OpenShift Container Platform 4.16

架构

OpenShift Container Platform 架构概述

Last Updated: 2024-10-03

OpenShift Container Platform 4.16 架构

OpenShift Container Platform 架构概述

法律通告

摘要

本文概述 OpenShift Container Platform 中的平台和应用架构。

目录

第 1 章 架构概述

OpenShift Container Platform 是一个基于云的 Kubernetes 容器平台。OpenShift Container Platform 的基础基于 Kubernetes，因此共享相同的技术。如需了解更多有关 OpenShift Container Platform 和 Kubernetes 的信息，请参阅 产品架构。

1.1. OPENSHIFT CONTAINER PLATFORM 架构的常见术语表

该术语表定义了架构内容中使用的常见术语。

访问策略

组角色，用于指明集群内的用户、应用程序和实体如何与另一个角色进行交互。访问策略会增加集群安全性。

准入插件

准入插件强制执行安全策略、资源限制或配置要求。

身份验证

为了控制对 OpenShift Container Platform 集群的访问，集群管理员可以配置用户身份验证，并确保只有批准的用户访问集群。要与 OpenShift Container Platform 集群交互，您必须对 OpenShift Container Platform API 进行身份验证。您可以通过在您对 OpenShift Container Platform API 的请求中提供 OAuth 访问令牌或 X.509 客户端证书来进行身份验证。

bootstrap

运行最小 Kubernetes 并部署 OpenShift Container Platform control plane 的临时机器。

证书签名请求 (CSR)

资源请求指示签名者为证书签名。此请求可能会获得批准或拒绝。

Cluster Version Operator (CVO)

检查 OpenShift Container Platform Update Service 的 Operator，它根据图中的当前组件版本和信息查看有效的更新和更新路径。

Compute 节点

负责执行集群用户工作负载的节点。Compute 节点也称为 worker 节点。

配置偏移

在节点上配置与机器配置指定的内容不匹配的情况。

containers

包括软件及其所有依赖项的轻量级和可执行镜像。由于容器虚拟化操作系统，您可以在任何位置运行容器，从数据中心到公共或私有云到本地主机。

容器编配引擎

用于实现容器部署、管理、扩展和联网的软件。

容器工作负载

在容器中打包和部署的应用程序。

控制组 (cgroups)

将进程集合分区到组中，以管理和限制资源进程占用。

control plane（控制平面）

一个容器编配层，用于公开 API 和接口来定义、部署和管理容器的生命周期。control plane 也称为 control plane 机器。

CRI-O

Kubernetes 原生容器运行时实现，可与操作系统集成以提供高效的 Kubernetes 体验。

部署

维护应用程序生命周期的 Kubernetes 资源对象。

Docker

包含要在终端执行以编译镜像的用户命令的文本文件。

托管 control plane

OpenShift Container Platform 功能，允许从其 data plane 和 worker 在 OpenShift Container Platform 集群上托管 control plane。这个模型执行以下操作：

- 优化 control plane 所需的基础架构成本。
- 改进集群创建时间。
- 启用使用 Kubernetes 原生高级别元语托管 control plane。例如，部署和有状态的集合。
- 在 control plane 和工作负载之间允许强大的网络分段。

混合云部署

部署在跨裸机、虚拟、私有和公有云环境中提供一致的平台。这提供了速度、灵活性和可移植性。

Ignition

RHCOS 在初始配置期间用于操作磁盘的实用程序。它可完成常见的磁盘任务，如分区磁盘、格式化分区、写入文件和配置用户等。

安装程序置备的基础架构

安装程序部署并配置运行集群的基础架构。

kubelet

在集群的每个节点上运行的一个主节点代理，以确保容器在 pod 中运行。

kubernetes manifest（清单）

JSON 或 YAML 格式的 Kubernetes API 对象的规格。配置文件可以包含部署、配置映射、secret、守护进程集。

机器配置守护进程 (MCD)

定期检查节点进行配置偏移的守护进程。

Machine Config Operator（MCO）

将新配置应用到集群机器的 Operator。

机器配置池 (MCP)

一组基于它们处理的资源的机器（如 control plane 组件或用户工作负载）。

metadata

有关集群部署工件的附加信息。

微服务

编写软件的方法。应用程序可以使用微服务相互独立，划分为最小的组件。

镜像 registry

包含 OpenShift Container Platform 镜像的 mirror registry。

单体式应用程序

自我包含、构建并打包为单个组件的应用程序。

命名空间

命名空间隔离所有进程可见的特定系统资源。在一个命名空间中，只有属于该命名空间的进程才能看到这些资源。

networking

OpenShift Container Platform 集群的网络信息。

node

OpenShift Container Platform 集群中的 worker 机器。节点是虚拟机 (VM) 或物理计算机。

OpenShift CLI (oc)

在终端上运行 OpenShift Container Platform 命令的命令行工具。

OpenShift Dedicated

Amazon Web Services (AWS) 和 Google Cloud Platform (GCP)上的托管 RHEL OpenShift Container Platform 产品。OpenShift Dedicated 侧重于构建和扩展应用程序。

OpenShift Update Service (OSUS)

对于可访问互联网的集群，Red Hat Enterprise Linux (RHEL) 通过 OpenShift 更新服务提供更新，它作为公共 API 后面的一个托管服务运行。

OpenShift 镜像 registry

OpenShift Container Platform 提供的 registry 来管理镜像。

Operator

在 OpenShift Container Platform 集群中打包、部署和管理 Kubernetes 应用程序的首选方法。Operator 将人类操作知识编码到一个软件程序中，易于打包并与客户共享。

OperatorHub

包含要安装的 OpenShift Container Platform Operator 的平台。

Operator Lifecycle Manager (OLM)

OLM 可帮助您安装、更新和管理 Kubernetes 原生应用程序的生命周期。OLM 是一个开源工具包，用于以有效、自动化且可扩展的方式管理 Operator。

OSTree

对于基于 Linux 的操作系统升级系统，会对完整的文件系统树执行原子升级。OSTree 使用可寻址对象存储跟踪对文件系统树的有意义的更改，旨在补充现有的软件包管理系统。

无线 (OTA) 更新

OpenShift Container Platform Update Service（OpenShift Update Service，简称 OSUS）为 OpenShift Container Platform（包括 Red Hat Enterprise Linux CoreOS（RHCOS））提供了无线更新（over-the air update）功能。

pod

一个或多个带有共享资源（如卷和 IP 地址）的容器，在 OpenShift Container Platform 集群中运行。pod 是定义、部署和管理的最小计算单元。

私有 registry

OpenShift Container Platform 可以使用实施容器镜像 registry API 作为镜像源的任何服务器，供开发人员推送和拉取其私有容器镜像。

公共 registry

OpenShift Container Platform 可以使用实施容器镜像 registry API 作为镜像源的任何服务器，供开发人员推送和拉取其公共容器镜像。

RHEL OpenShift Container Platform Cluster Manager

一个受管服务，您可以安装、修改、操作和升级 OpenShift Container Platform 集群。

RHEL Quay Container Registry

为 OpenShift Container Platform 集群提供大多数容器镜像和 Operator 的 Quay.io 容器 registry。

复制控制器

指示一次需要运行多少个 pod 副本资产。

基于角色的访问控制 (RBAC)

重要的安全控制，以确保集群用户和工作负载只能访问执行其角色所需的资源。

route

路由用于公开服务，以允许从 OpenShift Container Platform 实例外部的用户和应用程序对 pod 进行网络访问。

扩展

资源容量的增加或减少。

service

服务在一组 pod 上公开正在运行的应用程序。

Source-to-Image (S2I) 镜像

基于 OpenShift Container Platform 中应用源代码的编程语言创建的镜像，以部署应用程序。

storage

OpenShift Container Platform 支持许多类型的存储，包括内部存储和云供应商。您可以在 OpenShift Container Platform 集群中管理持久性和非持久性数据的容器存储。

Telemetry

此组件用于收集 OpenShift Container Platform 的大小、健康和状态等信息。

模板

模板描述了一组可参数化和处理的对象，用于生成对象列表，供 OpenShift Container Platform 用于创建。

用户置备的基础架构

您可以在自己提供的基础架构上安装 OpenShift Container Platform。您可以使用安装程序来生成置备集群基础架构所需的资产，再创建集群基础架构，然后将集群部署到您提供的基础架构中。

Web 控制台

用于管理 OpenShift Container Platform 的用户界面(UI)。

worker 节点

负责执行集群用户工作负载的节点。Worker 节点也称为计算节点。

其他资源

- 如需有关网络的更多信息，请参阅 OpenShift Container Platform 网络。
- 如需有关存储的更多信息，请参阅 OpenShift Container Platform 存储。
- 如需有关身份验证的更多信息，请参阅 OpenShift Container Platform 身份验证。
- 如需有关 Operator Lifecycle Manager (OLM) 的更多信息，请参阅 OLM。
- 有关日志记录的更多信息，请参阅关于日志记录。
- 如需有关无线 (OTA) 更新的更多信息，请参阅 OpenShift 更新简介。

1.2. 关于安装和更新

作为集群管理员，您可以使用 OpenShift Container Platform 安装程序 使用以下方法之一安装集群：

- 安装程序置备的基础架构
- 用户置备的基础架构

1.3. 关于 CONTROL PLANE

control plane 管理 worker 节点和集群中的 pod。您可以使用机器配置池(MCP)配置节点。MCP 是基于它们处理的资源的机器组，如 control plane 组件或用户工作负载。OpenShift Container Platform 为主机分配不同的角色。这些角色定义集群中的机器的功能。集群包含标准 control plane 和 worker 角色类型的定义。

您可以使用 Operator 来打包、部署和管理 control plane 上的服务。Operator 在 OpenShift Container Platform 中是很重要的组件，因为它们提供以下服务：

- 执行健康检查
- 提供监视应用程序的方法
- 管理无线更新
- 确保应用程序保持指定状态

1.4. 关于面向开发人员的容器化应用程序

作为开发者，您可以使用不同的工具、方法和格式来根据您的独特要求开发容器化应用程序，例如：

- 使用各种 build-tool、base-image 和 registry 选项构建简单容器应用程序。
- 使用 OperatorHub 和模板等支持组件来开发您的应用程序。
- 将应用程序打包并部署为 Operator。

您还可以创建 Kubernetes 清单，并将其存储在 Git 存储库中。Kubernetes 适用于称为 pod 的基本单元。pod 是集群中正在运行的进程的单一实例。Pod 可以包含一个或多个容器。您可以通过对一组 pod 及其访问策略进行分组来创建服务。服务为其他应用程序提供永久内部 IP 地址和主机名，供在 pod 创建和销毁时使用。Kubernetes 根据应用程序的类型定义工作负载。

1.5. ABOUT RED HAT ENTERPRISE LINUX COREOS (RHCOS) AND IGNITION

作为集群管理员，您可以执行以下 Red Hat Enterprise Linux CoreOS(RHCOS)任务：

- 了解下一代 单用途容器操作系统技术。
- 选择如何配置 Red Hat Enterprise Linux CoreOS(RHCOS)
- 选择如何部署 Red Hat Enterprise Linux CoreOS(RHCOS)：
  - 安装程序置备的部署
  - 用户置备的部署

OpenShift Container Platform 安装程序创建部署集群所需的 Ignition 配置文件。Red Hat Enterprise Linux CoreOS(RHCOS)在初始配置过程中使用 Ignition 执行常见磁盘任务，如分区、格式化、写入文件和配置用户。首次启动时，Ignition 从安装介质或您指定的位置读取其配置，并将配置应用到机器。

您可以了解 Ignition 的工作原理，以及 OpenShift Container Platform 集群中的 Red Hat Enterprise Linux CoreOS(RHCOS)机器的过程，查看 Ignition 配置文件，在安装后更改 Ignition 配置。

1.6. 关于准入插件

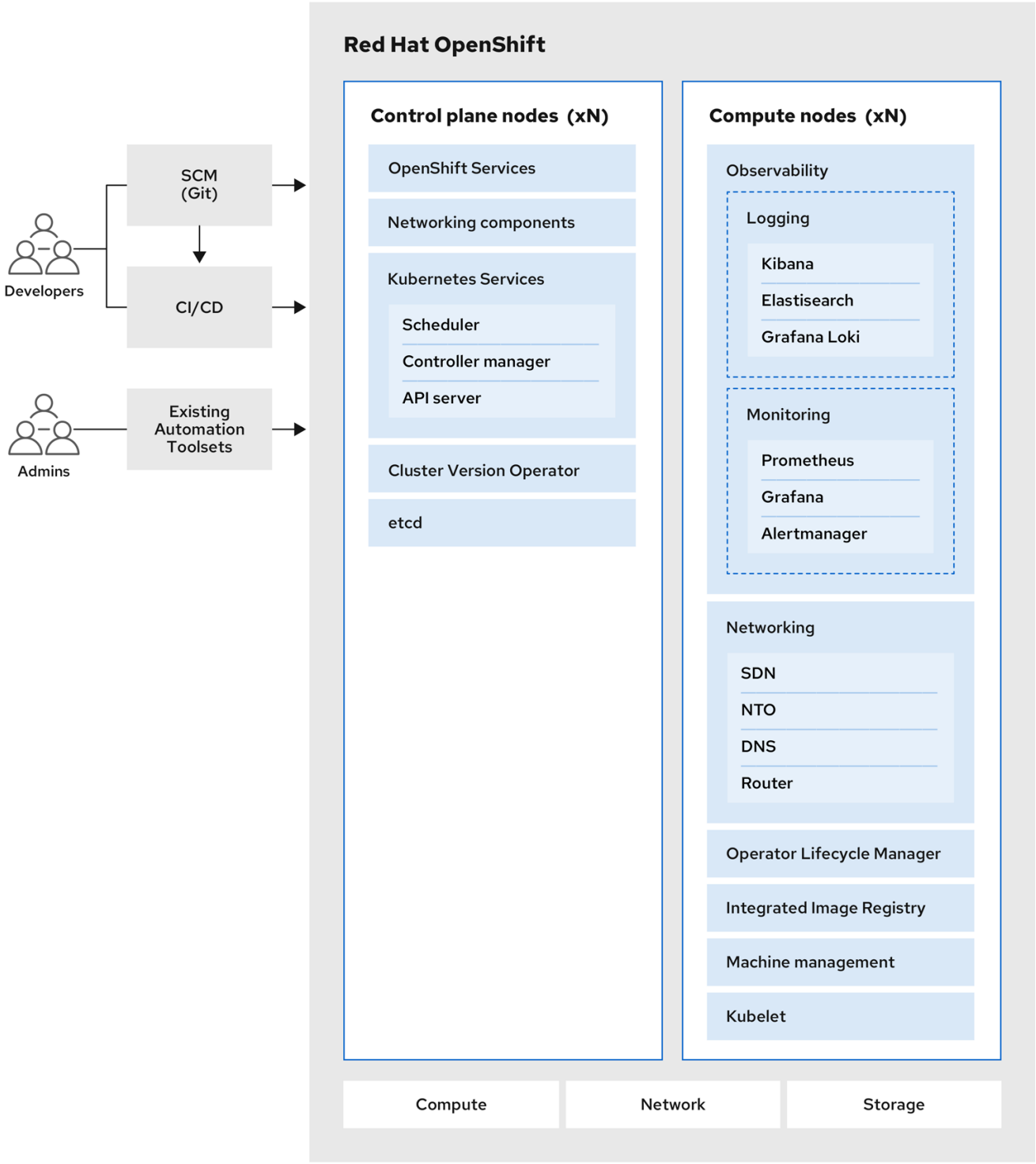
您可以使用 准入插件 规范 OpenShift Container Platform 的功能。在对资源请求进行身份验证和授权后，准入插件会截获对 master API 的资源请求以验证资源请求，并确保扩展策略得到遵循。准入插件用于强制执行安全策略、资源限制、配置要求和其他设置。

第 2 章 OPENSHIFT CONTAINER PLATFORM 架构

2.1. OPENSHIFT CONTAINER PLATFORM 简介

OpenShift Container Platform 是用于开发和运行容器化应用程序的平台。它旨在允许支持的应用程序和数据中心从少量机器和应用程序扩展到为数百万客户端服务的数千台机器。

OpenShift Container Platform 以 Kubernetes 为基础，为大规模电信、流视频、游戏、银行和其他应用提供引擎技术。借助红帽开放技术中的实现，您可以将容器化应用程序从单一云扩展到内部和多云环境。



2.1.1. 关于 Kubernetes

尽管容器镜像和从中运行的容器是现代应用程序开发的主要构建块，但要大规模运行它们，则需要可靠且灵活的分发系统。Kubernetes 是编配容器的事实标准。

Kubernetes 是一个开源容器编配引擎，用于自动化容器化应用程序的部署、扩展和管理。Kubernetes 的一般概念比较简单：

- 从一个或多个 worker 节点开始，以运行容器工作负载。
- 从一个或多个 control plane 节点管理这些工作负载的部署。
- 将容器嵌套到名为 pod 的部署单元中。使用 pod 可以为容器提供额外的元数据，并可在单个部署实体中对多个容器进行分组。
- 创建特殊种类的资产。例如，服务由一组 pod 及定义了访问方式的策略来表示。此策略可使容器连接到所需的服务，即便容器没有用于服务的特定 IP 地址。复制控制器（replication controller）是另一种特殊资产，用于指示一次需要运行多少个 pod 副本。您可以使用此功能来自动扩展应用程序，以适应其当前的需求。

短短数年，Kubernetes 已在大量的云和本地环境中被采用。借助开源开发模型，拥护和可以通过为组件（如网络、存储和身份验证）实施不同的技术来扩展 Kubernetes 的功能。

2.1.2. 容器化应用程序的好处

与使用传统部署方法相比，使用容器化应用程序具有许多优势。过去应用程序要安装到包含所有依赖项的操作系统上，容器能让一个应用程序随身携带自己的依赖项。创建容器化应用程序有很多好处。

2.1.2.1. 操作系统的好处

容器使用不含内核的小型专用 Linux 操作系统。它们的文件系统、网络、cgroups、进程表和命名空间与主机 Linux 系统分开，但容器可以在必要时与主机无缝集成。容器以 Linux 为基础，因此可以利用快速创新的开源开发模型带来的所有优势。

因为每个容器都使用专用的操作系统，所以您能够在同一主机上部署需要冲突软件依赖项的不同应用程序。每个容器都带有各自的依赖软件，并且管理自己的接口，如网络和文件系统，因此应用程序无需争用这些资产。

2.1.2.2. 部署和扩展优势

如果您在应用程序的主要版本之间进行滚动升级，则可以持续改进应用程序，既不会造成停机，又能仍然保持与当前版本的兼容性。

您还可以与现有版本一起部署和测试应用程序的新版本。容器通过测试后，只要部署更多新容器并删除旧容器便可。

由于应用程序的所有软件依赖项都在容器本身内解决，因此数据中心的每台主机上都能使用标准的操作系统。您无需逐一为应用主机配置特定的操作系统。当数据中心需要更多容量时，您可以部署另一个通用主机系统。

同样，扩展容器化应用程序也很简单。OpenShift Container Platform 提供了一种简单的、标准方式的容器化服务扩展功能。例如，如果将应用程序构建为一组微服务，而非大型的单体式应用程序，您可以分别扩展各个微服务来满足需求。有了这一能力，您可以只扩展需要的服务，而不是整个应用程序，从而在使用最少资源的前提下满足应用程序需求。

2.1.3. OpenShift Container Platform 概述

OpenShift Container Platform 为 Kubernetes 带来企业级增强，具体包括以下所列：

- 混合云部署。您可以将 OpenShift Container Platform 集群部署到各种公有云平台或数据中心。
- 集成了红帽技术。OpenShift Container Platform 中的主要组件源自 Red Hat Enterprise Linux（RHEL） 和相关的红帽技术。OpenShift Container Platform 得益于红帽企业级优质软件的严格测试和认证计划。
- 开源开发模型。开发以开放方式完成，源代码可从公共软件存储库中获得。这种开放协作促进了快速创新和开发。

虽然 Kubernetes 擅长管理应用程序，但它并未指定或管理平台级要求或部署过程。强大而灵活的平台管理工具和流程是 OpenShift Container Platform 4.16 具备的重要优势。以下各节介绍 OpenShift Container Platform 的一些独特功能和优势。

2.1.3.1. 定制操作系统

OpenShift Container Platform 使用 Red Hat Enterprise Linux CoreOS (RHCOS)，它是一款面向容器的操作系统，专为从 OpenShift Container Platform 运行容器化应用程序而设计，并可使用新工具提供快速安装、基于 Operator 的管理和简化的升级。

RHCOS 包括：

- Ignition，OpenShift Container Platform 将其用作首次启动系统配置来进行机器的初次上线和配置。
- CRI-O，Kubernetes 的原生容器运行时实现，可与操作系统紧密集成来提供高效和优化的 Kubernetes 体验。CRI-O，提供用于运行、停止和重启容器的工具。它完全取代了 OpenShift Container Platform 3 中使用的 Docker Container Engine。
- Kubelet，Kubernetes 的主要节点代理，负责启动和监视容器。

在 OpenShift Container Platform 4.16 中，所有 control plane 都需要使用 RHCOS。但 compute（计算）机器（也被称为 worker ）可以使用 Red Hat Enterprise Linux (RHEL) 做为操作系统。如果选择使用 RHEL worker，与将 RHCOS 用于所有集群机器相比，您必须执行更多的系统维护。

2.1.3.2. 简化的安装和更新流程

使用 OpenShift Container Platform 4.16 时，如果您拥有具有正确权限的帐户，通过运行单个命令并提供几个值，就能在支持的云中部署生产集群。如果使用支持的平台，您还可以自定义云安装或在数据中心中安装集群。

