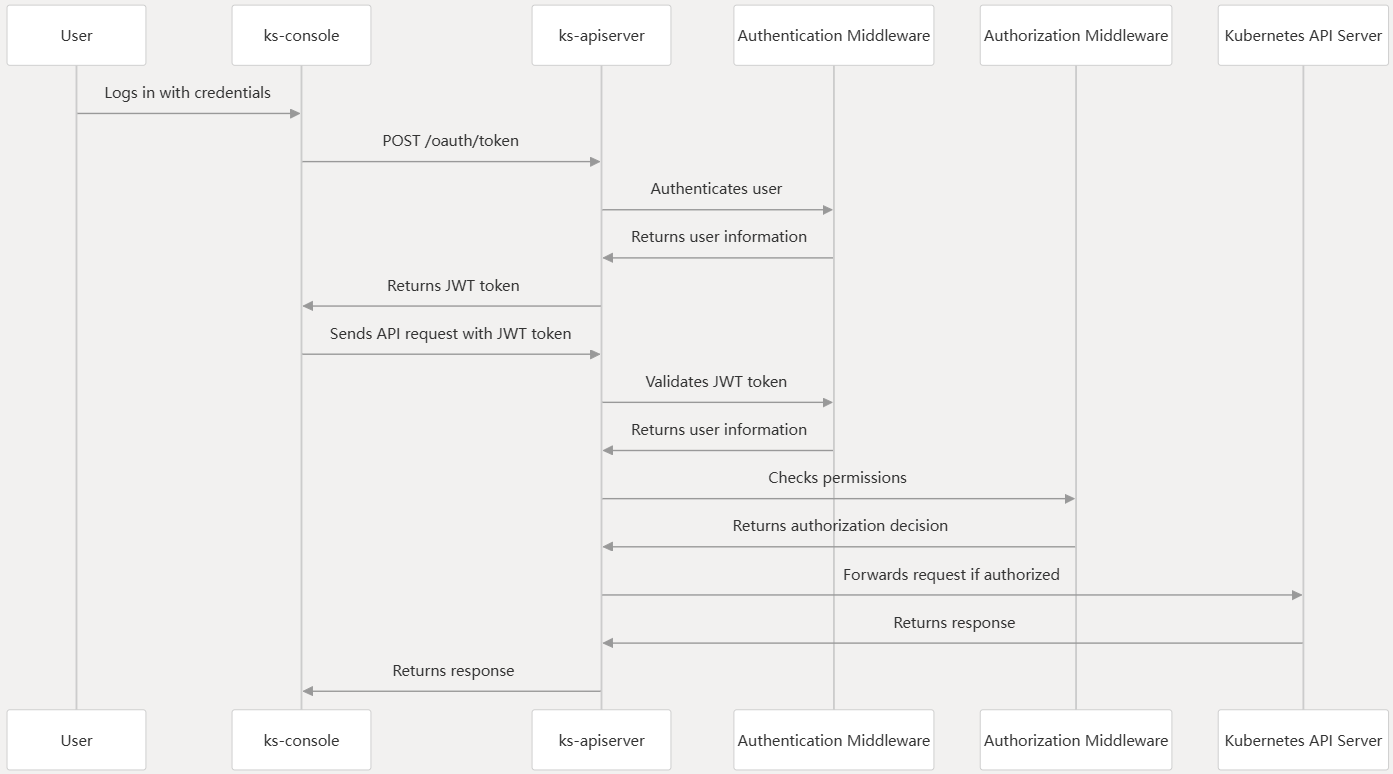
## 7.2、多租户层次结构图



WorkspaceTemplate 在主机集群中定义，并作为 Workspaces 同步到成员集群。命名空间通过标签和注释与 WorkSpaces 关联。这种多级方法在不同团队和项目之间提供了隔离，同时实现了高效的资源共享。