对于将 RHCOS 用于所有机器的集群，不论是更新还是升级，OpenShift Container Platform 都是一个简单又高度自动化的流程。由于 OpenShift Container Platform 可以从中央 control plane 全面控制每台机器上运行的系统和服务（包括操作系统本身），因此升级被设计为一个自动事件。如果集群包含 RHEL worker 机器，则 control plane 可从简化的更新过程中受益，但您还需执行更多任务来升级 RHEL 机器。

2.1.3.3. 其他主要功能

Operator 既是 OpenShift Container Platform 4.16 代码库的基本单元，又是部署供应用程序使用的应用程序和软件组件的便捷方式。在 OpenShift Container Platform 中，Operator 可充当平台的基础，不再需要手动升级操作系统和 control plane 应用程序。OpenShift Container Platform Operator（如 Cluster Version Operator 和 Machine Config Operator）允许对这些关键组件进行简化的集群范围内管理。

Operator Lifecycle Manager (OLM) 和 OperatorHub 提供了相应的工具，可用于存储 Operator 并将其分发给开发和部署应用程序的人员。

Red Hat Quay Container Registry 是一个 Quay.io 容器 registry，为 OpenShift Container Platform 集群提供大多数容器镜像和 Operator。Quay.io 是 Red Hat Quay 的一个公共 registry 版本，可存储数百万镜像和标签。

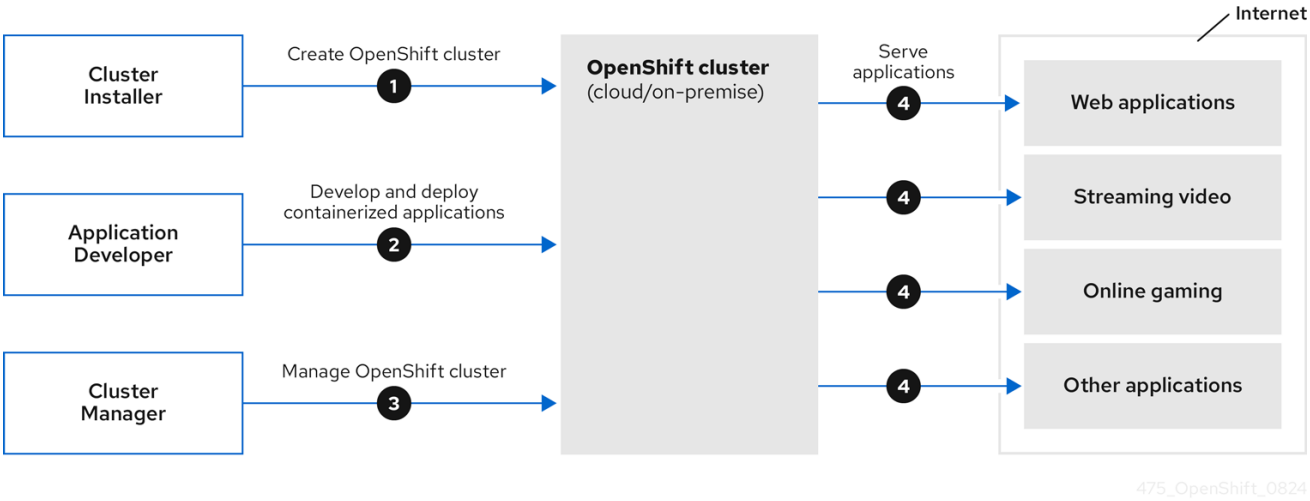
OpenShift Container Platform 中的其他 Kubernetes 增强功能包括软件定义网络 (SDN)、身份验证、日志聚合、监视和路由方面的改进。OpenShift Container Platform 还提供功能齐全的 web 控制台和自定义 OpenShift CLI (**oc**) 界面。

2.1.3.4. OpenShift Container Platform 生命周期

下图显示了 OpenShift Container Platform 的基本生命周期：

- 创建 OpenShift Container Platform 集群
- 管理集群
- 开发和部署应用程序
- 扩展应用程序

图 2.1. OpenShift Container Platform 高级概述



2.1.4. OpenShift Container Platform 互联网访问

在 OpenShift Container Platform 4.16 中，您需要访问互联网来安装集群。

您必须具有以下互联网访问权限：

- 访问 OpenShift Cluster Manager 以下载安装程序并执行订阅管理。如果集群可以访问互联网，并且没有禁用 Telemetry，该服务会自动授权您的集群。
- 访问 Quay.io，以获取安装集群所需的软件包。
- 获取执行集群更新所需的软件包。

重要

如果您的集群无法直接访问互联网，则可以在置备的某些类型的基础架构上执行受限网络安装。在此过程中，您可以下载所需的内容，并使用它为镜像 registry 填充安装软件包。对于某些安装类型，集群要安装到的环境不需要访问互联网。在更新集群之前，您要更新镜像 registry（mirror registry） 的内容。

第 3 章 安装和更新

3.1. 关于 OPENSHIFT CONTAINER PLATFORM 安装

OpenShift Container Platform 安装程序提供了四个部署集群的方法，相关信息包括在以下列表中：

- **交互式** ：您可以使用基于 Web 的 辅助安装程序（Assisted Installer）部署集群。对于可以连接到互联网的网络，这是一个理想的方法。Assisted Installer 是安装 OpenShift Container Platform 的最简单方法，它提供智能默认值，并在安装集群前执行预动态验证。它还提供了一个 RESTful API 用于自动化和高级配置场景。

- **本地基于代理的** ：对于断开连接的环境或有网络限制的环境，您可以使用基于代理的安装程序。它提供了 Assisted Installer 的许多优点，但您必须首先下载并配置基于代理的安装程序。使用命令行界面完成配置。这个方法适用于断开连接的环境。

- **自动** ：您可以在安装程序置备的基础架构中部署集群。安装程序使用每个集群主机的基板管理控制器 (BMC) 进行置备。您可以在有连接或断开连接的环境中部署集群。

- **完全控制**：您可以在自己准备和维护的基础架构上部署集群，这种方法提供了最大的定制性。您可以在有连接或断开连接的环境中部署集群。

每种方法部署的集群具有以下特征：

- 没有单点故障的高可用性基础架构，默认可用。

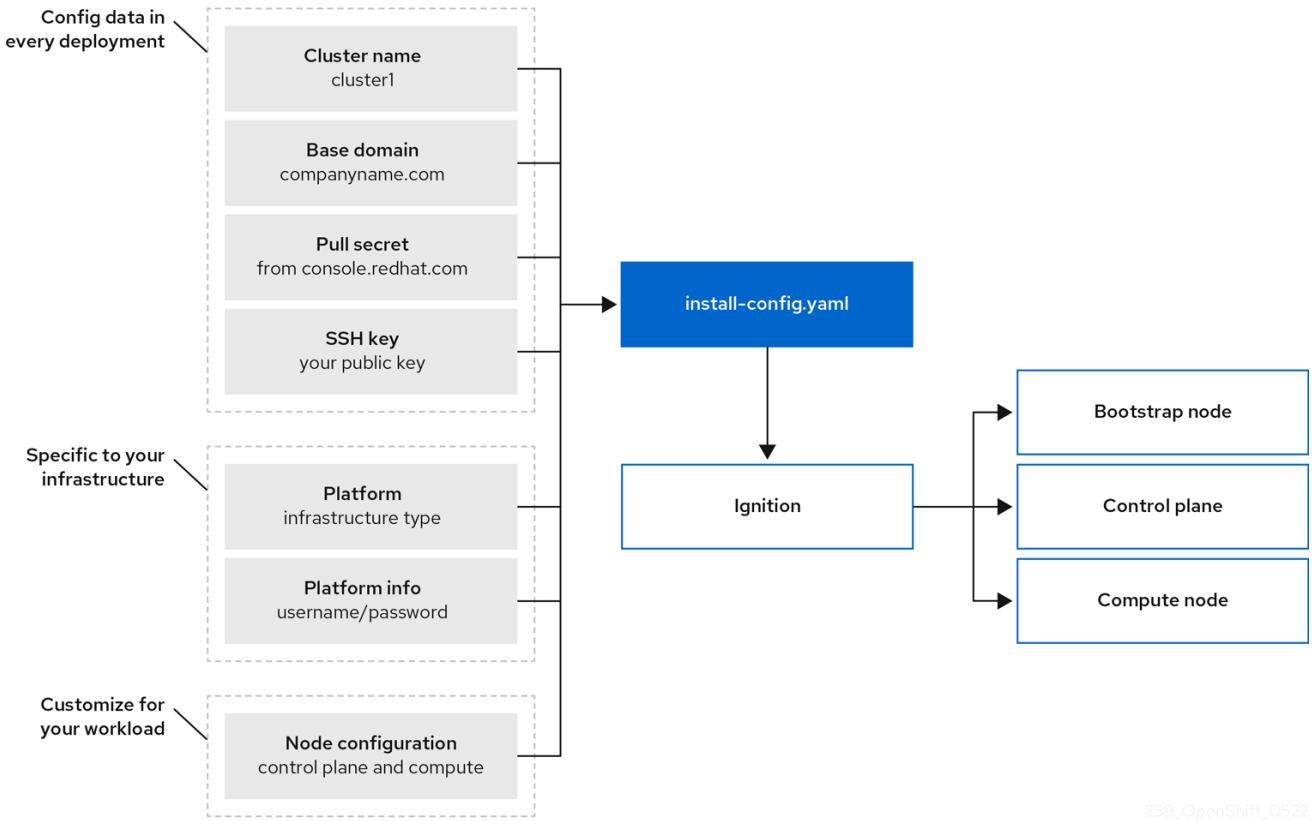
- 管理员可以控制要应用的更新，以及应用的时间。

3.1.1. 关于安装程序

您可以使用安装程序部署每种集群。安装程序会生成主要资产，如 bootstrap、control plane 和计算机器的 Ignition 配置文件。您可以使用这三个机器配置开始使用 OpenShift Container Platform 集群，它为您提供了正确配置的基础架构。

OpenShift Container Platform 安装程序使用一组目标和依赖项来管理集群安装。安装程序具有一组必须实现的目标，并且每个目标都有一组依赖项。因为每个目标仅关注其自己的依赖项，所以安装程序可以并行地实现多个目标，最终组成一个正常运行的集群。安装程序会识别并使用现有组件，而不是运行命令来再次创建它们，因为程序满足依赖项。

图 3.1. OpenShift Container Platform 安装目标和依赖项



3.1.2. 关于 Red Hat Enterprise Linux CoreOS (RHCOS)

在安装后，每一个集群机器都将使用 Red Hat Enterprise Linux CoreOS (RHCOS) 作为操作系统。RHCOS 是 Red Hat Enterprise Linux (RHEL) 的不可变容器主机版本，具有默认启用 SELinux 的 RHEL 内核。RHCOS 包括作为 Kubernetes 节点代理的 **kubelet**，以及为 Kubernetes 优化的 CRI-O 容器运行时。

OpenShift Container Platform 4.16 集群中的每一 control plane 机器都必须使用 RHCOS，其中包括一个关键的首次启动置备工具，称为 Ignition。这一工具让集群能够配置机器。操作系统更新作为可引导容器镜像（使用 **OSTree** 作为后端）提供，该镜像由 Machine Config Operator 在集群中部署。实际的操作系统更改通过使用 **rpm-ostree** 在每台机器上作为原子操作原位进行。通过结合使用这些技术，OpenShift Container Platform 可以像管理集群上的任何其他应用程序一样管理操作系统，通过原位升级使整个平台保持最新状态。这些原位更新可以减轻运维团队的负担。

如果将 RHCOS 用作所有集群机器的操作系统，则集群将管理其组件和机器的所有方面，包括操作系统在内。因此，只有安装程序和 Machine Config Operator 才能更改机器。安装程序使用 Ignition 配置文件设置每台机器的确切状态，安装后则由 Machine Config Operator 完成对机器的更多更改，例如应用新证书或密钥等。

3.1.3. OpenShift Container Platform 集群支持的平台

在 OpenShift Container Platform 4.16 中，您可以在以下平台上安装使用安装程序置备的基础架构集群：

- Amazon Web Services (AWS)
- 裸机
- Google Cloud Platform (GCP)

- IBM Cloud®
- Microsoft Azure
- Microsoft Azure Stack Hub
- Nutanix
- Red Hat OpenStack Platform (RHOSP)
  - 最新的 OpenShift Container Platform 版本支持最新的 RHOSP 长生命版本和中间版本。如需完整的 RHOSP 发行版本兼容性信息，请参阅 RHOSP 上的 OpenShift Container Platform 支持列表。
- VMware vSphere

对于所有这些集群，包括用来运行安装过程的计算机在内的所有机器都必须可直接访问互联网，以便为平台容器拉取镜像并向红帽提供 telemetry 数据。

重要

安装后，不支持以下更改：

- 混合云供应商平台.
- 混合云供应商组件。例如，在安装集群的平台上使用另一个平台的持久性存储框架。

在 OpenShift Container Platform 4.16 中，您可以在以下平台上安装使用用户置备的基础架构集群：

- AWS
- Azure
- Azure Stack Hub
- 裸机
- GCP
- IBM Power®
- IBM Z® 或 IBM® LinuxONE
- RHOSP
  - 最新的 OpenShift Container Platform 版本支持最新的 RHOSP 长生命版本和中间版本。如需完整的 RHOSP 发行版本兼容性信息，请参阅 RHOSP 上的 OpenShift Container Platform 支持列表。
- AWS 上的 VMware Cloud
- VMware vSphere

根据平台支持的情况，您可以在用户置备的基础架构上执行安装，以便您可以运行具有完整互联网访问的机器，将集群放在一个代理的后面，或者执行断开连接的安装。

在断开连接的网络安装中，您可以下载安装集群所需的镜像（image），将它们放在镜像 registry（mirror registry）中，然后使用那些数据安装集群。虽然您需要访问互联网来为平台容器拉取镜像，但在 vSphere 或裸机基础架构上进行断开连接的网络安装，您的集群机器不需要直接访问互联网。

OpenShift Container Platform 4.x Tested Integrations 页面中提供了有关针对不同平台进行集成测试的详细信息。

3.1.4. 安装过程

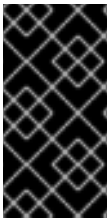
除了 Assisted Installer 外，当安装 OpenShift Container Platform 集群时，您必须从 OpenShift Cluster Manager Hybrid Cloud Console 上的适当的 **Cluster Type** 页面下载安装程序。此控制台管理：

- 帐户的 REST API。
- registry 令牌，这是用于获取所需组件的 pull secret。
- 集群注册，将集群身份与您的红帽帐户相关联，以便收集使用指标。

在 OpenShift Container Platform 4.16 中，安装程序是对一组资产执行一系列文件转换的 Go 二进制文件。与安装程序交互的方式因您的安装类型而异。考虑以下安装用例：

- 要使用 Assisted Installer 部署集群，您可以使用 Assisted Installer 配置集群设置。没有安装程序可以下载和配置。设置完集群配置后，您可以下载发现 ISO，然后使用该镜像引导集群机器。您可以使用 Assisted Installer 在完全集成的 Nutanix、vSphere 和裸机上安装集群，以及以前没有集成的环境中安装集群。如果在裸机上安装，您需要提供所有集群基础架构和资源，包括网络、负载均衡、存储和所有集群机器。
- 要使用基于代理的安装程序部署集群，您可以首先下载基于代理的安装程序。然后，您可以配置集群并生成发现镜像。您可以使用发现镜像引导集群机器，它会安装一个与安装程序进行通信的代理，并为您处理置备，您不需要与安装程序进行交互或自行设置置备程序机器。您需要提供所有集群基础架构和资源，包括网络、负载均衡、存储和单个集群机器。这个方法适用于断开连接的环境。
- 对于具有安装程序置备的基础架构集群，您可以将基础架构启动和置备委派给安装程序，而不是亲自执行。安装程序将创建支持集群所需的所有网络、机器和操作系统，除非您载裸机上安装。如果在裸机上安装，您必须提供所有集群基础架构和资源，包括 bootstrap 机器、网络、负载均衡、存储和单个集群机器。
- 如果亲自为集群置备和管理基础架构，则必须提供所有集群基础架构和资源，包括 Bootstrap 机器、网络、负载均衡、存储和独立的集群机器。

对于安装程序，在安装过程中会使用三组文件：名为 **install-config.yaml** 的安装配置文件、Kubernetes 清单，以及您的集群类型的 Ignition 配置文件。



重要

在安装过程中，您可以修改控制基础 RHCOS 操作系统的 Kubernetes 和 Ignition 配置文件。但是，没有可用的验证机制来确认您对这些对象所做修改是适当的。如果修改了这些对象，集群可能会无法运行。由于存在这种风险，修改 Kubernetes 和 Ignition 配置文件不受支持，除非您遵循记录的流程或在红帽支持指示下操作。

安装配置文件转换为 Kubernetes 清单，然后清单嵌套到 Ignition 配置文件中。安装程序使用这些 Ignition 配置文件来创建集群。

运行安装程序时，所有配置文件会被修剪，因此请务必备份需要再次使用的所有配置文件。

重要

安装之后，您无法修改在安装过程中设置的参数，但可以修改一些集群属性。

使用辅助安装程序的安装过程
使用辅助安装程序进行安装涉及使用基于 Web 的用户界面或使用 RESTful API 以互动方式创建集群配置。Assisted Installer 用户界面会提示您输入所需的值，并为其余参数提供合理的默认值，除非在用户界面或使用 API 中更改它们。Assisted Installer 生成发现镜像，您可以下载并用来引导集群机器。镜像安装 RHCOS 和代理，代理会为您处理置备。您可以使用 Assisted Installer 安装 OpenShift Container Platform，并在 Nutanix、vSphere 和裸机上完全集成。另外，您可以在其他没有集成的情况下使用 Assisted Installer 安装 OpenShift Container Platform。

OpenShift Container Platform 管理集群的所有方面，包括操作系统本身。每台机器在启动时使用的配置引用其加入的集群中托管的资源。此配置允许集群在应用更新时自行管理。

如果可能，请使用 Assisted Installer 功能来避免下载和配置基于代理的安装程序。

基于代理的基础架构的安装过程
基于代理的安装与使用 Assisted Installer 类似，唯一的不同是需要在最初下载并安装基于代理的安装程序。当您希望利用 Assisted Installer 所带来的变量，并需要在断开连接的环境中安装集群时，可以使用基于代理的安装。

如果可能，请使用基于代理的安装功能来避免创建带有 bootstrap 虚拟机的置备程序机器，然后置备和维护集群基础架构。

采用安装程序置备的基础架构的安装过程
默认安装类型为使用安装程序置备的基础架构。默认情况下，安装程序充当安装向导，提示您输入它无法自行确定的值，并为其余参数提供合理的默认值。您还可以自定义安装过程来支持高级基础架构场景。安装程序将为集群置备底层基础架构。

您可以安装标准集群或自定义集群。对于标准集群，您要提供安装集群所需的最低限度详细信息。对于自定义集群，您可以指定有关平台的更多详细信息，如 control plane 使用的机器数量、集群部署的虚拟机的类型，或 Kubernetes 服务网络的 CIDR 范围。

若有可能，可以使用此功能来避免置备和维护集群基础架构。在所有其他环境中，可以使用安装程序来生成置备集群基础架构所需的资产。

对于安装程序置备的基础架构的集群，OpenShift Container Platform 可以管理集群的所有方面，包括操作系统本身。每台机器在启动时使用的配置引用其加入的集群中托管的资源。此配置允许集群在应用更新时自行管理。

采用用户置备的基础架构的安装过程
您还可以在自己提供的基础架构上安装 OpenShift Container Platform。您可以使用安装程序来生成置备集群基础架构所需的资产，再创建集群基础架构，然后将集群部署到您提供的基础架构中。

如果不使用安装程序置备的基础架构，您必须自己管理和维护集群资源。以下列表详细介绍了其中一些自我管理的资源：

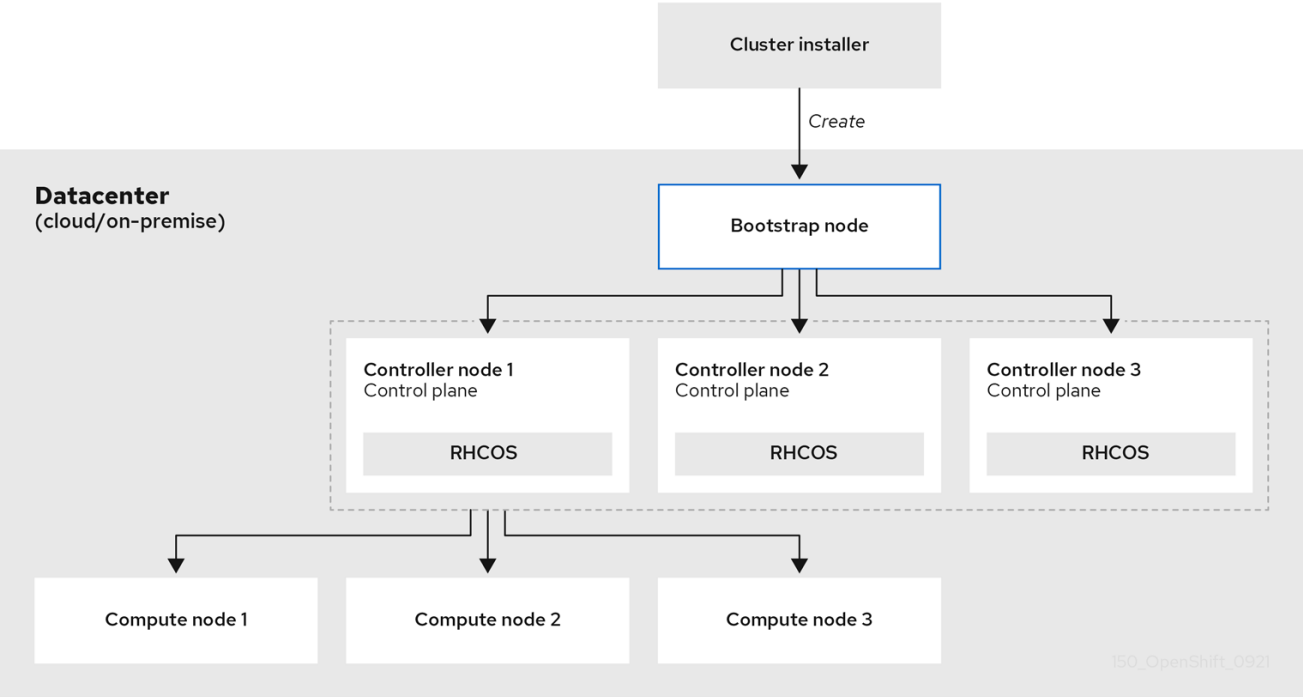
- 组成集群的 control plane 和计算机器的底层基础架构
- 负载均衡器
- 集群网络，包括 DNS 记录和所需的子网
- 集群基础架构和应用程序的存储

如果您的集群使用用户置备的基础架构，您可以选择将 RHEL 计算机器添加到集群中。

安装过程详细信息

置备集群时，集群中的每台机器都需要有关集群的信息。OpenShift Container Platform 在初始配置过程中使用临时 bootstrap 机器为永久 control plane 提供所需的信息。临时 bootstrap 机器使用一个带有描述如何创建集群的 Ignition 配置文件进行引导。bootstrap 机器创建组成控制平面（control plane）的 control plane 机器。然后，control plane 机器创建计算（compute）机器。下图说明了这一过程：

图 3.2. 创建 bootstrap、control plane 和计算机器



集群机器初始化后，Bootstrap 机器将被销毁。所有集群都使用 Bootstrap 过程来初始化集群，但若您自己置备集群的基础架构，则必须手动完成许多步骤。

重要

- 安装程序生成的 Ignition 配置文件包含在 24 小时后过期的证书，然后在过期时进行续订。如果在更新证书前关闭集群，且集群在 24 小时后重启，集群会自动恢复过期的证书。一个例外是，您必须手动批准待处理的 **node-bootstrapper** 证书签名请求(CSR)来恢复 kubelet 证书。如需更多信息，*请参阅从过期的 control plane 证书* 中恢复的文档。
- 建议您在 Ignition 配置文件生成后的 12 小时内使用它们，因为 24 小时的证书会在集群安装后的 16 小时到 22 小时间进行轮转。通过在 12 小时内使用 Ignition 配置文件，您可以避免在安装过程中因为执行了证书更新而导致安装失败的问题。

bootstrapp 集群涉及以下步骤：

1. bootstrap 机器启动并开始托管 control plane 机器引导所需的远程资源。如果您置备基础架构，此步骤需要人工干预。
2. bootstrap 机器启动单节点 etcd 集群和一个临时 Kubernetes control plane。
3. control plane 机器从 bootstrap 机器获取远程资源并完成启动。如果您置备基础架构，此步骤需要人工干预。
4. 临时 control plane 将生产环境的 control plane 调度到生产环境 control plane 机器。

5. Cluster Version Operator（CVO）在线并安装 etcd Operator。etcd Operator 在所有 control plane 节点上扩展 etcd。
6. 临时 control plane 关机，并将控制权交给生产环境 control plane。
7. bootstrap 机器将 OpenShift Container Platform 组件注入生产环境 control plane。
8. 安装程序关闭 bootstrap 机器。如果您置备基础架构，此步骤需要人工干预。
9. control plane 设置计算节点。
10. control plane 以一组 Operator 的形式安装其他服务。

完成此 bootstrap 过程后，将生成一个全面运作的 OpenShift Container Platform 集群。然后，集群会下载并配置日常运作所需的其余组件，包括在受支持的环境中创建计算（compute）机器。

安装范围

OpenShift Container Platform 安装程序的作用范围特意设计得比较狭窄。它旨在简化操作并确保成功。安装完成后，您可以完成更多的配置任务。

其他资源

- 如需有关 OpenShift Container Platform 配置资源的详细信息，请参阅可用的集群自定义。

3.2. 关于 OPENSHIFT UPDATE 服务

OpenShift Update Service (OSUS) 为 OpenShift Container Platform 提供推荐的更新，包括 Red Hat Enterprise Linux CoreOS (RHCOS)。它提供了一个图表，其中包含组件 Operator 的*顶点（vertices）*和连接它们的 *边（edges）*。图中的边代表了您可以安全更新到的版本。顶点是更新的有效负载，用于指定受管集群组件的预期状态。

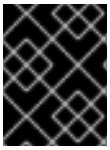
集群中的 Cluster Version Operator (CVO) 会检查 OpenShift Container Platform 更新服务，并根据当前组件版本和图中的信息决定有效的更新和更新路径。当您请求更新时，CVO 使用对应的发行镜像来更新集群。发行工件 (artifact) 作为容器镜像托管在 Quay 中。

为了让 OpenShift Update Service 仅提供兼容的更新，可以使用一个版本验证管道来驱动自动化过程。每个发行工件都会被验证是否与支持的云平台和系统架构以及其他组件包兼容。在管道确认有适用的版本后，OpenShift Update Service 会通知您它可用。

重要

OpenShift Update Service 显示当前集群的所有推荐更新。如果 OpenShift Update Service 不建议更新路径，这可能是因为一个与更新路径相关的已知问题，如不兼容或可用性。

两个控制器在持续更新模式下运行。第一个控制器持续更新有效负载清单，将清单应用到集群，并输出 Operator 的受控推出的状态，以指示它们是否处于可用、升级或失败状态。第二个控制器轮询 OpenShift Update Service，以确定更新是否可用。



重要

仅支持更新到较新的版本。不支持将集群还原或回滚到以前的版本。如果您的更新失败，请联系红帽支持。

在更新过程中，Machine Config Operator(MCO)将新配置应用到集群机器。MCO 会处理由 **maxUnavailable** 字段指定的、协调机器配置池中的节点数量，并将它们标记为不可用。在默认情况下，这个值被设置为 **1**。MCO 根据 **topology.kubernetes.io/zone** 标签，按区字母更新受影响的节点。如果一个区域有多个节点，则首先更新最旧的节点。对于不使用区的节点，如裸机部署中的节点，节点会按使用的时间更新，首先更新最旧的节点。MCO 一次更新机器配置池中由 **maxUnavailable** 字段指定的节点数量。然后，MCO 会应用新配置并重启机器。

> **警告**
>
> 对于 OpenShift Container Platform 中的所有机器配置池，**maxUnavailable** 的默认设置是 **1**。建议您不要更改这个值，且一次只更新一个 control plane 节点。对于 control plane 池，请不要将这个值改为 **3**。

如果您将 Red Hat Enterprise Linux (RHEL) 机器用作 worker，MCO 不会在这些机器上更新 kubelet，因为您必须首先在这些机器上更新 OpenShift API。

当新版本规格应用到旧的 kubelet 时，RHEL 机器无法返回 **Ready** 状态。在机器可用前，您无法完成更新。但是，因为已设置了最大不可用节点数，所以可以在一定机器无法使用的情况下，确保正常的集群操作。

OpenShift Update Service 由 Operator 和一个或多个应用程序实例组成。

3.3. 非受管 OPERATOR 的支持策略

Operator 的 *管理状态* 决定了一个 Operator 是否按设计积极管理集群中其相关组件的资源。如果 Operator 设置为 *非受管（unmanaged）* 状态，它不会响应配置更改，也不会收到更新。

虽然它可以在非生产环境集群或调试过程中使用，但处于非受管状态的 Operator 不被正式支持，集群管理员需要完全掌控各个组件的配置和升级。

可使用以下方法将 Operator 设置为非受管状态：

- **独立 Operator 配置**
  独立 Operator 的配置中具有 **managementState** 参数。这可以通过不同的方法来访问，具体取决于 Operator。例如，Red Hat OpenShift Logging Operator 通过修改它管理的自定义资源（CR）来达到此目的，而 Cluster Samples Operator 使用了集群范围配置资源。

  将 **managementState** 参数更改为 **Unmanaged** 意味着 Operator 不会主动管理它的资源，也不会执行与相关组件相关的操作。一些 Operator 可能不支持此管理状态，因为它可能会损坏集群，需要手动恢复。

  > **警告**
  >
  > 将独立 Operator 更改为**非受管**状态会导致不支持该特定组件和功能。报告的问题必须在 **受管（Managed）** 状态中可以重复出现才能继续获得支持。

- **Cluster Version Operator (CVO) 覆盖**
  可将 **spec.overrides** 参数添加到 CVO 配置中，以便管理员提供对组件的 CVO 行为覆盖的列表。将一个组件的 **spec.overrides[].unmanaged** 参数设置为 **true** 会阻止集群升级并在设置 CVO 覆盖后提醒管理员：

  > Disabling ownership via cluster version overrides prevents upgrades. Please remove overrides before continuing.

  警告

  设置 CVO 覆盖会使整个集群处于不受支持状态。在删除所有覆盖后，必须可以重现报告的问题方可获得支持。

3.4. 后续步骤

- 选择集群安装方法并为用户准备它

第 4 章 RED HAT OPENSHIFT CLUSTER MANAGER

Red Hat OpenShift Cluster Manager 是一个受管服务，您可以安装、修改、操作和升级 Red Hat OpenShift 集群。此服务允许您通过单一仪表板处理机构的所有集群。

OpenShift Cluster Manager 指导您在 AWS(ROSA)和 OpenShift Dedicated 集群上安装 OpenShift Container Platform、Red Hat OpenShift Service。它还负责管理安装后的 OpenShift Container Platform 集群，以及您的 ROSA 和 OpenShift Dedicated 集群。

您可以使用 OpenShift Cluster Manager 来执行以下操作：

- 创建新集群
- 查看集群详情和指标
- 使用扩展、更改节点标签、网络、身份验证等任务管理集群
- 管理访问控制
- 监控集群
- 调度升级

4.1. 访问 RED HAT OPENSHIFT CLUSTER MANAGER

您可以使用配置的 OpenShift 帐户访问 OpenShift Cluster Manager。

先决条件

- 您有一个帐户，该帐户属于 OpenShift 组织。
- 如果要创建集群，您的机构指定了配额。

流程

- 使用您的登录凭据登录 OpenShift Cluster Manager。

4.2. 常规操作

在集群页面的顶部，用户可以在整个集群中执行一些操作：

- **打开控制台**会启动一个 web 控制台，以便集群所有者可以向集群发出命令。
- **操作**下拉菜单允许集群所有者重命名集群的显示名称，更改集群中的负载均衡器和持久性存储的数量（如果适用），手动设置节点数并删除集群。
- **刷新**图标会强制刷新集群。

4.3. 集群标签页

选择一个活跃的、安装的集群会显示与该集群关联的标签页。集群安装完成后显示以下标签页：

- 概述
- Access control

- 附加组件
- 网络
- Insights 公告
- 机器池
- 支持
- 设置

4.3.1. 概述标签

Overview选项卡提供有关如何配置集群的信息：

- **集群 ID**是创建集群的唯一标识符。此 ID 可用于从命令行向集群发出命令。
- **Type** 显示集群正在使用的 OpenShift 版本。
- **region** 是服务器区域。
- **Provider** 显示集群构建了哪些云供应商。
- **Availability** 显示集群使用的可用区类型，可以是单个或者多区。
- **Version** 是集群中安装的 OpenShift 版本。如果有可用的更新，您可以从此字段更新。
- **Created at** 显示集群创建的日期和时间。
- **Owner** 标识创建集群并具有所有者权限。
- **Subscription type** 显示创建时选择的订阅模式。
- **Infrastructure type** 是集群使用的帐户类型。
- **Status** 显示集群的当前状态。
- **Total vCPU** 显示此集群可用虚拟 CPU 总数。
- **Total memory** 显示此集群可用内存总量。
- 负载**均衡器**
- **Persistent storage** 显示此集群中可用的存储量。
- **Nodes** 显示集群上的实际和所需节点。这些数字可能会因为集群扩展而不匹配。
- **Network** 字段显示网络连接的地址和前缀。
- 选项卡的 **Resource usage** 部分显示图形中使用的资源。
- **Advisor recommendations** 部分提供与安全性、性能、可用性和稳定性相关的见解。本节需要使用远程健康功能。请参阅*附加资源*部分中的*使用 Insights 识别集群中的问题*。
- **Cluster history** 部分显示集群完成的所有内容，包括创建和确定新版本时。

4.3.2. 访问控制标签页

Access control 选项卡允许集群所有者设置身份提供程序，授予升级的权限，并为其他用户授予角色。

先决条件

- 您必须是集群所有者，或具有适当权限才能授予集群上的角色。

流程

1. 选择 **授予角色** 按钮。
2. 为您希望授予集群角色的用户输入红帽帐户登录。
3. 选择对话框中的**授予角色**按钮。
4. 对话框关闭，所选用户会显示"Cluster Editor"访问。

4.3.3. 附加组件标签页

Add-ons 选项卡显示可添加到集群中的所有可选附加组件。选择所需的附加组件，然后在显示的附加组件下面选择 **Install**。

4.3.4. Insights 公告标签页

Insights Advisor 选项卡使用 OpenShift Container Platform 的 Remote Health 功能来识别和缓解安全性、性能、可用性和稳定性风险。请参阅 OpenShift Container Platform 文档中的使用 Insights 发现集群中的问题。

4.3.5. 机器池标签

如果有足够的可用配额或编辑现有机器池，**Machine pool** 选项卡允许集群所有者创建新的机器池。

选择 **More options** > **Scale** 打开 "Edit node count" 对话框。在此对话框中，您可以更改每个可用区的节点数。如果启用了自动扩展，您也可以为自动扩展设置范围。

4.3.6. 支持标签

在 **Support** 选项卡中，您可以为应接收集群通知的个人添加通知联系人。您提供的用户名或电子邮件地址必须与部署了集群的红帽机构的用户帐户相关。

另外，在这个选项卡中，您可以创建一个支持问题单来请求集群的技术支持。

4.3.7. 设置标签页

Settings 选项卡为集群所有者提供几个选项：

- **Monitoring** （默认启用）允许报告用户定义的操作。请参阅了解监控堆栈。
- 通过**更新策略**，您可以确定集群是否在指定时间的某一天上自动更新，或者是否可以手动调度所有更新。
- **节点排空** 会设置在更新过程中遵守保护工作负载的持续时间。当传递此持续时间时，节点会被强制删除。

- **更新状**态显示当前的版本，以及是否有可用的更新。

4.4. 其他资源

- 如需 OpenShift Cluster Manager 的完整文档，请参阅 OpenShift Cluster Manager 文档。

第 5 章 关于 KUBERNETES OPERATOR 的多集群引擎

扩展 Kubernetes 环境的一个挑战是管理不断增加的系统的生命周期。要满足这一挑战，您可以使用多集群引擎。Operator 为受管 OpenShift Container Platform 集群提供完整的生命周期功能，并为其他 Kubernetes 发行版本提供部分生命周期管理。它有两种方法：

- 作为一个独立的 operator，它作为 OpenShift Container Platform 或 OpenShift Kubernetes Engine 订阅的一部分安装
- 作为 Red Hat Advanced Cluster Management for Kubernetes 的一部分

5.1. 在 OPENSHIFT CONTAINER PLATFORM 中使用多集群引擎进行集群管理

在 OpenShift Container Platform 上启用多集群引擎时，您可以获得以下功能：

- 托管 control planes，它是基于 HyperShift 项目的功能。使用集中托管的 control plane，您可以以超大规模的方式运行 OpenShift Container Platform 集群。
- Hive，为 hub 置备自我管理的 OpenShift Container Platform 集群，并为这些集群完成初始配置。
- klusterlet 代理，将受管集群注册到 hub。
- 基础架构 Operator，管理辅助服务的部署，以编配 OpenShift Container Platform 的内部裸机和 vSphere 安装，如裸机上的单节点 OpenShift。Infrastructure Operator 包括 GitOps 零接触置备 (ZTP)，它通过 GitOps 工作流在裸机和 vSphere 置备上完全自动化集群创建，以管理部署和配置更改。
- 打开集群管理，它提供管理 Kubernetes 集群的资源。

多集群引擎包含在 OpenShift Container Platform 支持订阅中，它与内核有效负载分开交付。要开始使用多集群引擎，请部署 OpenShift Container Platform 集群，然后安装 Operator。如需更多信息，请参阅安装和升级多集群引擎 operator。

5.2. 使用 RED HAT ADVANCED CLUSTER MANAGEMENT 进行集群管理

如果您需要在带有多集群引擎的 OpenShift Container Platform 之外提供集群管理功能，请考虑 Red Hat Advanced Cluster Management。多集群引擎是 Red Hat Advanced Cluster Management 的一个完整部分，默认是启用的。

5.3. 其他资源

有关多集群引擎的完整文档，请参阅带有多集群引擎的集群生命周期文档，它是 Red Hat Advanced Cluster Management 产品文档的一部分。

第 6 章 CONTROL PLANE 架构

control plane 由 control plane 机器组成，负责管理 OpenShift Container Platform 集群。control plane 机器管理计算机器（也被称为 worker）上的工作负载。集群本身通过Cluster Version Operator (CVO)、Machine Config Operator 和一组单独 Operator 的操作来管理对机器的所有升级。

6.1. 使用机器配置池进行节点配置管理

运行 control plane 组件或用户工作负载的机器会根据其处理的资源类型划分为组。这些机器组称为机器配置池（MCP）。每个 MCP 管理一组节点及其对应的机器配置。节点的角色决定了它所属的 MCP ; MCP 会根据其分配的节点角色标签管理节点。MCP 中的节点具有相同的配置 ; 这意味着节点可以扩展并缩减，以适应增加或降低的工作负载。

默认情况下，在安装时集群创建两个 MCP：**master** 和 **worker**。每个默认 MCP 都有一个定义的配置，由 Machine Config Operator（MCO）应用，该配置负责管理 MCP 有助于 MCP 升级。您可以创建额外的 MCP 或自定义池来管理具有超出默认节点类型的自定义用例的节点。

自定义池是从 worker 池中继承其配置的池。它们将任何机器配置用于 worker 池，但添加了仅针对自定义池部署更改的能力。由于自定义池从 worker 池继承其配置，对 worker 池的任何更改都会应用到自定义池。MCO 不支持从 worker 池中继承其配置的自定义池。

注意

节点只能包含在一个 MCP 中。如果节点有多个与多个 MCP 对应的标签，如 **worker,infra**，它由 infra 自定义池而不是 worker 池管理。自定义池根据节点标签在选择节点时具有优先权。不属于自定义池的节点由 worker 池管理。

建议您为集群中要管理的每个节点角色创建一个自定义池。例如，如果您创建 infra 节点来处理 infra 工作负载，建议创建一个自定义 infra MCP 将那些节点分组在一起。如果您将 **infra** 角色标签应用到 worker 节点，使其具有 **workerinfra dual** 标签，但没有自定义 infra MCP，则 MCO 认为它是一个 worker 节点。如果您从节点中删除 **worker** 标签，并应用 **infra** 标签而不将其分组到自定义池中，则该节点不可被 MCO 识别，且不受集群管理。

重要

任何使用 **infra** 角色标记的节点如果只运行 infra 工作负载，则不计算到订阅总数中。管理 infra 节点的 MCP 与集群决定订阅的方式相互排斥 ; 为节点添加适当 **infra** 角色的标签，并使用污点以防止用户工作负载调度到该节点上，这是避免为 infra 工作负载添加订阅的唯一要求。

MCO 独立应用池更新。例如，如果有会影响所有池的更新，则每个池更新中的节点会相互并行。如果您添加自定义池，则该池中的节点还会尝试与 master 和 worker 节点同时更新。

在某些情况下，节点上的配置与当前应用的机器配置指定不完全匹配。这个状态被称为 *配置偏移*。Machine Config Daemon(MCD)定期检查节点是否有配置偏移。如果 MCD 检测到配置偏移，MCO 会将节点标记为 **降级(degraded)**，直到管理员更正节点配置。降级的节点在线且可操作，但无法更新。

其他资源

- 了解配置偏移检测

6.2. OPENSHIFT CONTAINER PLATFORM 中的机器角色

OpenShift Container Platform 为主机分配不同的角色。这些角色定义机器在集群内的功能。集群包含标准 **master** 和 **worker** 角色类型的定义。

注意

集群还包含 **bootstrap** 角色的定义。由于 bootstrap 机器仅在集群安装期间使用，因此其功能在集群安装文档中阐述。

6.2.1. control plane 和节点主机兼容性

OpenShift Container Platform 版本必须与 control plane 主机和节点主机间的匹配。例如，在一个 4.16 集群中，所有 control plane 主机都必须是 4.16，所有节点也必需是 4.16。

在集群升级过程中出现临时的不匹配可以被接受。例如，当从以前的 OpenShift Container Platform 版本升级到 4.16 时，一些节点会升级到 4.16。因为较长的 control plane 主机和节点主机偏移可能会使旧的计算机器存在错误并缺失一些功能。用户应该尽快解决偏移的 control plane 主机和节点主机。

kubelet 服务不能比 **kube-apiserver** 更新，并根据您的 OpenShift Container Platform 版本号是否是奇数还是偶数，最多可以早于两个次版本。下表显示了适当的版本兼容性：