## 7.3、基于角色的访问控制 （RBAC）

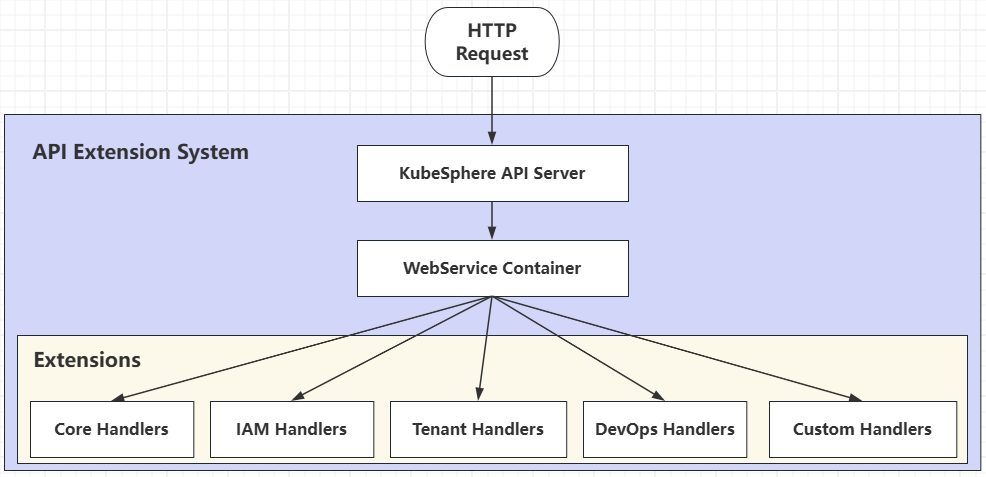
认证鉴权流程



# 扩展组件

## 8.1、扩展架构

KubeSphere包含灵活的API扩展机制：



可以通过实现其余部分来添加新的 API 扩展 。Handler 接口并注册到容器。API 服务器为不同的功能注册了各种处理程序：

|  |
| --- |
| func(s \* APIServer) installKubeSphereAPIs() {  handlers: = [] rest.Handler {  configv1alpha2.NewHandler( & s.Options, s.RuntimeClient),  resourcev1alpha3.NewHandler(s.RuntimeCache, counter, s.K8sVersion),  operationsv1alpha2.NewHandler(s.RuntimeClient),  resourcesv1alpha2.NewHandler(s.RuntimeClient, s.K8sVersion, s.K8sClient.Master(), s.TerminalOptions),  tenantapiv1alpha3.NewHandler(s.RuntimeClient, s.K8sVersion, s.ClusterClient, amOperator, imOperator, rbacAuthorizer),  // ...more handlers  }  for \_,  handler: = range handlers {  urlruntime.Must(handler.AddToContainer(s.container))  }  } |

## 8.2、核心扩展组件

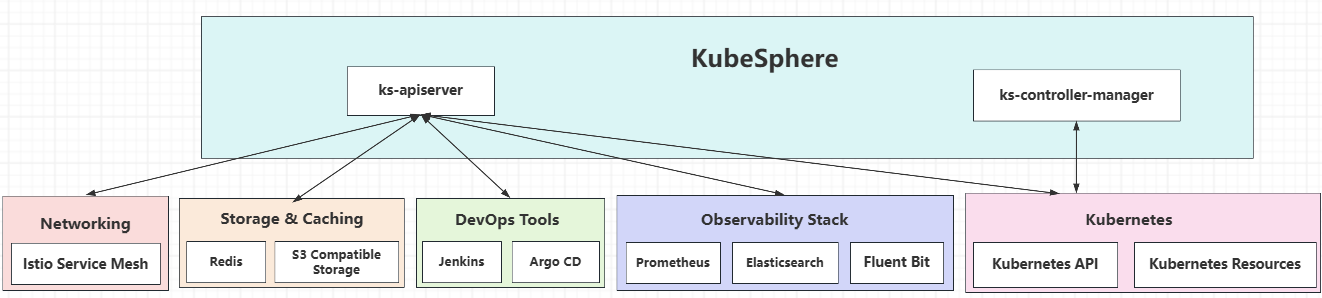
KubeSphere 提供了许多扩展组件，可以根据用户需求进行安装：

|  |  |  |
| --- | --- | --- |
| **Component** | **描述** | **特征** |
| DevOps | CI/CD pipeline集成 | Jenkins 集成、GitOps 与 Argo CD、S2I/B2I |
| Service Mesh | 基于 Istio 的服务网格 | 流量管理、可视化、跟踪 |
| Observability | 可观测性，监控、日志记录、警报 | Prometheus、Elasticsearch、自定义指标、警报 |
| App Store | 应用程序生命周期管理 | 基于 Helm 的应用程序、模板、版本控制 |
| Edge Computing | 边缘节点管理 | KubeEdge 集成，边缘工作负载调度 |
| Networking | 高级联网 | 多个 CNI 支持、网络策略、IP 池 |

扩展组件基于 Kubernetes 自定义资源定义 （CRD） 和控制器构建，扩展了 Kubernetes API 以支持 KubeSphere 特定的资源。

## 8.3、与外部系统集成

KubeSphere 与各种外部系统集成以提供全面的功能：



**Kubernetes 集成**

KubeSphere 构建在 Kubernetes 之上并扩展了其功能。它与 Kubernetes API 服务器通信以进行资源管理，并将自定义资源存储在 Kubernetes API 服务器中。

**可观测性集成**

KubeSphere 集成：

* Prometheus：用于指标收集和监控
* Elasticsearch：用于日志存储和分析
* Fluent Bit：用于日志收集

**DevOps 集成**

KubeSphere 通过与以下集成来提供 CI/CD 能力：

* Jenkins：用于持续集成
* Argo CD用于持续交付

**存储和缓存**

* Redis for caching：用于缓存
* S3-compatible storage ：用于构件和备份的 S3 兼容存储

# 关键依赖项

KubeSphere 依赖于几个关键依赖项：

* Kubernetes （v0.31.x）：核心容器编排平台
* Helm （v3.16.x）：Kubernetes 应用程序的软件包管理器
* Prometheus：监控和警报工具包
* Elasticsearch：日志存储和搜索引擎
* Istio：微服务服务网格
* Jenkins：CI/CD 自动化服务器

这些依赖项通过 Go 模块进行管理，如 go.mod 文件中所示。

# 总结

KubeSphere 的架构遵循微内核 + 扩展组件设计，主要有三个组件：API 服务器、控制器管理器和控制台。此体系结构支持：

1. Flexibility 灵活性：扩展的动态管理
2. Scalability 可伸缩性：多集群管理
3. Multi-tenancy 多租户：分层访问控制
4. Extensibility 可扩展性：易于与各种系统集成

KubeSphere 为管理 Kubernetes 集群和应用程序提供了一个全面的平台。其模块化架构允许根据需要安装组件，其多集群功能支持管理分布式 Kubernetes 环境。API 服务器充当网关，处理身份验证、授权和请求路由。控制器管理器通过多个控制器实现业务逻辑。控制台提供了一个用户友好的 Web 界面。基于 Web 的界面使具有不同 Kubernetes 专业知识水平的用户能够访问它，而扩展系统则为高级用例提供了灵活性。