OpenShift Container Platform 版本	支持的 kubelet 偏移
奇数的 OpenShift Container Platform 次版本 [1]	最多早于一个版本
偶数 OpenShift Container Platform 次版本 [2]	最多早于两个版本

1. 例如，OpenShift Container Platform 4.11, 4.13。
2. 例如，OpenShift Container Platform 4.10、4.12。

6.2.2. 集群 worker

在 Kubernetes 集群中，worker 节点运行和管理 Kubernetes 用户请求的实际工作负载。worker 节点公告其容量，以及 control plane 服务调度程序，决定在哪些节点上启动 pod 和容器。以下重要服务在每个 worker 节点上运行：

- CRI-O，即容器引擎。
- kubelet，这是接受并履行运行和停止容器工作负载的请求的服务。
- 服务代理，用于管理跨 worker 的 pod 的通信。
- runC 或 crun 低级别容器运行时，用于创建和运行容器。

注意

有关如何启用 crun 而不是默认的 runC 的详情，请参考创建 **ContainerRuntimeConfig** CR 的文档。

在 OpenShift Container Platform 中，计算机器控制分配了 **worker** 机器的计算机器。具有 **worker** 角色的机器驱动计算工作负载，这些工作负载由自动扩展它们的特定机器池管理。由于 OpenShift Container Platform 具有支持多个机器类型的能力，所以具有 **worker** 角色的机器被归类为 *compute* 机器。在本发

行版本中，*worker 机器*和 *计算机器*是可以被互换使用的术语，因为计算机器的唯一默认类型是 worker 机器。在未来的 OpenShift Container Platform 版本中，默认情况下可能会使用不同类型的计算机器，如基础架构机器。

注意

计算机器集是 **machine-api** 命名空间下的计算机器资源分组。计算机器集是设计在特定云供应商上启动新计算机器的配置。相反，机器配置池（MCP）是 Machine Config Operator（MCO）命名空间的一部分。MCP 用于将机器分组在一起，以便 MCO 能够管理其配置并便于升级。

6.2.3. 集群 control plane

在 Kubernetes 集群中，*master* 节点运行控制 Kubernetes 集群所需的服务。在 OpenShift Container Platform 中，control plane 由具有 **master** 机器角色的 control plane 机器组成。它们不仅仅包含用于管理 OpenShift Container Platform 集群的 Kubernetes 服务。

对于大多数 OpenShift Container Platform 集群，control plane 机器由一系列独立的机器 API 资源定义。对于支持的云供应商和 OpenShift Container Platform 版本组合，control plane 可使用 control plane 机器集管理。额外的控件应用到 control plane 机器，以防止您删除所有 control plane 机器并破坏集群。

注意

所有生产部署都必须使用三个 control plane 节点。

control plane 上属于 Kubernetes 类别的服务包括 Kubernetes API 服务器、etcd、Kubernetes 控制器管理器和 Kubernetes 调度程序。

表 6.1. 在 control plane 上运行的 Kubernetes 服务

组件	描述
Kubernetes API 服务器	Kubernetes API 服务器验证并配置 pod、服务和复制控制器的数据。它还为集群的共享状态提供了一个焦点。
etcd	etcd 存储持久 control plane 状态，其他组件则监视 etcd 的更改，以使其自身进入指定状态。
Kubernetes 控制器管理器	Kubernetes 控制器管理器监视 etcd 是否有对象的更改，如复制、命名空间，务帐户控制器对象，然后使用 API 来强制实施指定的状态。多个这样的过程会创建在某个时间点上有一个活跃群首的集群。
Kubernetes 调度程序	Kubernetes 调度程序监视没有分配节点的新创建的 pod，并选择托管该 pod 的最佳节点。

另外，在 control plane 上运行的 OpenShift 服务包括 OpenShift API 服务器、OpenShift 控制器管理器、OpenShift OAuth API 服务器和 OpenShift OAuth 服务器。

表 6.2. 在 control plane 上运行的 OpenShift 服务

组件	描述
OpenShift API 服务器	OpenShift API 服务器验证并配置 OpenShift 资源（如项目、路由和模板）的数据。 OpenShift API 服务器由 OpenShift API Server Operator 管理。
OpenShift 控制器管理器	OpenShift 控制器管理器监视 etcd 是否有 OpenShift 对象的更改，如项目、路由和模板控制器对象，然后使用 API 来强制实施指定的状态。 OpenShift 控制器管理器由 OpenShift Controller Manager Operator 管理。
OpenShift OAuth API 服务器	OpenShift OAuth API 服务器验证并配置数据以向 OpenShift Container Platform 进行身份验证，如用户、组和 OAuth 令牌。 OpenShift OAuth API 服务器由 Cluster Authentication Operator 管理。
OpenShift OAuth 服务器	用户从 OpenShift OAuth 服务器请求令牌，以向 API 进行身份验证。 OpenShift OAuth 服务器由 Cluster Authentication Operator 管理。

control plane 机器上的某些服务作为 systemd 服务运行，另一些则作为静态 pod 运行。

systemd 服务适合需要始终在特定系统启动后不久出现的服务。对于 control plane 机器，这包括允许远程登录的 sshd。它还包括以下服务：

- CRI-O 容器引擎 (crio)，用于运行和管理容器。OpenShift Container Platform 4.16 使用 CRI-O，而不是 Docker Container Engine。
- kubelet (kubelet)，从 control plane 服务接受管理机器上容器的请求。

CRI-O 和 Kubelet 必须作为 systemd 服务直接在主机上运行，因为它们必须先运行，然后您才能运行其他容器。

installer-* 和 **revision-pruner-*** control plane pod 必须使用 root 权限运行，因为它们需要写入属于 root 用户的 **/etc/kubernetes** 目录。这些 pod 位于以下命名空间中：

- **openshift-etcd**
- **openshift-kube-apiserver**
- **openshift-kube-controller-manager**
- **openshift-kube-scheduler**

6.3. OPENSHIFT CONTAINER PLATFORM 中的 OPERATOR

Operator 是 OpenShift Container Platform 中最重要的组件。Operator 是 control plane 上打包、部署和管理服务的首选方法。它们还可以为用户运行的应用程序提供优势。

Operator 与 Kubernetes API 和 CLI 工具（如 **kubectl** 和 **oc** 命令）集成。它们提供了监控应用程序、执行健康检查、管理无线(OTA)更新的方法，并确保应用程序保持在指定的状态。

Operator 还提供了更为精细的配置体验。若要配置各个组件，您可以修改 Operator 公开的 API，而不必修改全局配置文件。

因为 CRI-O 和 Kubelet 在每个节点上运行，所以几乎所有其他集群功能都可以通过使用 Operator 在 control plane 上进行管理。使用 Operator 添加到 control plane 的组件包括重要的网络服务和凭证服务。

虽然这两个操作都遵循类似的 Operator 概念和目标，但 OpenShift Container Platform 中的 Operator 由两个不同的系统管理，具体取决于其用途：

- 由 Cluster Version Operator (CVO) 管理的 Cluster Operator 被默认安装来执行集群功能。
- 可选的附加组件 Operator 由 Operator Lifecycle Manager(OLM)管理，供用户在其应用程序中运行。

6.3.1. Cluster Operators

在 OpenShift Container Platform 中，所有集群功能都划分到一系列默认 *集群 Operator* 中。集群 Operator 管理集群功能的特定方面，如集群范围的应用程序日志记录、Kubernetes control plane 管理或机器置备系统。

集群 Operator 由一个 **ClusterOperator** 对象表示，集群管理员可在 OpenShift Container Platform Web 控制台的 **Administration → Cluster Settings** 页面中查看它。每个集群 Operator 都会提供一个确定集群功能的简单 API。Operator 将管理组件生命周期的细节隐藏起来。Operator 可以管理一个组件或数十个组件，但最终目标始终是通过自动化常见操作来减轻运维负担。

其他资源

- 集群 Operator 参考

6.3.2. 附加组件 Operator

Operator Lifecycle Manager(OLM)和 OperatorHub 是 OpenShift Container Platform 中的默认组件，可帮助将 Kubernetes 原生应用程序作为 Operator 进行管理。它们一起提供用于发现、安装和管理集群中可用的可选附加组件 Operator 的系统。

在 OpenShift Container Platform Web 控制台中使用 OperatorHub，集群管理员和授权用户可以选择从 Operator 目录安装的 Operator。在从 OperatorHub 安装 Operator 后，可以以全局的方式或针对特定命名空间，在用户应用程序中运行。

提供默认目录源，其中包括 Red Hat Operator、经过认证的 Operator 和社区 Operator。集群管理员也可以添加自己的自定义目录源，这些源可以包含一组自定义的 Operator。

开发人员可以使用 Operator SDK 来帮助编写利用 OLM 功能的自定义 Operator。然后，可以将它们的 Operator 捆绑并添加到自定义目录源中，该源可以添加到集群中，并可供用户使用。

注意

OLM 不管理组成 OpenShift Container Platform 架构的集群 Operator。

其他资源

- 如需有关在 OpenShift Container Platform 中运行附加组件 Operator 的更多详细信息，请参阅 Operator Lifecycle Manager(OLM) 和 OperatorHub 中的 *Operator* 指南部分。
- 如需有关 Operator SDK 的更多信息，请参阅开发 Operator。

6.4. 关于 MACHINE CONFIG OPERATOR

OpenShift Container Platform 4.16 集成了操作系统和集群管理。由于集群管理自己的更新，包括集群节点上 Red Hat Enterprise Linux CoreOS (RHCOS) 的更新，因此 OpenShift Container Platform 提供了可靠的生命周期管理体验，能够简化节点升级的编配。

OpenShift Container Platform 使用三个守护进程集和控制器来简化节点管理。这些守护进程集通过使用标准的 Kubernetes 式构造来编配操作系统更新和主机配置更改。它们包括：

- **machine-config-controller**，协调从 control plane 进行的机器升级。它监控所有集群节点并编配其配置更新。
- **machine-config-daemon** 守护进程集在集群中的每个节点上运行，并根据 MachineConfigController 的指示将机器更新为机器配置定义的配置。 当节点检测到更改时，它会排空其 pod，应用更新并重启。这些更改以 Ignition 配置文件的形式出现，这些文件应用指定的机器配置并控制 kubelet 配置。更新本身在容器中交付。此过程是成功管理 OpenShift Container Platform 和 RHCOS 更新的关键。
- **machine-config-server** 守护进程集，在加入集群时为 control plane 节点提供 Ignition 配置文件。

机器配置是 Ignition 配置的子集。**machine-config-daemon** 读取机器配置，以查看是否需要进行 OSTree 更新，或者是否必须应用一系列 systemd kubelet 文件更改、配置更改，或者对操作系统或 OpenShift Container Platform 配置的其他更改。

执行节点管理操作时，您可以创建或修改 **KubeletConfig** 自定义资源(CR)。

重要

当对机器配置进行修改时，Machine Config Operator（MCO） 会自动重启所有对应的节点，以使更改生效。

您可以使用节点中断策略缓解某些机器配置更改造成的中断。请参阅*了解机器配置更改后节点重启行为*。

或者，您可以在进行更改前防止节点在机器配置更改后自动重启。通过在对应的机器配置池中将 **spec.paused** 字段设置为 **true** 来暂停自动引导过程。暂停后，机器配置更改不会生效，除非将 **spec.paused** 字段设置为 **false**，且节点已重启至新配置。

以下修改不会触发节点重新引导：

- 当 MCO 检测到以下任何更改时，它会在不排空或重启节点的情况下应用更新：
  - 在机器配置的 **spec.config.passwd.users.sshAuthorizedKeys** 参数中更改 SSH 密钥。
  - 在 **openshift-config** 命名空间中更改全局 pull secret 或 pull secret。
  - Kubernetes API Server Operator 自动轮转 **/etc/kubernetes/kubelet-ca.crt** 证书颁发机构（CA）。
- 当 MCO 检测到对 **/etc/containers/registries.conf** 文件的更改时，如添加或编辑 **ImageDigestMirrorSet**、**ImageTagMirrorSet** 或 **ImageContentSourcePolicy** 对象，它会排空对应的节点，应用更改并取消记录节点。对于以下更改，节点排空不会发生：
  - 增加了一个 registry，带有为每个镜像（mirror）设置了 **pull-from-mirror = "digest-only"** 参数。
  - 增加了一个镜像（mirror），带有在一个 registry 中设置的 **pull-from-mirror = "digest-only"** 参数。
  - 在 **unqualified-search-registries** 列表中添加项目。

在某些情况下，节点上的配置与当前应用的机器配置指定不完全匹配。这个状态被称为 *配置偏移*。Machine Config Daemon(MCD)定期检查节点是否有配置偏移。如果 MCD 检测到配置偏移，MCO 会将节点标记为 **降级(degraded)**，直到管理员更正节点配置。降级的节点在线且可操作，但无法更新。

其他资源

- 有关检测配置偏移的更多信息，请参阅了解配置偏移检测。
- 如需有关在 Machine Config Operator 更改机器配置后防止 control plane 机器重启的信息，请参阅禁用 Machine Config Operator 自动重新引导。
- 了解机器配置更改后节点重启行为。

6.5. ETCD 概述

etcd 是一个一致的、分布式键值存储，它包含较小的数据，可以完全包括在内存中。虽然 etcd 是许多项目的核心组件，但它是 Kubernetes 的主要数据存储，但这是容器编配的标准系统。

6.5.1. 使用 etcd 的好处

通过使用 etcd，您可以以多种方式获益：

- 保持云原生应用程序的一致性运行时间，并在单个服务器失败的情况下保持工作
- 为 Kubernetes 存储和复制所有集群状态
- 分发配置数据，以便为配置节点提供冗余和弹性

6.5.2. etcd 的工作原理

为确保集群配置和管理的可靠方法，etcd 使用 etcd Operator。Operator 简化了在 OpenShift Container Platform 等 Kubernetes 容器平台中使用 etcd。使用 etcd Operator，您可以创建和删除 etcd 成员、重新定义集群大小、执行备份和升级 etcd。

etcd Operator 会观察、分析和操作：

1. 它使用 Kubernetes API 观察集群状态。
2. 它分析了当前状态和您希望的状态之间的区别。
3. 它通过 etcd 集群管理 API，Kubernetes API 或两者解决了两者的不同。

etcd 会维持集群状态，它会持续更新。这个状态会被持续保留，这会导致高频率的、大量的小更改。因此，使用快速、低延迟 I/O 支持 etcd 集群成员至关重要。有关 etcd 的最佳实践的更多信息，请参阅"推荐 etcd 实践"。

其他资源

- 推荐的 etcd 实践
- 备份 etcd

6.6. 托管 CONTROL PLANE 简介

您可以使用 Red Hat OpenShift Container Platform 托管 control plane 来降低管理成本，优化集群部署时间，并分离管理和工作负载问题，以便专注于应用程序。

通过使用以下平台上的 multicluster engine for Kubernetes operator 版本 2.0 或更高版本 提供托管的 control plane：

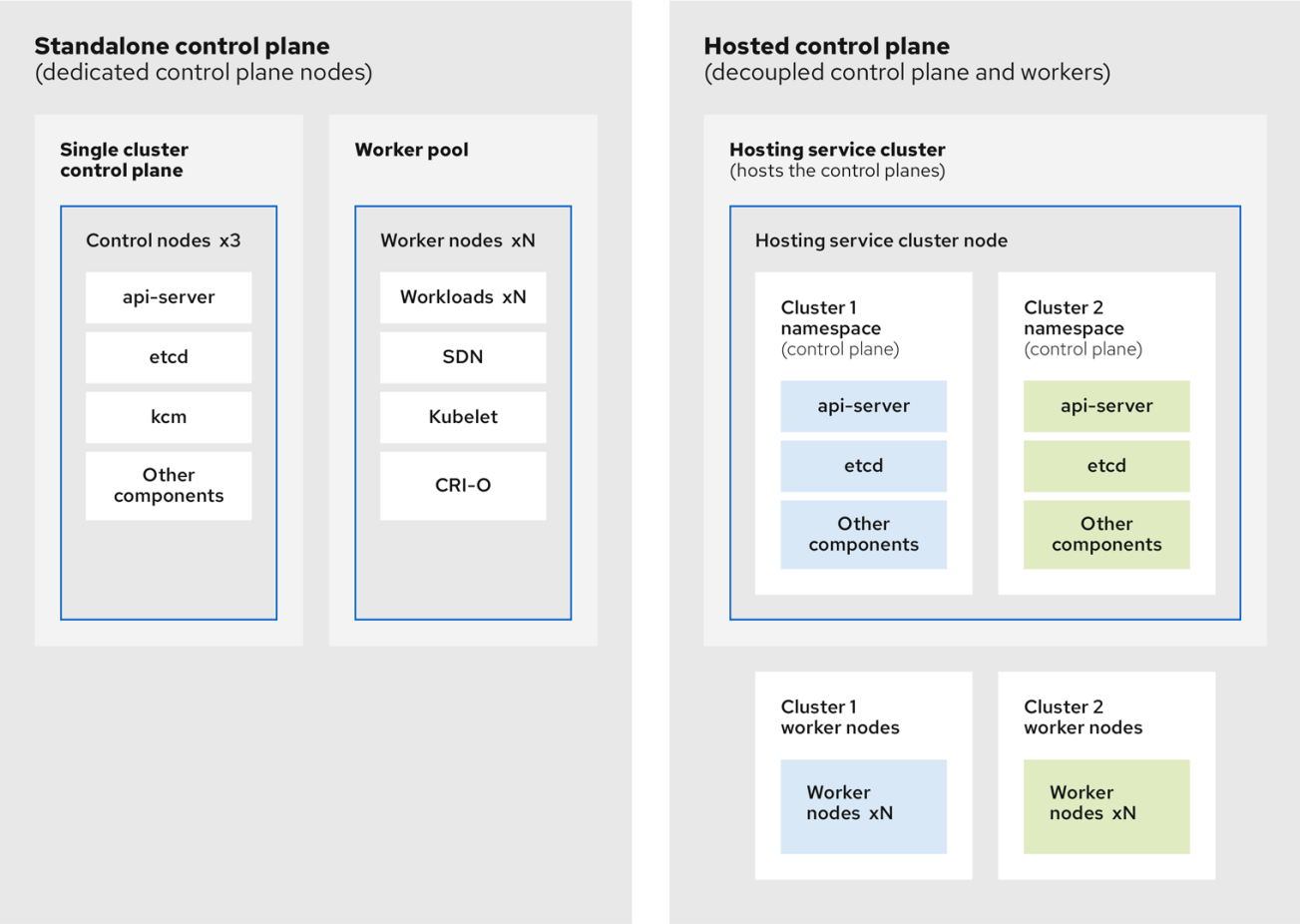
- 使用 Agent 供应商进行裸机
- OpenShift Virtualization 作为连接的环境中正式发布的功能，以及断开连接的环境中的技术预览功能
- Amazon Web Services (AWS) 为技术预览功能
- IBM Z 作为技术预览
- IBM Power，一个技术预览功能

6.6.1. 托管 control plane 的架构

OpenShift Container Platform 通常以组合或独立部署，集群由 control plane 和数据平面组成。control plane 包括 API 端点、存储端点、工作负载调度程序和确保状态的指示器。data plane 包括运行工作负载的计算、存储和网络。

独立的 control plane 由一组专用的节点（可以是物理或虚拟）托管，最小数字来确保仲裁数。网络堆栈被共享。对集群的管理员访问权限提供了对集群的 control plane、机器管理 API 和有助于对集群状态贡献的其他组件的可见性。

虽然独立模式运行良好，但在某些情况下需要与 control plane 和数据平面分离的架构。在这些情况下，data plane 位于带有专用物理托管环境的独立网络域中。control plane 使用 Kubernetes 原生的高级别原语（如部署和有状态集）托管。control plane 被视为其他工作负载。



6.6.2. 托管 control plane 的优点

使用托管 OpenShift Container Platform 的 control plane，您可以为真正的混合云方法打下基础，并享受一些其他优势。

- 管理和工作负载之间的安全界限很强大，因为 control plane 分离并在专用的托管服务集群中托管。因此，您无法将集群的凭证泄漏到其他用户。因为基础架构 secret 帐户管理也已被分离，所以集群基础架构管理员无法意外删除 control plane 基础架构。
- 使用托管 control plane，您可以在较少的节点上运行多个 control plane。因此，集群更为经济。
- 因为 control plane 由 OpenShift Container Platform 上启动的 pod 组成，所以 control planes 快速启动。同样的原则适用于 control plane 和工作负载，如监控、日志记录和自动扩展。
- 从基础架构的角度来看，您可以将 registry、HAProxy、集群监控、存储节点和其他基础架构组件推送到租户的云供应商帐户，将使用情况隔离到租户。

- 从操作的角度来看，多集群管理更为集中，从而减少了影响集群状态和一致性的外部因素。站点可靠性工程师具有调试问题并进入集群的数据平面的中心位置，这可能会导致更短的时间解析 (TTR) 并提高生产效率。

其他资源

- 托管 control plane

6.6.3. 托管控制平面（control plane）的常见概念和用户角色表

当使用托管的 control plane 用于 OpenShift Container Platform 时，了解其关键概念和涉及的用户角色非常重要。

6.6.3.1. 概念

托管的集群

一个 OpenShift Container Platform 集群，其控制平面和 API 端点托管在管理集群中。托管的集群包括控制平面和它的对应的数据平面。

托管的集群基础架构

存在于租户或最终用户云账户中的网络、计算和存储资源。

托管控制平面

在管理集群上运行的 OpenShift Container Platform 控制平面，它由托管集群的 API 端点公开。控制平面的组件包括 etcd、Kubernetes API 服务器、Kubernetes 控制器管理器和 VPN。

托管集群

请参阅 *管理集群*。

受管集群

hub 集群管理的集群。此术语特定于在 Red Hat Advanced Cluster Management 中管理 Kubernetes Operator 的多集群引擎的集群生命周期。受管集群（managed cluster）与 *管理集群（management cluster）*不同。如需更多信息，请参阅管理的集群。

管理集群

部署 HyperShift Operator，以及用于托管集群的控制平面所在的 OpenShift Container Platform 集群。管理集群与 *托管集群（hosting cluster）*是同义的。

管理集群基础架构

管理集群的网络、计算和存储资源。

节点池

包含计算节点的资源。control plane 包含节点池。计算节点运行应用程序和工作负载。

6.6.3.2. Personas

集群实例管理员

假设此角色的用户等同于独立 OpenShift Container Platform 中的管理员。此用户在置备的集群中具有 **cluster-admin** 角色，但可能无法在更新或配置集群时关闭。此用户可能具有只读访问权限，来查看投射到集群中的一些配置。

集群实例用户

假设此角色的用户等同于独立 OpenShift Container Platform 中的开发人员。此用户没有 OperatorHub 或机器的视图。

集群服务消费者

假设此角色的用户可以请求控制平面和 worker 节点，驱动更新或修改外部化配置。通常，此用户无法管理或访问云凭证或基础架构加密密钥。集群服务消费者人员可以请求托管集群并与节点池交互。假设此角色的用户具有在逻辑边界中创建、读取、更新或删除托管集群和节点池的用户。

集群服务提供商

假设此角色的用户通常具有管理集群上的 **cluster-admin** 角色，并具有 RBAC 来监控并拥有 HyperShift Operator 的可用性，以及租户托管的集群的 control plane。集群服务提供商用户角色负责多个活动，包括以下示例：

- 拥有服务级别的对象，用于实现控制平面可用性、正常运行时间和稳定性。
- 为管理集群配置云帐户以托管控制平面
- 配置用户置备的基础架构，其中包括主机对可用计算资源的了解

6.6.4. 托管 control plane 的版本控制

对于 OpenShift Container Platform 的每个主要、次版本或补丁版本，会发布两个托管的 control plane 组件：

- HyperShift Operator
- **hcp** 命令行界面 (CLI)

HyperShift Operator 管理由 **HostedCluster** API 资源表示的托管集群的生命周期。HyperShift Operator 会随每个 OpenShift Container Platform 发行版本一起发布。HyperShift Operator 在 **hypershift** 命名空间中创建 **supported-versions** 配置映射。配置映射包含受支持的托管集群版本。

您可以在同一管理集群中托管不同版本的 control plane。

supported-versions 配置映射对象示例

```
apiVersion: v1
data:
  supported-versions: '{"versions":["4.16"]}'
kind: ConfigMap
metadata:
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
```

您可以使用 **hcp** CLI 创建托管集群。

您可以使用 **hypershift.openshift.io** API 资源，如 **HostedCluster** 和 **NodePool**，以大规模创建和管理 OpenShift Container Platform 集群。**HostedCluster** 资源包含 control plane 和通用数据平面配置。当您创建 **HostedCluster** 资源时，您有一个完全正常工作的 control plane，没有附加的节点。 **NodePool** 资源是一组可扩展的 worker 节点，附加到 **HostedCluster** 资源。

API 版本策略通常与 Kubernetes API 版本 的策略一致。

第 7 章 NVIDIA GPU 架构概述

NVIDIA 支持在 OpenShift Container Platform 上使用图形处理单元 (GPU) 资源。OpenShift Container Platform 是一个以安全为中心的、强化的 Kubernetes 平台，由红帽开发并提供支持，用于大规模部署和管理 Kubernetes 集群。OpenShift Container Platform 包括对 Kubernetes 的增强，以便用户可以轻松地配置和使用 NVIDIA GPU 资源来加快工作负载。

NVIDIA GPU Operator 利用 OpenShift Container Platform 中的 Operator 框架来管理运行 GPU 加速工作负载所需的 NVIDIA 软件组件的完整生命周期。

这些组件包括 NVIDIA 驱动程序（为了启用 CUDA）、GPU 的 Kubernetes 设备插件、NVID Container Toolkit、使用 GPU 特性发现(GFD)、基于 DCGM 的监控等的自动节点标记。

注意

NVIDIA GPU Operator 的支持仅由 NVIDIA 提供。有关从 NVIDIA 获取支持的更多信息，请参阅 NVIDIA 支持。

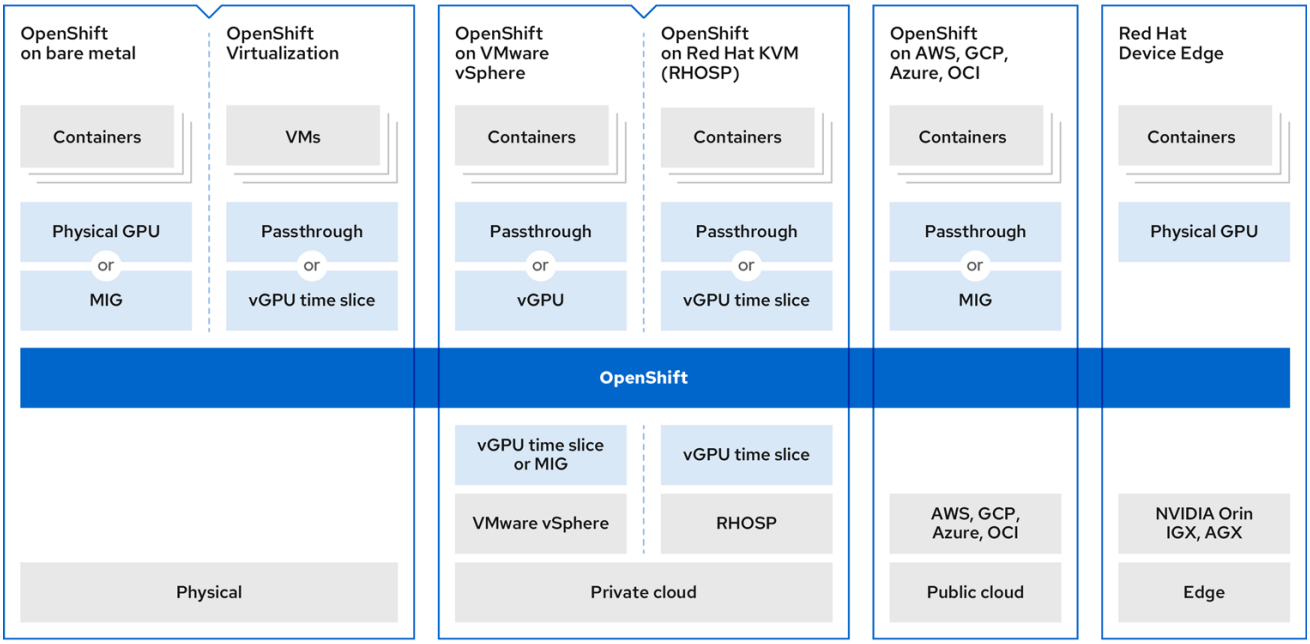
7.1. NVIDIA GPU 先决条件

- 包括至少一个 GPU worker 节点的，可正常工作的 OpenShift 集群。
- 以 **cluster-admin** 身份访问 OpenShift 集群，以执行必要的步骤。
- 已安装 OpenShift CLI (**oc**)。
- 已安装节点功能发现 (NFD) Operator 并创建了 **nodefeaturediscovery** 实例。

7.2. NVIDIA GPU 启用

下图显示了如何为 OpenShift 启用 GPU 架构：

图 7.1. NVIDIA GPU 启用



注意

MIG 仅支持 A30, A100, A100X, A800, AX800, H100, 和 H800。

7.2.1. GPU 和裸机

您可以在 NVIDIA 认证的裸机服务器上部署 OpenShift Container Platform，但有一些限制：

- control plane 节点可以是 CPU 节点。
- Worker 节点必须是 GPU 节点，只要 AI/ML 工作负载在这些 worker 节点上执行。另外，worker 节点可以托管一个或多个 GPU，但它们必须是相同的类型。例如，一个节点可以有两个 NVIDIA A100 GPU，但不支持在一个节点中带有一个 A100 GPU 和一个 T4 GPU。Kubernetes 的 NVIDIA 设备插件不支持在同一节点上混合不同的 GPU 模型。
- 在使用 OpenShift 时，请注意，需要一个、三个或更多个服务器。不支持带有两个服务器的集群。单一服务器部署称为单一节点 openShift (SNO)，使用此配置的 OpenShift 环境不具有高可用性。

您可以选择以下方法之一来访问容器化 GPU：

- GPU passthrough（GPU 透传）
- 多实例 GPU (MIG)

其他资源

- Red Hat OpenShift on Bare Metal Stack

7.2.2. GPU 和虚拟化

虽然许多开发人员和企业都在转型到容器化应用程序和无服务器基础架构，但仍然有大量对在虚拟机(VM)上运行的应用程序进行开发和维护的需求。Red Hat OpenShift Virtualization 提供此功能，使企业能够将虚拟机合并到集群中的容器化工作流中。

您可以选择以下方法之一将 worker 节点连接到 GPU：

- 用于访问和使用虚拟机(VM)中的 GPU 硬件的 GPU 透传。
- 当 GPU 计算的容量没有因为工作负载而饱和时，可以进行 GPU (vGPU) 时间分片。

其他资源

- 带有 OpenShift Virtualization 的 NVIDIA GPU Operator

7.2.3. GPU 和 vSphere

您可以在可托管不同 GPU 类型的 NVIDIA 认证的 VMware vSphere 服务器上部署 OpenShift Container Platform。

如果虚拟机使用了 vGPU 实例，必须在 hypervisor 中安装 NVIDIA GPU 驱动程序。对于 VMware vSphere，此主机驱动程序以 VIB 文件的形式提供。

可分配给 worker 节点虚拟机的最大 vGPU 数量取决于 vSphere 的版本：

- vSphere 7.0：每个虚拟机最多 4 个 vGPU

- vSphere 8.0：每个虚拟机最大 8 个 vGPU

> **注意**
>
> vSphere 8.0 引入了对与一个虚拟机关联的多个完整或部分同配置集的支持。

您可以选择以下方法之一将 worker 节点附加到 GPU：

- 用于访问和使用虚拟机(VM)中的 GPU 硬件的 GPU 透传
- 当不需要所有 GPU 时，可以使用 GPU (vGPU) 时间分片

与裸机部署类似，需要一个或多个服务器。不支持带有两个服务器的集群。

其他资源

- 带有 NVIDIA vGPU 的 VMware vSphere 上的 OpenShift Container Platform

7.2.4. GPU 和 Red Hat KVM

您可以在基于 NVIDIA 认证的虚拟机 (KVM) 服务器上使用 OpenShift Container Platform。

与裸机部署类似，需要一个或多个服务器。不支持带有两个服务器的集群。

但是，与裸机部署不同，您可以在服务器中使用不同类型的 GPU。这是因为您可以将这些 GPU 分配给作为 Kubernetes 节点的不同虚拟机。唯一的限制是，一个 Kubernetes 节点在自己本身上必须具有相同的 GPU 类型。

您可以选择以下方法之一来访问容器化 GPU：

- 用于访问和使用虚拟机(VM)中的 GPU 硬件的 GPU 透传
- 当不需要所有 GPU 时，可以使用 GPU (vGPU) 时间分片

要启用 vGPU 功能，必须在主机级别安装特殊驱动程序。这个驱动程序作为 RPM 软件包提供。对于 GPU 透传分配，不需要这个主机驱动程序。

其他资源

- 如何在 KVM 上部署 OpenShift Container Platform 4.13

7.2.5. GPU 和 CSP

您可以将 OpenShift Container Platform 部署到主要的云服务供应商 (CSP) 之一：Amazon Web Services (AWS)、Google Cloud Platform (GCP)或 Microsoft Azure。

有两种操作模式：完全管理的部署和自我管理的部署。

- 在完全管理的部署中，一切都由红帽与 CSP 合作实现自动化。您可以通过 CSP Web 控制台请求 OpenShift 实例，集群由红帽自动创建并完全管理。您不必担心环境中节点故障或错误。红帽完全负责维护集群的正常运行时间。完全管理的服务在 AWS 和 Azure 上提供。对于 AWS，OpenShift 服务称为 ROSA (Red Hat OpenShift Service on AWS)。对于 Azure，该服务称为 Azure Red Hat OpenShift。

- 在自我管理的部署中，您需要自行实例化和维护 OpenShift 集群。红帽提供了 OpenShift-install 工具来支持 OpenShift 集群的部署。自我管理的服务可全局提供给所有 CSP。

重要的是，此计算实例是一个 GPU 加速的计算实例，并且 GPU 类型与 NVIDIA AI Enterprise 支持的 GPU 列表匹配。例如，T4、V100 和 A100 是此列表的一部分。

您可以选择以下方法之一来访问容器化 GPU：

- 用于访问和使用虚拟机(VM)中的 GPU 硬件的 GPU 透传。
- 当不需要整个 GPU 时，可以进行 GPU (vGPU) 时间分片。

其他资源

- 云中的 Red Hat Openshift

7.2.6. GPU 和 Red Hat Device Edge

Red Hat Device Edge 提供对 MicroShift 的访问。MicroShift 提供了单节点部署的简单性和资源约束（边缘）计算所需的功能和服务。Red Hat Device Edge 满足在资源受限环境中部署的裸机、虚拟、容器化或 Kubernetes 工作负载的需求。

您可以在 Red Hat Device Edge 环境中的容器上启用 NVIDIA GPU。

您可以使用 GPU 透传来访问容器化 GPU。

其他资源

- 如何在 Red Hat Device Edge 中使用 NVIDIA GPU 加速工作负载

7.3. GPU 共享方法

红帽和 NVIDIA 已开发了 GPU 并发和共享机制，以简化企业级 OpenShift Container Platform 集群上的 GPU 加速计算。

应用程序通常会有不同的计算要求，这可能会使 GPU 使用率不足。为每个工作负载提供正确的计算资源数量对于降低部署成本和最大化 GPU 使用率至关重要。

用于提高 GPU 使用率的并发机制，范围从编程模型 API 到系统软件和硬件分区，包括虚拟化。以下列表显示了 GPU 并发机制：

- 计算统一设备架构 (CUDA) 流
- Time-slicing（时间分片）
- CUDA 多进程服务 (MPS)
- 多实例 GPU (MIG)
- 使用 vGPU 的虚拟化

在将 GPU 并发机制用于不同的 OpenShift Container Platform 场景时，请考虑以下 GPU 共享建议：

裸机

vGPU 不可用。考虑使用支持 MIG 的卡。

虚拟机

vGPU 是最佳选择。

裸机中的较旧的 NVIDIA 卡中没有 MIG

考虑使用 time-slicing。

具有多个 GPU 的虚拟机，您需要透传和 vGPU

考虑使用单独的虚拟机。

使用 OpenShift Virtualization 和多个 GPU 的裸机

对于托管的虚拟机，考虑使用透传，对呀容器，考虑使用时间分片。

其他资源

- 提高 GPU 利用率

7.3.1. CUDA 流

Compute Unified Device Architecture (CUDA) 是由 NVIDIA 开发的并行计算平台和编程模型，用于 GPU 上的常规计算。

流是 GPU 上问题顺序执行的操作序列。CUDA 命令通常在默认流中按顺序执行，任务在前面的任务完成后才会启动。

跨不同流的异步处理操作允许并行执行任务。在一个流中发布的任务之前、期间或另一个任务签发到另一个流后运行。这允许 GPU 以任何规定的顺序同时运行多个任务，从而提高性能。

其他资源

- 异步并发执行

7.3.2. Time-slicing（时间分片）

当您运行多个 CUDA 应用程序时，在超载 GPU 上调度 GPU 时间的交集工作负载。

您可以通过为 GPU 定义一组副本来启用 Kubernetes 上的 GPU 的时间，每个副本都可以独立分发到 pod 来运行工作负载。与多实例 GPU (MIG) 不同，在副本之间不存在内存或故障隔离，但对于某些工作负载，这优于根本不进行共享。在内部，GPU 时间分片用于从同一底层 GPU 将多个工作负载分配到不同的副本。

对于时间分片，您可以应用一个集群范围的默认配置。您还可以应用特定于节点的配置。例如，您只能将时间分片配置应用到具有 Tesla T4 GPU 的节点，而不能将节点改为使用其他 GPU 模型。

您可以通过应用集群范围的默认配置来组合这两种方法，然后标记节点以授予这些节点接收特定于节点的配置。

7.3.3. CUDA 多进程服务

CUDA 多进程服务 (MPS) 允许单个 GPU 使用多个 CUDA 进程。进程在 GPU 上并行运行，消除了 GPU 计算资源的饱和。MPS 还支持并发执行或重叠内核操作和从不同进程复制的内存，以增强利用率。

其他资源

- CUDA MPS

7.3.4. 多实例 GPU

使用多实例 GPU (MIG)，您可以将 GPU 计算单元和内存分成多个 MIG 实例。每个实例都代表了从系统的角度来看的独立 GPU 设备，并可连接到节点上运行的任何应用程序、容器或虚拟机。使用 GPU 的软件将每个 MIG 实例视为单独的 GPU。

当您有一个不需要整个 GPU 完全功能的应用程序时，MIG 很有用。新的 NVIDIA Ampere 架构的 MIG 功能允许您将硬件资源分成多个 GPU 实例，每个实例都可作为独立的 CUDA GPU 提供给操作系统。

NVIDIA GPU Operator 版本 1.7.0 及更高版本为 A100 和 A30 Ampere 卡提供 MIG 支持。这些 GPU 实例旨在支持最多 7 个独立的 CUDA 应用程序，以便它们与专用硬件资源完全隔离。

其他资源

- NVIDIA 多实例 GPU 用户指南

7.3.5. 使用 vGPU 的虚拟化

虚拟机 (VM) 可以使用 NVIDIA vGPU 直接访问单个物理 GPU。您可以创建虚拟 GPU，供虚拟机在企业之间共享，并由其他设备访问。

此功能将 GPU 性能与 vGPU 提供的管理和安全优势相结合。vGPU 提供的其他好处包括虚拟机环境的主动管理和监控、混合 VDI 和计算工作负载的工作负载平衡以及多个虚拟机之间的资源共享。

其他资源

- 虚拟 GPU

7.4. OPENSHIFT CONTAINER PLATFORM 的 NVIDIA GPU 功能

NVIDIA Container Toolkit

NVIDIA Container Toolkit 可让您创建并运行 GPU 加速容器。工具包包括容器运行时库和工具，用于自动配置容器以使用 NVIDIA GPU。

NVIDIA AI Enterprise

NVIDIA AI Enterprise 是端到端的云原生 AI 和数据分析软件套件，由 NVIDIA 认证系统进行优化、认证和支持。

NVIDIA AI Enterprise 包括对 Red Hat OpenShift Container Platform 的支持。支持以下安装方法：

- 带有 GPU Passthrough 的裸机或 VMware vSphere 上的 OpenShift Container Platform。
- 带有 NVIDIA vGPU 的 VMware vSphere 上的 OpenShift Container Platform。

GPU 功能发现

NVIDIA GPU Feature Discovery for Kubernetes 是一个软件组件，可让您为节点上可用的 GPU 自动生成标签。GPU 功能发现使用节点功能发现(NFD)来执行此标记。

Node Feature Discovery Operator (NFD)通过使用硬件特定信息标记节点来管理 OpenShift Container Platform 集群中硬件功能和配置的发现。NFD 使用特定于节点的属性标记主机，如 PCI 卡、内核、操作系统版本等。

您可以通过搜索 "Node Feature Discovery" 在 Operator Hub 中找到 NFD Operator。

带有 OpenShift Virtualization 的 NVIDIA GPU Operator

到目前为止，GPU Operator 只置备了 worker 节点来运行 GPU 加速的容器。现在，GPU Operator 也可以用来置备 worker 节点来运行 GPU 加速的虚拟机 (VM)。
您可以根据将 GPU 工作负载配置为在这些节点上运行，将 GPU Operator 配置为将不同的软件组件部署到 worker 节点。

GPU 监控仪表板

您可以安装监控仪表板，在 OpenShift Container Platform Web 控制台的集群 **Observe** 页面中显示 GPU 用量信息。GPU 使用率信息包括可用 GPU 数、功耗（watts）、温度（Celsius）、利用率（百分比）以及其他每个 GPU 的指标。

其他资源

- NVIDIA 认证系统
- NVIDIA AI Enterprise
- NVIDIA Container Toolkit
- 启用 GPU 监控仪表板
- OpenShift Container Platform 中的 MIG 支持
- OpenShift 中的 NVIDIA GPU 时间分片
- 在断开连接的或 airgapped 环境中部署 GPU Operator
- Node Feature Discovery Operator

第 8 章 了解 OPENSHIFT CONTAINER PLATFORM 开发

为了在开发和运行企业级品质应用程序时充分利用容器的功能，请确保您的环境受相应工具的支持，让容器能够：

- 创建为离散的微服务，可以连接到其他容器化和非容器化服务。例如，您可能希望将应用程序与数据库衔接，或将监控应用程序附加到数据库。
- 具有弹性，因此在服务器崩溃或需要停机维护或退役时，容器可以在另一台机器上启动。
- 实现自动化，以自动获取代码更改，然后启动和部署自身的新版本。
- 得以扩展或复制，在需求增加时为客户端提供更多实例，在需求下降时缩减为更少的实例。
- 以不同的方式运行，具体由应用程序的类型决定。例如，一个应用程序可能每月运行一次来生成报告，然后退出。另一个应用程序可能需要持续运行，并且必须对客户端高度可用。
- 受到管理，以便您可以监视应用程序的状态并在出现问题时做出反应。

容器得到广泛接受，对能让容器适合企业使用的工具和方法的需求也随之诞生，这使得容器有了丰富的选择。

本节的其余部分介绍在 OpenShift Container Platform 中构建和部署容器化 Kubernetes 应用程序时可以选择创建的资产。它还说明您可以采用哪些方法来满足不同类型的应用程序和开发需求。

8.1. 关于容器化应用程序开发

您可以通过多种方式使用容器来进行应用程序开发，每种方法更适合的使用情景也各不相同。为了说明这种多样性，本文在介绍一系列方法时首先从开发单个容器开始，最后将容器部署为面向大型企业的任务关键型应用程序。这些方法展示了不同的工具、格式和方法，供您用于容器化应用程序开发。本主题描述：

- 构建一个简单容器并将其存储在 registry 中
- 创建 Kubernetes 清单并将其保存到 Git 存储库
- 使 Operator 能够与其他项共享您的应用程序

8.2. 构建一个简单容器

您对应用程序有了一个想法，想要对其进行容器化。

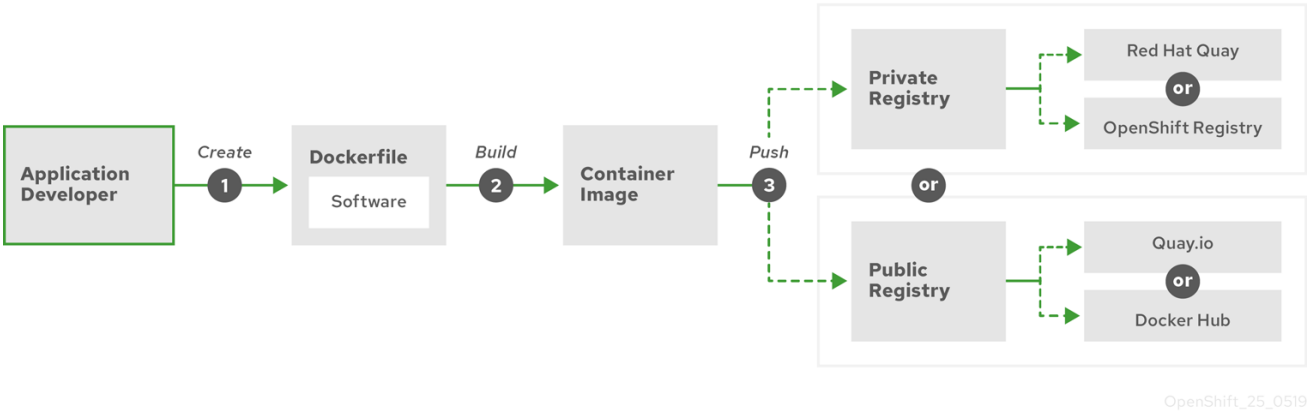
首先，您需要一个用于构建容器的工具，如 buildah 或 docker，还需要一个描述容器中内容的文件，通常是 Dockerfile。

接下来，您需要一个位置来推送生成的容器镜像，以便可以将它拉取到您想要它运行的位置。这个位置就是容器 registry。

大多数 Linux 操作系统上都会默认安装各个组件的一些示例，但 Dockerfile 除外，需要您自己提供。

下图显示了构建和推送镜像的流程：

图 8.1. 创建简单容器化应用程序并将其推送到 registry



如果您使用运行 Red Hat Enterprise Linux (RHEL) 作为操作系统的计算机，则容器化应用程序创建过程需要以下几个步骤：

1. 安装容器构建工具：RHEL 包含一组工具，其中包括用于构建和管理容器的 podman、buildah 和 skopeo。
2. 创建一个 Dockerfile 来组合基础镜像和软件：有关构建容器的信息存放在名为 **Dockerfile** 的文件中。在这个文件中，您要标识从中构建的基本镜像、要安装的软件包，以及要复制到容器中的软件。您还要标识参数值，如公开到容器外部的网络端口和挂载到容器内的卷。将您的 Dockerfile 和您要容器化的软件放到 RHEL 系统上的目录中。
3. 运行 buildah 或 docker 构建：运行 **buildah build-using-dockerfile** 或 **docker build** 命令，将您选择的基础镜像拉取到本地系统，再创建一个存储在本地的容器镜像。 您还可以使用 buildah，在不用 Dockerfile 的前提下构建容器镜像。
4. 标记（tag）并推送到 registry：向新容器镜像添加标签 (tag)，以标识要在其中存储和共享容器的 registry 位置。然后，通过运行 **podman push** 或 **docker push** 命令将该镜像推送到 registry。
5. 拉取并运行镜像：从具有 podman 或 docker 等容器客户端工具的任何系统，运行用于标识新镜像的命令。例如，运行 **podman run <image_name>** 或 **docker run <image_name>** 命令。其中，**<image_name>** 是新容器镜像的名称，类似于 **quay.io/myrepo/myapp:latest**。registry 可能需要凭证才能推送和拉取镜像。

如需更多有关构建容器镜像、将其推送到 registry 并运行它们的过程的详细信息，请参阅使用 Buildah 自定义镜像构建。

8.2.1. 容器构建工具选项

使用 buildah、podman 和 skopeo 构建和管理容器会导致行业标准容器镜像中包含专门为在 OpenShift Container Platform 或其他 Kubernetes 环境中部署容器而调整的功能。这些工具是无守护进程的，可在没有 root 权限的情况下运行，运行它们所需的开销更少。

重要

Kubernetes 1.20 中弃用了对 Docker Container Engine 作为容器运行时的支持，并将在以后的发行版本中删除。但是，Docker 生成的镜像将继续在集群中使用所有运行时，包括 CRI-O。如需更多信息，请参阅 Kubernetes 博客公告。

最终在 OpenShift Container Platform 中运行容器时，您要使用 CRI-O 容器引擎。CRI-O 在 OpenShift Container Platform 集群中的每一 worker 和 control plane 机器上运行，但 CRI-O 尚不支持作为 OpenShift Container Platform 外的独立运行时。

8.2.2. 基础镜像选项

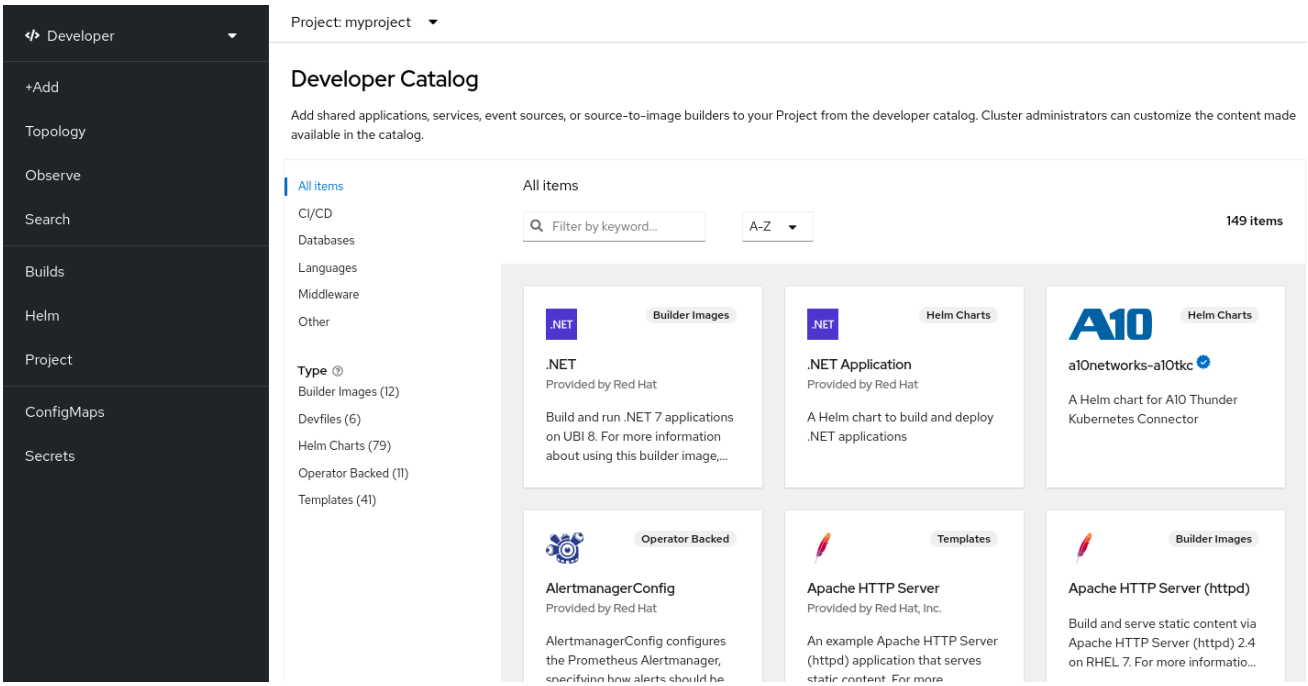
选择用来构建应用程序的基础镜像包含一组软件，这些软件为应用程序提供一个 Linux 系统。在您构建自己的镜像时，您的软件将放置到该文件系统中，可以像对待操作系统一样看待该文件系统。基础镜像的选择会对容器未来的安全性、效率和可升级性产生重大影响。

红帽提供了一组新的基础镜像，称为红帽通用基础镜像 (UBI)。这些镜像基于 Red Hat Enterprise Linux，与红帽过去提供的基础镜像相似，但有一个主要区别：无需红帽订阅就能自由重新分发。因此，您可以在 UBI 镜像上构建应用程序，不必担心如何共享它们或需要为不同的环境创建不同的镜像。

这些 UBI 镜像具有标准、初始和最低版本。您还可以将 Red Hat Software Collections 镜像用作依赖特定运行时环境（如 Node.js、Perl 或 Python）的应用程序的基础。其中一些运行时基础镜像的特殊版本称为 Source-to-Image (S2I) 镜像。使用 S2I 镜像时，您可以将代码插入到可随时运行该代码的基础镜像环境中。

S2I 镜像可用于直接从 OpenShift Container Platform Web UI 中使用。在 Developer 视角中，进入到 **+Add** 视图，并在 **Developer Catalog** 标题中查看 Developer Catalog 中的所有可用服务。

图 8.2. 为需要特定运行时的应用选择 S2I 基础镜像



8.2.3. Registry 选项

容器镜像仓库（registry）是存储容器镜像的位置，以便您可以与他人共享，并将它们提供给最终运行的平台。您可以选择提供免费帐户的大型公共容器 registry，也可选择提供更多存储和特殊功能的高级版本。您还可以安装自己的 registry，供您的组织专用或有选择地共享给他人。

要获取红帽和认证合作伙伴提供的镜像，您可以从 Red Hat Registry 中提取。Red Hat Registry 存在于两个位置：**registry.access.redhat.com**（无需身份验证，但已弃用）和 **registry.redhat.io**（需要身份验证）。您可以在红帽生态系统目录的容器镜像部分了解 Red Hat Registry 中的红帽及合作伙伴的镜像。除了列出红帽容器镜像外，它还显示有关这些镜像的内容和质量的广泛信息，包括基于已应用安全更新的健康分数。

大型公共 registry 包括 Docker Hub 和 Quay.io。Quay.io registry由红帽所有和管理。OpenShift Container Platform 中使用的许多组件都存储在 Quay.io 中，包括用于部署 OpenShift Container Platform 本身的容器镜像和 Operator。Quay.io 还提供了存储其他类型内容的方法，包括 Helm Charts。

如果需要私有容器 registry，OpenShift Container Platform 本身包括一个私有容器 registry，随 OpenShift Container Platform 一起安装并在其集群上运行。红帽也提供 Quay.io registry 的私有版本，称为 Red Hat Quay。Red Hat Quay 包括跨地域复制、Git 构建触发器、Clair 镜像扫描和许多其他功能。

此处提到的所有 registry 都可能需要凭证才能从中下载镜像。这些凭证中有些是通过 OpenShift Container Platform 在整个集群范围内提供的，另一些凭证则可以分配给个人。

8.3. 为 OPENSHIFT CONTAINER PLATFORM 创建 KUBERNETES 清单

尽管容器镜像是容器化应用程序的基本构建块，但需要更多信息才能在 Kubernetes 环境（如 OpenShift Container Platform）中管理和部署该应用程序。创建镜像后的典型后续步骤：

- 了解 Kubernetes 清单中使用的不同资源
- 就您将运行的应用程序做出一些决策
- 收集支持组件
- 创建清单并将该清单存储到 Git 存储库中，以便您可以将其存储在源版本控制系统中，对其进行审核、跟踪和升级，并将其部署到下一环境中，在必要时回滚到旧版本，以及与他人共享

8.3.1. 关于 Kubernetes pod 和服务

容器镜像是 docker 的基本单元，而 Kubernetes 使用的基本单元称为 pods。Pod 代表构建应用程序的下一步。pod 可以包含一个或多个容器。关键之处在于 pod 是您部署、扩展和管理的单个单元。

在决定 pod 中要放入的内容时，可扩展性和命名空间或许是要考虑的主要项目。为便于部署，您可能需要将容器部署到 pod 中，并在 pod 中包含其本身的日志记录和监控容器。以后，在您运行 pod 并需要扩展额外的实例时，其他那些容器也会随之扩展。对于命名空间，pod 中的容器共享相同的网络接口、共享存储卷和资源限制（如内存和 CPU）。这样一来，将 pod 内容作为一个单元进行管理变得更加轻松。pod 中的容器还可以使用标准的进程间通信（如 System V 信号或 POSIX 共享内存）相互通信。

虽然单个 pod 代表 Kubernetes 中的一个可扩展单元，但服务提供了一个途径，能够将一系列 pod 分组到一起以创造完整且稳定的应用程序，完成诸如负载均衡之类的任务。 服务也比 pod 更持久，因为服务可以一直从同一 IP 地址使用，直到您删除为止。在使用服务时，可通过名称来请求服务，OpenShift Container Platform 集群则将该名称解析为您可用于访问构成该服务的 pod 的 IP地址和端口。

从本质上讲，容器化应用程序与运行它们的操作系统是隔开的，进而与其用户隔开。Kubernetes 清单的一部分描述了如何通过定义网络策略将应用程序公开给内部和外部网络，对您的容器化应用的通信进行精细的控制。要将来自集群外部的 HTTP、HTTPS 和其他服务的传入请求连接到集群内部的服务，可以使用 **Ingress** 资源。

如果容器需要磁盘存储而不是数据库存储（可以通过服务提供），则可以将卷添加到清单中，使该存储可供 pod 使用。您可以配置清单以创建持久性卷 (PV)，或动态创建添加到 **Pod** 定义中的卷。

在定义了组成应用程序的一组 pod 后，您可以在 **Deployment** 和 **DeploymentConfig** 对象中定义这些 pod。

8.3.2. 应用程序类型

接下来，考虑您的应用程序类型对其运行方式的影响。

Kubernetes 定义了适用于不同类型应用程序的不同工作负载。要确定适合应用程序的工作负载，请考虑该应用程序是否：

- 运行结束后即告完成。 例如，启动某个应用程序以生成报告，并在报告完成时退出。之后一个月内可能不会再次运行这个应用程序。对于这些类型的应用程序，合适的 OpenShift Container Platform 对象包括 **Job** 和 **CronJob** 对象。
- 预计将持续运行。 对于长时间运行的应用程序，您可以编写一个部署。
- 需要高度可用。 如果应用程序需要高可用性，那么您需要调整部署的大小，使其包含不止一个实例。**Deployment** 或 **DeploymentConfig** 对象可以包含该类型的应用程序的副本集。借助副本集，pod 可以跨越多个节点运行，确保即使在 worker 中断时该应用程序也始终可用。
- 需要在每个节点上运行。 某些类型的 Kubernetes 应用程序设计为在集群中的每个 master 节点或 worker 节点上运行。例如，DNS 和监控应用程序需要在每个节点上持续运行。您可以将这类应用程序作为守护进程集运行。您还可以根据节点标签在节点的子集上运行守护进程。
- 需要生命周期管理。 当您要移交应用程序供其他人使用时，请考虑创建 Operator。Operator 可帮助您构建智能功能，自动处理备份和升级之类的事务。与 Operator Lifecycle Manager (OLM) 相结合，集群管理器可以将 Operator 公开给选定命名空间，以便集群中的用户可以运行它们。
- 具有标识或编号要求。应用程序可能具有标识或编号要求。 例如，您可能需要运行应用程序的三个实例，并将这些实例命名为 **0**、**1** 和 **2**。一个有状态的集合适合这个应用程序。 有状态的集合对需要独立存储的应用程序（如数据库和 zookeeper 集群）最有用。

8.3.3. 可用的支持组件

您编写的应用程序可能需要支持组件，如数据库或日志记录组件。为满足这一需求，可以从 OpenShift Container Platform Web 控制台中提供的以下目录获取所需的组件：

- OperatorHub，可在每个 OpenShift Container Platform 4.16 集群中使用。借助 OperatorHub，集群操作员可以使用来自红帽、红帽认证合作伙伴和社区成员的 Operator。集群操作员可以在集群中的所有命名空间或选定命名空间中提供这些 Operator，让开发人员能够通过他们的应用程序启动并配置这些 Operator。
- 模板，对于一次性类型的应用程序很有用。在该应用程序中，组件的生命周期在安装后并不重要。模板提供了一种简便方式，可以从最小的开销开始开发 Kubernetes 应用程序。模板可以是资源定义列表，可以是 **Deployment**、**Service**、**Route** 或其他对象。如果要更改名称或资源，可通过参数形式在模板中设置这些值。

您可以根据开发团队的特定需求，配置支持的 Operator 和模板，然后在开发人员开展工作的命名空间中提供它们。 许多人将共享模板添加到 **openshift** 命名空间中，因为可以从所有其他命名空间访问这个命名空间。

8.3.4. 应用清单

借助 Kubernetes 清单（manifest），您可以更加全面地了解组成 Kubernetes 应用程序的组件。您可以将这些清单编写为 YAML 文件，并通过应用到集群来部署它们，例如通过运行 **oc apply** 命令。

8.3.5. 后续步骤

此时，请考虑对容器开发过程进行自动化的方法。理想情况下，您可以使用某种 CI 管道来构建镜像并将其推送到 registry。特别是，GitOps 管道可将容器开发与 Git 存储库集成在一起，您将使用 Git 存储库来存储构建应用程序所需的软件。

到目前为止的工作流程可能如下所示：

- 第 1 天：编写一些 YAML。然后，运行 **oc apply** 命令将 YAML 应用于集群并测试其是否正常工作。
- 第 2 天：将 YAML 容器配置文件放进您的 Git 存储库中。想要安装该应用或协助您改进的人可以从那里拉取 YAML，并应用到他们用于运行应用程序的集群中。
- 第 3 天：考虑为应用程序编写 Operator。

8.4. 面向 OPERATOR 进行开发

如果您的应用程序要提供给他人运行，最好将其打包并部署为 Operator。如前文所述，Operator 向您的应用程序添加了一个生命周期组件。使用它可以实现在安装应用程序后需要进行的维护任务。

在将应用程序创建为 Operator 时，您可以纳入自己有关如何运行和维护应用程序的知识。您可以内置用来升级、备份、扩展应用程序或跟踪其状态的功能。正确配置应用程序后，维护任务（如更新 Operator）可以自动发生，并且不为 Operator 用户所见。

例如，设置为在特定时间自动备份数据的 Operator 就非常实用。让 Operator 在设定的时间管理应用程序备份，可以使系统管理员免于记忆这些事务。

传统上手动完成的任何应用程序维护（如备份数据或轮转证书）都可以借助 Operator 自动完成。

第 9 章 RED HAT ENTERPRISE LINUX COREOS (RHCOS)

9.1. 关于 RHCOS

Red Hat Enterprise Linux CoreOS (RHCOS) 通过提供带有自动化远程升级功能的 Red Hat Enterprise Linux (RHEL) 的质量标准来代表下一代单用途容器操作系统技术。

对于所有 OpenShift Container Platform 机器，仅支持将 RHCOS 作为 OpenShift Container Platform 4.16 的组件。RHCOS 是唯一受 OpenShift Container Platform control plane 或 master 机器支持的操作系统。虽然 RHCOS 是所有集群机器的默认操作系统，但您仍可以创建使用 RHEL 作为其操作系统的计算（compute）机器（也称为 worker ）。OpenShift Container Platform 4.16 中有两个通用的部署 RHCOS 的方法：

- 如果在安装程序置备的基础架构上安装集群，则 RHCOS 镜像会在安装过程中下载到目标平台中。适当的 Ignition 配置文件（控制 RHCOS 配置）也被下载并用于部署机器。
- 如果将集群安装到您自己管理的基础架构上，则必须遵循安装文档来获取 RHCOS 镜像，生成 Ignition 配置文件，并且使用 Ignition 配置文件来置备机器。

9.1.1. RHCOS 主要功能

下表描述了 RHCOS 操作系统的主要功能：

- **基于 RHEL**：底层操作系统主要由 RHEL 组件构成。支持 RHEL 的相同质量、安全性和控制措施也支持 RHCOS。例如，RHCOS 软件位于 RPM 软件包中，并且每个 RHCOS 系统都以 RHEL 内核以及由 systemd 初始化系统管理的一组服务启动。
- **控制的不可变性**：尽管 RHCOS 包含 RHEL组件，但它的管理要比默认的 RHEL 安装更加严格。管理从 OpenShift Container Platform 集群远程执行。设置 RHCOS 机器时，您只能修改一些系统设置。这种受控的不变性使 OpenShift Container Platform 可以存储集群中 RHCOS 系统的最新状态，因而始终都能创建额外的机器并根据最新的 RHCOS 配置执行更新。
- **CRI-O 容器运行时**：尽管 RHCOS 包含运行 Docker 所需的 OCI 和 libcontainer 格式容器的功能，但它融合的是 CRI-O 容器引擎，而非 Docker 容器引擎。通过专注于 Kubernetes 平台（例如 OpenShift Container Platform）所需的功能，CRI-O 可以提供与不同 Kubernetes 版本兼容的特定功能。与提供更大功能集的容器引擎相比，CRI-O 对内存的要求更低，且对安全的攻击面更小。目前，CRI-O 是 OpenShift Container Platform 集群中唯一可用的引擎。
  CRI-O 可以使用 runC 或 crun 容器运行时来启动和管理容器。有关如何启用 crun 的详情，请参考创建 **ContainerRuntimeConfig** CR 的文档。
- **容器工具程序组**：对于诸如构建、复制和以其他方式管理容器的任务，RHCOS 用一组兼容的容器工具来代替 Docker CLI 工具。podman CLI 工具支持许多容器运行时功能，例如运行、启动、停止、列举和删除容器及容器镜像。skopeo CLI 工具可以复制、身份认证和签名镜像。您可以使用 **crictl** CLI 工具来处理 CRI-O 容器引擎中的容器和 pod。虽然不建议在 RHCOS 中直接使用这些工具，但可以把它们用于调试目的。
- **rpm-ostree 升级**：RHCOS 具有使用**rpm-ostree** 系统进行事务升级的功能。更新是通过容器镜像交付的，并且是 OpenShift Container Platform 更新过程的一部分。部署之后，拉取、提取容器镜像并将其写入磁盘，然后修改启动加载程序以启动到新版本。机器将以滚动方式重启并进入更新，确保对集群容量的影响最小。
- **Bootupd 固件和 bootloader 更新**：Package manager 和混合系统（如如 **rpm-ostree**）不会更新固件或者 bootloader。通过 **bootupd**，RHCOS 用户现在可以访问跨系统发行、系统分析操作系统更新工具，该工具管理在现代架构中（如 x86_64、ppc64le 和 aarch64）运行的 UEFI 和旧 BIOS 引导模式中的固件和引导更新。

有关如何安装 **bootupd** 的详情，请参考*使用 bootupd 更新引导装载程序*文档。

- **通过 Machine Config Operator 更新**：在 OpenShift Container Platform 中，Machine Config Operator 处理操作系统升级。**rpm-ostree** 以原子单元形式提供升级，不像 **yum** 升级那样单独升级各个软件包。新的操作系统部署在升级过程中进行，并在下次重启时才会生效。如果升级出现问题，则进行一次回滚并重启就能使系统返回到以前的状态。OpenShift Container Platform 中的 RHCOS 升级是在集群更新期间执行的。

对于 RHCOS 系统，**rpm-ostree** 文件系统的布局具有以下特征：

- **/usr** 是操作系统二进制文件和库的存储位置，并且是只读的。我们不支持更改此设置。
- **/etc**、**/boot** 和 **/var** 在系统上是可写的，但只能由 Machine Config Operator 更改。
- **/var/lib/containers** 是用于存储容器镜像的图形存储位置。

9.1.2. 选择如何配置 RHCOS

RHCOS 的设计思想是，在需要最小用户配置的情况下在 OpenShift Container Platform 集群中进行部署。在它的最基本的形式中包括：

- 从置备的基础架构（如 AWS）开始，或者自行置备基础架构。
- 在运行 **openshift-install** 时，在 **install-config.yaml** 文件中提供少量信息，如凭证和集群名称。

由于 OpenShift Container Platform 中的 RHCOS 系统被设计为通过 OpenShift Container Platform 集群来进行全面管理，因此不建议直接修改 RHCOS 机器。为了调试，您可以对 RHCOS 机器集群进行有限的直接访问，但您不应该直接配置 RHCOS 系统。相反，如果您需要在 OpenShift Container Platform 节点上添加或更改功能，请考虑采用以下方式进行更改：

- **Kubernetes 工作负载对象，如 DaemonSet 和 Deployment**：如果要将服务或其他用户级别功能添加到集群中，请考虑将它们添加为 Kubernetes 工作负载对象。为了减少在后续的升级中破坏集群的风险，在特定节点配置之外应用这些功能是最佳方法。
- **第 2 天自定义配置任务**：如果可能，在不对集群节点进行任何自定义配置的情况下启动集群，并在集群启动后再进行必要的节点更改。这些更改可以更轻松地管理，且不太可能破坏以后的更新。创建机器配置或修改 Operator 自定义资源是进行这些自定义的方法。
- **第 1 天的自定义配置**：对于集群首次启动时必须执行的自定义配置，可以使用以下方法修改集群以便更改可在首次引导时就生效。第 1 天自定义配置可通过运行 **openshift-install** 期间的 Ignition 配置和清单文件实现，也可以在由用户置备的 ISO 安装过程中添加引导选项实现。

以下是您可以在第 1 天进行定制的示例：

- **内核参数**：如果在集群首次引导时需要特定的内核功能或调整。
- **磁盘加密**：如果您的安全规则需要对节点上的根文件系统加密，比如 FIPS 支持。
- **内核模块**：如果 Linux 内核没有默认为某个特定的硬件设备（如网卡或视频卡）提供可用的模块。
- **Chronyd**：如果要为节点提供特定的时钟设置，如时间服务器的位置。

要完成这些任务，您可以为 **openshift-install** 提供参数，使其包含额外的对象，如 **MachineConfig**。这些创建机器配置的步骤可在集群启动后传递给 Machine Config Operator。

注意

- 安装程序生成的 Ignition 配置文件包含在 24 小时后过期的证书，然后在过期时进行续订。如果在更新证书前关闭集群，且集群在 24 小时后重启，集群会自动恢复过期的证书。一个例外是，您必须手动批准待处理的 **node-bootstrapper** 证书签名请求(CSR)来恢复 kubelet 证书。如需更多信息，*请参阅从过期的 control plane 证书* 中恢复的文档。

- 建议您在 Ignition 配置文件生成后的 12 小时内使用它们，因为 24 小时的证书会在集群安装后的 16 小时到 22 小时间进行轮转。通过在 12 小时内使用 Ignition 配置文件，您可以避免在安装过程中因为执行了证书更新而导致安装失败的问题。

9.1.3. 选择如何部署 RHCOS

OpenShift Container Platform 的 RHCOS 安装的区别取决于，您是在安装程序置备的基础架构上部署，还是在由用置备的基础架构上部署：

- **安装程序置备**：有些云环境提供预配置的基础架构，以便您使用最小配置来启动 OpenShift Container Platform 集群。对于这些类型的安装，您可以提供 Ignition 配置来在每个节点上放置内容，以便在集群首次启动时可用。

- **用户置备**：如果您置备自己的基础架构，则具有更大的灵活性来向 RHCOS 节点中添加内容。例如：引导 RHCOS ISO 安装程序时您可以添加内核参数来安装每个系统。但是，在多数情况下，如果操作系统本身需要配置，最好通过 Ignition 配置来提供那些配置。

Ignition 工具仅在首次设置 RHCOS 系统时运行。之后，可以使用机器配置来提供 Ignition 配置。

9.1.4. 关于 Ignition

Ignition 是 RHCOS 在初始配置期间用于操作磁盘的实用程序。它可完成常见的磁盘任务，如分区磁盘、格式化分区、写入文件和配置用户等。首次启动时，Ignition 从安装介质或您指定的位置读取其配置，并将配置应用到机器。

无论您要安装集群还是向其中添加机器，Ignition 始终都执行 OpenShift Container Platform 集群机器的初始配置。实际的系统设置大多都在每台机器上进行。对于每台机器，Ignition 都会获取 RHCOS 镜像并启动 RHCOS 内核。内核命令行上的选项标识部署的类型，以及启用了 Ignition 的初始 RAM 磁盘 (initramfs) 的位置。

9.1.4.1. Ignition 工作方式

要使用 Ignition 创建机器，需要 Ignition 配置文件。OpenShift Container Platform 安装程序创建部署集群所需的 Ignition 配置文件。这些文件基于您直接提供给安装程序或通过 **install-config.yaml** 文件提供的信息。

Ignition 配置机器的方式类似于 cloud-init 或 Linux Anaconda kickstart 等工具配置系统的方式，但有一些重要的区别：

- Ignition 从一个初始 RAM 磁盘运行，该磁盘与您要安装到的系统相隔开。因此，Ignition 可以重新分区磁盘、设置文件系统，以及对机器的持久文件系统执行其他更改。与之相反，cloud-init 会在系统启动时作为机器的初始系统的一部分运行，因而不易对磁盘分区之类的事项进行基本的更改。使用 cloud-init 时，难以在节点启动期间重新配置启动过程。

- Ignition 旨在初始化系统，而不是更改现有系统。机器完成初始化且内核在安装的系统上运行之后，OpenShift Container Platform 集群中的 Machine Config Operator 将完成所有后续的机器配置。

- Ignition 不是完成一组定义的操作，而是实施声明性配置。它会在新机器启动之前检查所有分区、文件、服务和其他项目是否就位。然后进行更改，例如将必要的文件复制到磁盘，以便新机器符合指定的配置。

- 在 Ignition 完成机器配置之后，内核将继续运行，但会丢弃初始 RAM 磁盘，并转至磁盘上已安装的系统。所有新的系统服务和其他功能都将启动，无需重启系统。

- 因为 Ignition 会确认所有新机器是否都符合声明的配置，所以不会存在配置不全的机器。如果机器设置失败，则初始化过程不会完成，而且 Ignition 也不会启动新机器。您的集群永不会包含配置不全的机器。如果 Ignition 无法完成，机器就不会添加到集群中。您必须添加新的机器。一些失败配置任务的结果可能一直不为人所知，直到后来依赖它的某些事物也失败才被发现。对这类问题的故障排除将会非常困难。而 Ignition 配置的工作方式可以防止出现这个问题。

- 如果 Ignition 配置存在问题，而导致一台机器的设置失败，Ignition 不会尝试使用相同的配置来设置另一台机器。例如，故障可能源自于某一个 Ignition 配置，而构成该配置的父级和子级配置都希望创建同一个文件。在这种情况下，出现的故障将导致 Ignition 配置无法再次用于设置其他机器，直到问题解决为止。

- 如果您有多个 Ignition 配置文件，您可获得该组配置的并集。由于 Ignition 是声明性的，配置之间的冲突可能会导致 Ignition 无法设置机器。这些文件中信息的次序无关紧要。Ignition 将以最有成效的方式对每项设置进行分类和实施。例如，如果一个文件需要有多个层级深的目录，而另一个文件需要其路径上的某一目录，则首先创建后一个文件。Ignition 按深度排序并创建所有文件、目录和链接。

- 因为 Ignition 可以从全空的硬盘开始，所以它可以做 cloud-init 不能做的任务：从头开始在裸机上设置系统（使用 PXE 启动等功能）。在裸机情形中，Ignition 配置注入启动分区，以便 Ignition 可以找到它并正确配置系统。

9.1.4.2. Ignition 操作序列

OpenShift Container Platform 集群中 RHCOS 机器的 Ignition 过程包括以下步骤：

- 机器获取其 Ignition 配置文件。control plane 机器从 bootstrap 机器获取 Ignition 配置文件，worker 机器从 control plane 机器获取 Ignition 配置文件。

- Ignition 在机器上创建磁盘分区、文件系统、目录和链接。它支持 RAID 阵列，但不支持 LVM 卷。

- Ignition 将持久文件系统的根目录挂载到 initramfs 中的 **/sysroot** 目录，然后开始在 **/sysroot** 中工作。

- Ignition 配置所有定义的文件系统，并将它们设置为在运行时进行相应地挂载。

- Ignition 运行 **systemd** 临时文件，将必要的文件填充到 **/var** 目录。

- Ignition 运行 Ignition 配置文件，以设置用户、systemd 单元文件和其他配置文件。

- Ignition 卸载 initramfs 中挂载的持久系统中的所有组件。

- Ignition 启动新机器的 init 进程，该过程在系统引导期间运行的机器上启动所有其他服务。

在此过程结束时，计算机已准备好加入群集，不需要重启。

9.2. 查看 IGNITION 配置文件

要查看用于部署 bootstrap 机器的 Ignition 配置文件，请运行以下命令：

```
$ openshift-install create ignition-configs --dir $HOME/testconfig
```

回答几个问题后，您所在的目录中将出现 **bootstrap.ign**、**master.ign** 和 **worker.ign** 文件。

要查看 **bootstrap.ign** 文件的内容，请通过 **jq** 过滤器对其进行管道传递。以下是该文件的片段：

```
$ cat $HOME/testconfig/bootstrap.ign | jq
{
  "ignition": {
    "version": "3.2.0"
  },
  "passwd": {
    "users": [
      {
        "name": "core",
        "sshAuthorizedKeys": [
          "ssh-rsa AAAAB3NzaC1yc...."
        ]
      }
    ]
  },
  "storage": {
    "files": [
      {
        "overwrite": false,
        "path": "/etc/motd",
        "user": {
          "name": "root"
        },
        "append": [
          {
            "source": "data:text/plain;charset=utf-
8; base64,VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVkIHdo
ZW4gdGhlIG1hc3RlciBpcyBmdWxseSB1cC4KClRoZSBwcmltYXJ5IHNlcnZpY2VzIGFyZSByZWxlYXNlL
WltYWdlLnNlcnZpY2UgZm9sbG93ZWQgYnkgYm9vdGt1YmUuc2VydmljZS4gVG8gd2F0Y2ggdGhlaXI
gc3RhdHVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RsIC1iIC1mIC11IHJlbGVhc2UtaW1hZ2Uuc2Vydm
ljZSAtdSBib290a3ViZS5zZXJ2aWNlCg=="
          }
        ],
        "mode": 420
      },
...
```

要解码 **bootstrap.ign** 文件中列出的文件内容，请将代表该文件内容的 base64 编码的数据字符串通过管道传递给 **base64 -d** 命令。以下示例使用了上方输出中添加至 bootstrap 机器的 **/etc/motd** 文件的内容：

```
$ echo
VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVkIHdoZW4gdG
hlIG1hc3RlciBpcyBmdWxseSB1cC4KClRoZSBwcmltYXJ5IHNlcnZpY2VzIGFyZSByZWxlYXNlLWltYWdl
LnNlcnZpY2UgZm9sbG93ZWQgYnkgYm9vdGt1YmUuc2VydmljZS4gVG8gd2F0Y2ggdGhlaXIgc3Rhd
HVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RsIC1iIC1mIC11IHJlbGVhc2UtaW1hZ2Uuc2VydmljZSAtd
SBib290a3ViZS5zZXJ2aWNlCg== | base64 --decode
```

输出示例

■

```
This is the bootstrap node; it will be destroyed when the master is fully up.

The primary services are release-image.service followed by bootkube.service. To watch their status, run e.g.

  journalctl -b -f -u release-image.service -u bootkube.service
```

对 **master.ign** 和 **worker.ign** 文件重复这些命令，查看每种机器类型的 Ignition 配置文件的来源。 对于 **worker.ign**，您应该会看到类似于下面这一行，它确认了如何从 bootstrap 获取 Ignition 配置：

```
"source": "https://api.myign.develcluster.example.com:22623/config/worker",
```

您可以从 **bootstrap.ign** 文件中了解到以下内容：

- 格式：文件的格式在 Ignition 配置规格中定义。MCO 稍后使用相同格式的文件，将更改合并到机器的配置中。
- 内容：由于 bootstrap 机器为其他机器提供 Ignition 配置，因此 master 机器和 worker 机器的 Ignition 配置信息都与 bootstrap 机器的配置一起存储在 **bootstrap.ign** 中。
- 大小：文件长度超过 1300 行，包含指向各种资源的路径。
- 要复制到机器的每个文件的内容实际上编码为数据 URL，这往往会使内容读起来有些混乱。（使用前面显示的 **jq** 和 **base64** 命令可使内容更易读。）
- 配置：Ignition 配置文件的不同部分通常涵盖刚放入机器文件系统中的文件，而不是用于修改现有文件的命令。例如，不添加与配置该服务的 NFS 相关的一节，是仅添加一个 NFS 配置文件，然后在系统启动时由 init 进程启动该文件。
- 用户：创建名为 **core** 的用户，并将您的 SSH 密钥分配给该用户。这样，您可以使用该用户名和凭证来登录集群。
- 存储：存储部分标识添加到每台机器的文件。一些值得注意的文件包括 **/root/.docker/config.json**（提供集群从容器镜像 registry 表拉取时所需的凭证），以及 **/opt/openshift/manifests** 中用于配置集群的一系列清单文件：
- systemd：**systemd** 部分包含用于创建 **systemd** 单元文件的内容。这些文件用于在启动时启动服务，还用于在运行中的系统上管理这些服务。
- 原语：Ignition 还公开低级别原语，其他工具可以此为基础进行构建。

9.3. 安装后更改 IGNITION 配置

机器配置池管理节点集群及其相应机器配置。机器配置包含集群的配置信息。列出所有已知的机器配置池：

```
$ oc get machineconfigpools
```

输出示例

```
NAME   CONFIG                                   UPDATED UPDATING DEGRADED
master master-1638c1aea398413bb918e76632f20799 False   False    False
worker worker-2feef4f8288936489a5a832ca8efe953 False   False    False
```

列出所有机器配置：

```
$ oc get machineconfig
```

输出示例

```
NAME                                       GENERATEDBYCONTROLLER   IGNITIONVERSION   CREATED
OSIMAGEURL

00-master                                  4.0.0-0.150.0.0-dirty   3.2.0             16m
00-master-ssh                              4.0.0-0.150.0.0-dirty                     16m
00-worker                                  4.0.0-0.150.0.0-dirty   3.2.0             16m
00-worker-ssh                              4.0.0-0.150.0.0-dirty                     16m
01-master-kubelet                          4.0.0-0.150.0.0-dirty   3.2.0             16m
01-worker-kubelet                          4.0.0-0.150.0.0-dirty   3.2.0             16m
master-1638c1aea398413bb918e76632f20799    4.0.0-0.150.0.0-dirty   3.2.0             16m
worker-2feef4f8288936489a5a832ca8efe953    4.0.0-0.150.0.0-dirty   3.2.0             16m
```

在涉及到应用这些机器配置时，Machine Config Operator 的行为与 Ignition 有些不同。机器配置按顺序读取（从 00* 到 99*）。机器配置中的标签标识每个所用于的节点类型（master 节点或 worker 节点）。如果同一文件出现在多个机器配置文件中，则以最后一个文件为准。例如，出现在 99* 文件中的任何文件都将替换出现在 00* 文件中的同一文件。输入的 **MachineConfig** 对象将合并为一个“呈现的”**MachineConfig** 对象，该对象将被 Operator 用作目标，也是您可以在 MachineConfigPool 中看到的值。

要查看从机器配置中管理的文件，请查找特定 **MachineConfig** 对象中的"Path:"。例如：

```
$ oc describe machineconfigs 01-worker-container-runtime | grep Path:
```

输出示例

```
        Path:       /etc/containers/registries.conf
        Path:       /etc/containers/storage.conf
        Path:       /etc/crio/crio.conf
```

确保为机器指定一个较新的名称（如 10-worker-container-runtime）。请记住，每个文件的内容都是 URL 样式的数据。然后将新机器配置应用到集群。

第 10 章 准入插件

准入（Admission）插件用于帮助规范 OpenShift Container Platform 的功能。

10.1. 关于准入插件

准入插件截获对 master API 的请求，以验证资源请求。在对请求进行身份验证并授权后，准入插件可确保遵循任何关联的策略。例如，它们通常会被用来强制执行安全策略、资源限制或配置要求。

准入插件以准入链的形式按序列运行。如果序列中的任何准入插件拒绝某个请求，则整个链将中止并返回错误。

OpenShift Container Platform 为每个资源类型都启用了默认的准入插件。这些是集群正常工作所需要的。准入插件会忽略那些不由它们负责的资源。

除了默认的插件外，准入链还可以通过调用自定义 webhook 服务器的 webhook 准入插件动态进行扩展。webhook 准入插件有两种：变异准入插件和验证准入插件。变异准入插件会首先运行，它可以修改资源并验证请求。验证准入插件会验证请求，并在变异准入插件之后运行，以便由变异准入插件触发的改变也可以被验证。

通过一个变异准入插件调用 webhook 服务器可能会对与目标对象相关的资源产生副作用。在这种情况下，必须执行相应的步骤来验证最终结果是正确的。

警告

应谨慎使用动态准入机制，因为它会影响到集群的 control plane 操作。在 OpenShift Container Platform 4.16 中使用通过 webhook 准入插件调用 webhook 服务器的功能时，请确保已仔细阅读了相关文档，并对变异所带来的副作用进行了测试。需要包括一个步骤，以便在请求没有通过整个准入链时，把资源恢复到其原始状态。

10.2. 默认准入插件

OpenShift Container Platform 4.16 中启用了默认的验证和准入插件。这些默认插件有助于基本的 control plane 功能，如入口策略、集群资源限制覆盖和配额策略。

重要

不要在默认项目中运行工作负载或共享对默认项目的访问权限。为运行核心集群组件保留默认项目。

以下默认项目被视为具有高度特权：**default**, **kube-public**, **kube-system**, **openshift**, **openshift-infra**, **openshift-node**，其他系统创建的项目的标签 **openshift.io/run-level** 被设置为 **0** 或 **1**。依赖于准入插件（如 pod 安全准入、安全性上下文约束、集群资源配额和镜像引用解析）的功能无法在高特权项目中工作。

以下列表包含默认准入插件：

例 10.1. 验证准入插件

- LimitRanger
- ServiceAccount
- PodNodeSelector
- 优先级
- PodTolerationRestriction
- OwnerReferencesPermissionEnforcement
- PersistentVolumeClaimResize
- RuntimeClass
- CertificateApproval
- CertificateSigning
- CertificateSubjectRestriction
- autoscaling.openshift.io/ManagementCPUsOverride
- authorization.openshift.io/RestrictSubjectBindings
- scheduling.openshift.io/OriginPodNodeEnvironment
- network.openshift.io/ExternalIPRanger
- network.openshift.io/RestrictedEndpointsAdmission
- image.openshift.io/ImagePolicy
- security.openshift.io/SecurityContextConstraint
- security.openshift.io/SCCExecRestrictions
- route.openshift.io/IngressAdmission
- config.openshift.io/ValidateAPIServer
- config.openshift.io/ValidateAuthentication
- config.openshift.io/ValidateFeatureGate
- config.openshift.io/ValidateConsole
- operator.openshift.io/ValidateDNS
- config.openshift.io/ValidateImage
- config.openshift.io/ValidateOAuth
- config.openshift.io/ValidateProject
- config.openshift.io/DenyDeleteClusterConfiguration

- **config.openshift.io/ValidateScheduler**
- **quota.openshift.io/ValidateClusterResourceQuota**
- **security.openshift.io/ValidateSecurityContextConstraints**
- **authorization.openshift.io/ValidateRoleBindingRestriction**
- **config.openshift.io/ValidateNetwork**
- **operator.openshift.io/ValidateKubeControllerManager**
- **ValidatingAdmissionWebhook**
- **ResourceQuota**
- **quota.openshift.io/ClusterResourceQuota**

例 10.2. 变异准入插件

- **NamespaceLifecycle**
- **LimitRanger**
- **ServiceAccount**
- **NodeRestriction**
- **TaintNodesByCondition**
- **PodNodeSelector**
- 优先级
- **DefaultTolerationSeconds**
- **PodTolerationRestriction**
- **DefaultStorageClass**
- **StorageObjectInUseProtection**
- **RuntimeClass**
- **DefaultIngressClass**
- **autoscaling.openshift.io/ManagementCPUsOverride**
- **scheduling.openshift.io/OriginPodNodeEnvironment**
- **image.openshift.io/ImagePolicy**
- **security.openshift.io/SecurityContextConstraint**
- **security.openshift.io/DefaultSecurityContextConstraints**
- **MutatingAdmissionWebhook**

10.3. WEBHOOK 准入插件

除了 OpenShift Container Platform 默认准入插件外，也可以通过称为 webhook 服务器的 webhook 准入插件来实施动态准入，从而扩展准入链的功能。Webhook 服务器通过 HTTP 在定义的端点调用。

OpenShift Container Platform 中有两个类型的 webhook 准入插件：

- 在准入过程中, *变异（mutating）准入插件*可以执行任务，如注入关联性标签。
- 在准入过程结束时，*验证（validating）准入插件*会用来确保对象被正确配置。例如，确保关联性标签与预期一样。如果验证通过，OpenShift Container Platform 会按照配置来调度对象。

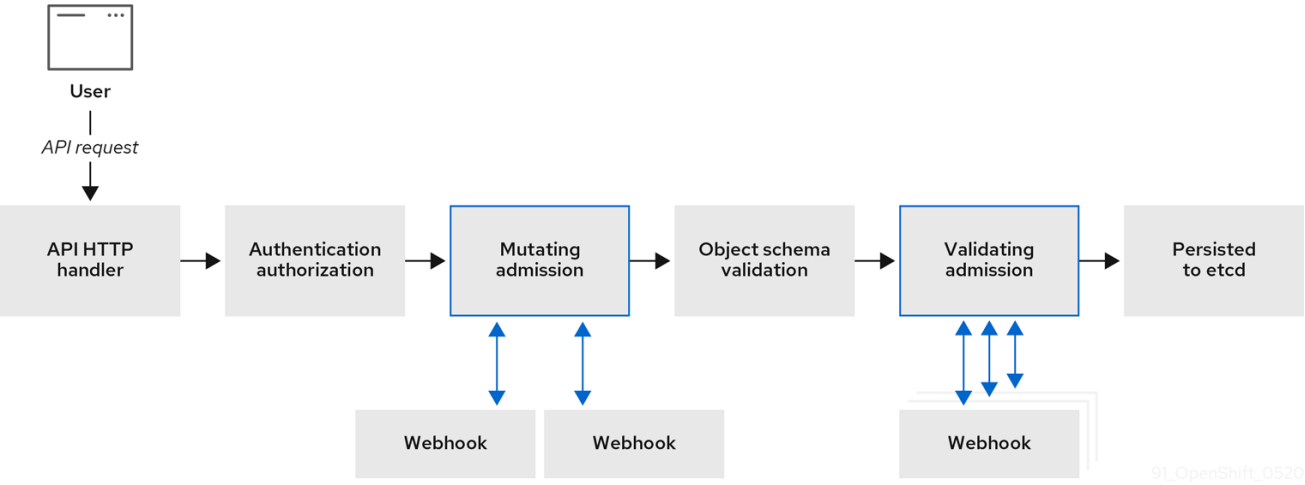
当 API 请求到来时，变异或验证准入插件会使用配置中的外部 webhook 列表,并并行调用它们：

- 如果所有 webhook 都批准请求，准入链将继续。
- 如果任何 webhook 拒绝了请求，则拒绝准入请求。而拒绝的原因是第一个拒绝的原因。
- 如果多个 webhook 拒绝准入请求，则只有第一个拒绝原因返回给用户。
- 如果在调用 webhook 时遇到错误，请求将被拒绝。或者根据错误策略的设置，webhook 会被忽略。如果错误策略被设置为 **Ignore**，则失败时请求将被无条件接受。如果策略被设置为 **Fail**，失败时请求将被拒绝。使用 **Ignore** 可能会为所有客户端造成无法预计的行为。

webhook 准入插件和 webhook 服务器之间的通信必须使用 TLS。生成一个 CA 证书，使用该证书为您的 webhook 准入服务器使用的服务器证书签名。PEM 编码的 CA 证书使用某种机制（如 service serving 证书 secret）提供给 webhook 准入插件。

下图演示了调用多个 webhook 服务器的序列准入链进程。

图 10.1. 带有变异准入插件和验证准入插件的 API 准入链



webhook 准入插件用例示例。在这个示例中，所有 pod 都必须具有一组通用标签。在本例中，变异准入插件可以注入标签，验证准入插件可以检查标签是否如预期。之后，OpenShift Container Platform 可以调度包含所需标签的 pod，并拒绝那些没有所需标签的 pod。

Webhook 准入插件常见使用案例包括：

- 命名空间保留。

- 限制由 SR-IOV 网络设备插件管理的自定义网络资源。
- 定义可启用污点以决定哪些 pod 应该调度到节点上的容限。
- Pod 优先级类验证。

注意

OpenShift Container Platform 中的最大默认 webhook 超时值为 13 秒，且无法更改。

10.4. WEBHOOK 准入插件类型

集群管理员可以通过 API 服务器准入链中的变异准入插件或验证准入插件调用 webhook 服务器。

10.4.1. 变异准入插件

变异（mutating）准入插件在准入过程的变异期间被调用，这允许在保留资源内容前修改资源内容。一个可通过变异准入插件调用的 webhook 示例是 Pod Node Selector 功能，它使用命名空间上的注解来查找标签选择器并将其添加到 pod 规格中。

变异准入插件配置示例

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration (1)
metadata:
  name: <webhook_name> (2)
webhooks:
- name: <webhook_name> (3)
  clientConfig: (4)
    service:
      namespace: default (5)
      name: kubernetes (6)
      path: <webhook_url> (7)
    caBundle: <ca_signing_certificate> (8)
  rules: (9)
  - operations: (10)
    - <operation>
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - <resource>
  failurePolicy: <policy> (11)
  sideEffects: None
```

1. 指定变异准入插件配置。
2. **MutatingWebhookConfiguration** 对象的名称。将 **<webhook_name>** 替换为适当的值。
3. 要调用的 webhook 的名称。将 **<webhook_name>** 替换为适当的值。
4. 如何连接、信任和将数据发送到 webhook 服务器的信息。

5. 创建前端服务的命名空间。
6. 前端服务的名称。
7. 用于准入请求的 webhook URL。将 **<webhook_url>** 替换为适当的值。
8. 为 webhook 服务器使用的服务器证书签名的 PEM 编码的 CA 证书。将 **<ca_signing_certificate>** 替换为采用 base64 格式的适当证书。
9. 定义 API 服务器何时应使用此 webhook 准入插件的规则。
10. 一个或多个触发 API 服务器调用此 webhook 准入插件的操作。可能的值包括 **create**、**update**、**delete** 或 **connect**。将 **<operation>** 和 **<resource>** 替换为适当的值。
11. 指定如果 webhook 服务器不可用时策略应如何执行。将 **<policy>** 替换为 **Ignore** （在失败时无条件接受请求）或 **Fail** （拒绝失败的请求）。使用 **Ignore** 可能会为所有客户端造成无法预计的行为。

重要

在 OpenShift Container Platform 4.16 中，由用户创建或通过变异准入插件的控制循环创建的对象可能会返回非预期的结果，特别是在初始请求中设置的值被覆盖时，我们不建议这样做。

10.4.2. 验证准入插件

在准入过程的验证阶段会调用验证准入插件。在此阶段，可以在特定的 API 资源强制不能改变，以确保资源不会再次更改。Pod Node Selector 也是一个 webhook 示例，它由验证准入插件调用，以确保所有 **nodeSelector** 字段均受命名空间的节点选择器限制。

验证准入插件配置示例

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration (1)
metadata:
  name: <webhook_name> (2)
webhooks:
- name: <webhook_name> (3)
  clientConfig: (4)
    service:
      namespace: default (5)
      name: kubernetes (6)
      path: <webhook_url> (7)
    caBundle: <ca_signing_certificate> (8)
  rules: (9)
  - operations: (10)
    - <operation>
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
```

```
  - <resource>
failurePolicy: <policy> (11)
sideEffects: Unknown
```

1. 指定验证准入插件配置。
2. **ValidatingWebhookConfiguration** 对象的名称。将 **<webhook_name>** 替换为适当的值。
3. 要调用的 webhook 的名称。将 **<webhook_name>** 替换为适当的值。
4. 如何连接、信任和将数据发送到 webhook 服务器的信息。
5. 创建前端服务的命名空间。
6. 前端服务的名称。
7. 用于准入请求的 webhook URL。将 **<webhook_url>** 替换为适当的值。
8. 为 webhook 服务器使用的服务器证书签名的 PEM 编码的 CA 证书。将 **<ca_signing_certificate>** 替换为采用 base64 格式的适当证书。
9. 定义 API 服务器何时应使用此 webhook 准入插件的规则。
10. 一个或多个触发 API 服务器调用此 webhook 准入插件的操作。可能的值包括 **create**、**update**、**delete** 或 **connect**。将 **<operation>** 和 **<resource>** 替换为适当的值。
11. 指定如果 webhook 服务器不可用时策略应如何执行。将 **<policy>** 替换为 **Ignore** （在失败时无条件接受请求）或 **Fail** （拒绝失败的请求）。使用 **Ignore** 可能会为所有客户端造成无法预计的行为。

10.5. 配置动态准入

此流程概述了配置动态准入的高级步骤。通过配置 webhook 准入插件来调用 webhook 服务器以扩展准入链的功能。

webhook 服务器也被配置为一个聚合的 API 服务器。这允许其他 OpenShift Container Platform 组件使用内部凭证与 webhook 通信，并可使用 **oc** 命令进行测试。另外，这还可在 webhook 中启用基于角色的访问控制（RBAC），并防止向 webhook 公开其他 API 服务器的令牌信息。

先决条件

- 一个具有集群管理员权限的 OpenShift Container Platform 账户。
- 已安装 OpenShift Container Platform CLI (**oc**)。
- 公布的 webhook 服务器容器镜像。

流程

1. 使用镜像 registry 构建 webhook 服务器容器镜像，并将其提供给集群。
2. 创建本地 CA 密钥和证书，并使用它们为 webhook 服务器的证书签名请求（CSR）签名。
3. 为 webhook 资源创建新项目：

```
$ oc new-project my-webhook-namespace (1)
```

1. 请注意，webhook 服务器可能会需要一个特定的名称。

4. 在名为 **rbac.yaml** 的文件中为聚合的 API 服务定义 RBAC 规则：

```
apiVersion: v1
kind: List
items:

- apiVersion: rbac.authorization.k8s.io/v1 (1)
  kind: ClusterRoleBinding
  metadata:
    name: auth-delegator-my-webhook-namespace
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: system:auth-delegator
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server

- apiVersion: rbac.authorization.k8s.io/v1 (2)
  kind: ClusterRole
  metadata:
    annotations:
    name: system:openshift:online:my-webhook-server
  rules:
  - apiGroups:
    - online.openshift.io
    resources:
    - namespacereservations (3)
    verbs:
    - get
    - list
    - watch

- apiVersion: rbac.authorization.k8s.io/v1 (4)
  kind: ClusterRole
  metadata:
    name: system:openshift:online:my-webhook-requester
  rules:
  - apiGroups:
    - admission.online.openshift.io
    resources:
    - namespacereservations (5)
    verbs:
    - create

- apiVersion: rbac.authorization.k8s.io/v1 (6)
  kind: ClusterRoleBinding
  metadata:
    name: my-webhook-server-my-webhook-namespace
  roleRef:
```

```
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: system:openshift:online:my-webhook-server
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server

- apiVersion: rbac.authorization.k8s.io/v1 (7)
  kind: RoleBinding
  metadata:
    namespace: kube-system
    name: extension-server-authentication-reader-my-webhook-namespace
  roleRef:
    kind: Role
    apiGroup: rbac.authorization.k8s.io
    name: extension-apiserver-authentication-reader
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server

- apiVersion: rbac.authorization.k8s.io/v1 (8)
  kind: ClusterRole
  metadata:
    name: my-cluster-role
  rules:
  - apiGroups:
    - admissionregistration.k8s.io
    resources:
    - validatingwebhookconfigurations
    - mutatingwebhookconfigurations
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - ""
    resources:
    - namespaces
    verbs:
    - get
    - list
    - watch

- apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: my-cluster-role
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: my-cluster-role
  subjects:
```

```
- kind: ServiceAccount
  namespace: my-webhook-namespace
  name: server
```

1. 将身份验证和授权委托给 webhook 服务器 API。
2. 允许 webhook 服务器访问集群资源。
3. 指向资源。在这个示例中指向 **namespacereservations** 资源。
4. 启用聚合的 API 服务器创建准入审核。
5. 指向资源。在这个示例中指向 **namespacereservations** 资源。
6. 启用 webhook 服务器访问集群资源。
7. 角色绑定来读取终止身份验证的配置。
8. 聚合的 API 服务器的默认集群角色和集群角色绑定。

5. 将这些 RBAC 规则应用到集群：

```
$ oc auth reconcile -f rbac.yaml
```

6. 创建名为 **webhook-daemonset.yaml** 的 YAML 文件，用于将 webhook 部署为命名空间中的守护进程设置服务器：

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  namespace: my-webhook-namespace
  name: server
  labels:
    server: "true"
spec:
  selector:
    matchLabels:
      server: "true"
  template:
    metadata:
      name: server
      labels:
        server: "true"
    spec:
      serviceAccountName: server
      containers:
      - name: my-webhook-container (1)
        image: <image_registry_username>/<image_path>:<tag> (2)
        imagePullPolicy: IfNotPresent
        command:
        - <container_commands> (3)
        ports:
        - containerPort: 8443 (4)
        volumeMounts:
        - mountPath: /var/serving-cert
```

```
        name: serving-cert
      readinessProbe:
        httpGet:
          path: /healthz
          port: 8443 (5)
          scheme: HTTPS
    volumes:
    - name: serving-cert
      secret:
        defaultMode: 420
        secretName: server-serving-cert
```

1. 请注意，webhook 服务器可能会预期有一个特定的容器名称。
2. 指向 webhook 服务器容器镜像。将 **<image_registry_username>/<image_path>:<tag>** 替换为适当的值。
3. 指定 webhook 容器运行命令。将 **<container_commands>** 替换为适当的值。
4. 定义 pod 中的目标端口。这个示例使用端口 8443。
5. 指定就绪探测使用的端口。这个示例使用端口 8443。

7. 部署守护进程集：

```
$ oc apply -f webhook-daemonset.yaml
```

8. 在名为 **webhook-secret.yaml** 的 YAML 文件中为 service serving 证书签名程序定义 secret:

```
apiVersion: v1
kind: Secret
metadata:
  namespace: my-webhook-namespace
  name: server-serving-cert
type: kubernetes.io/tls
data:
  tls.crt: <server_certificate> (1)
  tls.key: <server_key> (2)
```

1. 引用签名的 webhook 服务器证书。将 **<server_certificate>** 替换为适当的 base64 格式的证书。
2. 引用签名的 webhook 服务器密钥。将 **<server_key>** 替换为 base64 格式的适当密钥。

9. 创建 secret：

```
$ oc apply -f webhook-secret.yaml
```

10. 在名为 **webhook-service.yaml** 的 YAML 文件中定义服务帐户和服务：

```
apiVersion: v1
kind: List
items:
```

```
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    namespace: my-webhook-namespace
    name: server

- apiVersion: v1
  kind: Service
  metadata:
    namespace: my-webhook-namespace
    name: server
    annotations:
      service.beta.openshift.io/serving-cert-secret-name: server-serving-cert
  spec:
    selector:
      server: "true"
    ports:
    - port: 443 (1)
      targetPort: 8443 (2)
```

1. 定义服务侦听的端口。这个示例使用端口 443。
2. 定义服务转发连接到的 pod 中的目标端口。这个示例使用端口 8443。

11. 在集群中公开 webhook 服务器：

```
$ oc apply -f webhook-service.yaml
```

12. 在名为 **webhook-crd.yaml** 的文件中为 webhook 服务器定义自定义资源定义：

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: namespacereservations.online.openshift.io (1)
spec:
  group: online.openshift.io (2)
  version: v1alpha1 (3)
  scope: Cluster (4)
  names:
    plural: namespacereservations (5)
    singular: namespacereservation (6)
    kind: NamespaceReservation (7)
```

1. 反映 **CustomResourceDefinition spec** 值，格式为 **<plural>.<group>**。在这个示例中使用 **namespacereservations** 资源。
2. REST API 组名称。
3. REST API 版本名称。
4. 可接受的值是 **Namespaced** 或 **Cluster**。

5. URL 中包括的复数名称。
6. **oc** 输出中的别名。
7. 资源清单的引用。

13. 应用自定义资源定义：

```
$ oc apply -f webhook-crd.yaml
```

14. 在名为 **webhook-api-service.yaml** 的文件中配置 webhook 服务器也作为一个聚合的 API 服务器：

```
apiVersion: apiregistration.k8s.io/v1beta1
kind: APIService
metadata:
  name: v1beta1.admission.online.openshift.io
spec:
  caBundle: <ca_signing_certificate> (1)
  group: admission.online.openshift.io
  groupPriorityMinimum: 1000
  versionPriority: 15
  service:
    name: server
    namespace: my-webhook-namespace
  version: v1beta1
```

1. 为 webhook 服务器使用的服务器证书签名的 PEM 编码的 CA 证书。将 **<ca_signing_certificate>** 替换为采用 base64 格式的适当证书。

15. 部署聚合的 API 服务：

```
$ oc apply -f webhook-api-service.yaml
```

16. 在名为 **webhook-config.yaml** 的文件中定义 webhook 准入插件配置。本例使用验证准入插件：

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: namespacereservations.admission.online.openshift.io (1)
webhooks:
- name: namespacereservations.admission.online.openshift.io (2)
  clientConfig:
    service: (3)
      namespace: default
      name: kubernetes
      path: /apis/admission.online.openshift.io/v1beta1/namespacereservations (4)
    caBundle: <ca_signing_certificate> (5)
  rules:
  - operations:
    - CREATE
    apiGroups:
```

```
    - project.openshift.io
    apiVersions:
    - "*"
    resources:
    - projectrequests
  - operations:
    - CREATE
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - namespaces
  failurePolicy: Fail
```

1. **ValidatingWebhookConfiguration** 对象的名称。在这个示例中使用 **namespacereservations** 资源。
2. 要调用的 webhook 的名称。在这个示例中使用 **namespacereservations** 资源。
3. 通过聚合的 API 启用对 webhook 服务器的访问。
4. 用于准入请求的 webhook URL。在这个示例中使用 **namespacereservation** 资源。
5. 为 webhook 服务器使用的服务器证书签名的 PEM 编码的 CA 证书。将 **<ca_signing_certificate>** 替换为采用 base64 格式的适当证书。

17. 部署 webhook：

```
$ oc apply -f webhook-config.yaml
```

18. 验证 webhook 是否如预期运行。例如，如果您配置了动态准入以保留特定的命名空间，请确认创建这些命名空间的请求会被拒绝，并且创建非保留命名空间的请求会成功。

10.6. 其他资源

- 配置 SR-IOV Network Operator
- 使用节点污点控制 pod 放置
- Pod 优先级名称