Red Hat OpenShift Service on AWS 4

架构

架构概述.

Last Updated: 2024-07-02

Red Hat OpenShift Service on AWS 4 架构

架构概述.

法律通告

摘要

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

目录

第 1 章 架构概述 3
1.1. RED HAT OPENSHIFT SERVICE ON AWS 架构的常见术语表 3
1.2. 了解 RED HAT OPENSHIFT SERVICE ON AWS 与 OPENSHIFT CONTAINER PLATFORM 的不同 6
1.3. 关于 CONTROL PLANE 7
1.4. 关于面向开发人员的容器化应用程序 7
1.5. 关于准入插件 8

第 2 章 RED HAT OPENSHIFT SERVICE ON AWS 架构 9
2.1. RED HAT OPENSHIFT SERVICE ON AWS 简介 9

第 3 章 架构模型 12
3.1. 将 ROSA 与 HCP 和 ROSA CLASSIC 进行比较 12
3.2. 使用 HCP 架构的 ROSA 12
3.3. ROSA CLASSIC 架构 15

第 4 章 CONTROL PLANE 架构 20
4.1. RED HAT OPENSHIFT SERVICE ON AWS 中的机器角色 20
4.2. RED HAT OPENSHIFT SERVICE ON AWS 中的 OPERATOR 22
4.3. ETCD 概述 23

第 5 章 NVIDIA GPU 架构概述 25
5.1. NVIDIA GPU 先决条件 25
5.2. GPU 和 ROSA 25
5.3. GPU 共享方法 25
5.4. RED HAT OPENSHIFT SERVICE ON AWS 的 NVIDIA GPU 功能 27

第 6 章 了解 RED HAT OPENSHIFT SERVICE ON AWS 开发 29
6.1. 关于容器化应用程序开发 29
6.2. 构建一个简单容器 29
6.3. 为 RED HAT OPENSHIFT SERVICE ON AWS 创建 KUBERNETES 清单 32
6.4. 面向 OPERATOR 进行开发 34

第 7 章 准入插件 35
7.1. 关于准入插件 35
7.2. 默认准入插件 35
7.3. WEBHOOK 准入插件 38
7.4. WEBHOOK 准入插件类型 39
7.5. 其他资源 41

第 1 章 架构概述

Red Hat OpenShift Service on AWS 是一个基于云的 Kubernetes 容器平台。Red Hat OpenShift Service on AWS 的基础基于 Kubernetes，因此共享相同的技术。要了解更多有关 Red Hat OpenShift Service on AWS 和 Kubernetes 的信息，请参阅 产品架构。

1.1. RED HAT OPENSHIFT SERVICE ON AWS 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 Red Hat OpenShift Service on AWS 集群上的 Red Hat OpenShift Service 的访问，具有 **dedicated-admin** 角色的管理员可以配置用户身份验证，以确保只有批准的用户访问集群。要与 Red Hat OpenShift Service on AWS 集群交互，您必须使用 Red Hat OpenShift Service on AWS API 进行身份验证。您可以通过在对 Red Hat OpenShift Service on AWS API 的请求中提供 OAuth 访问令牌或 X.509 客户端证书进行验证。

bootstrap

运行最小 Kubernetes 并在 AWS control plane 上部署 Red Hat OpenShift Service 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 Red Hat OpenShift Service on AWS Update Service 的 Operator，并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

一个 Red Hat OpenShift Service on AWS 功能，允许从其 data plane 和 worker 在 AWS 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 Red Hat OpenShift Service on AWS 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

Red Hat OpenShift Service on AWS 集群的网络信息。

node

Red Hat OpenShift Service on AWS 集群中的 worker 机器。节点是虚拟机 (VM) 或物理计算机。

OpenShift CLI (oc)

在终端中运行 Red Hat OpenShift Service on AWS 命令的命令行工具。

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

由 Red Hat OpenShift Service on AWS 提供的 registry 来管理镜像。

Operator

在 Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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

Red Hat OpenShift Service on AWS Update Service (OSUS)为 Red Hat OpenShift Service on AWS 提供无线更新，包括 Red Hat Enterprise Linux CoreOS (RHCOS)。

pod

在 Red Hat OpenShift Service on AWS 集群中运行的一个或多个带有共享资源（如卷和 IP 地址）的容器。pod 是定义、部署和管理的最小计算单元。

私有 registry

Red Hat OpenShift Service on AWS 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

Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS Cluster Manager

一个受管服务，您可以在 AWS 集群上安装、修改、操作和升级 Red Hat OpenShift Service。

RHEL Quay Container Registry

为 Red Hat OpenShift Service on AWS 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

路由公开服务，以允许从 Red Hat OpenShift Service on AWS 实例外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 Red Hat OpenShift Service on AWS 中应用源代码的编程语言创建的镜像，以部署应用程序。

storage

Red Hat OpenShift Service on AWS 支持许多类型的云供应商存储。您可以在 Red Hat OpenShift Service on AWS 集群中管理持久性和非持久性数据的容器存储。

Telemetry

一个组件，用于收集 Red Hat OpenShift Service on AWS 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，以生成由 Red Hat OpenShift Service on AWS 创建的对象列表。

Web 控制台

用于管理 Red Hat OpenShift Service on AWS 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

- 如需有关存储的更多信息，请参阅 Red Hat OpenShift Service on AWS 存储 。
- 如需有关身份验证的更多信息，请参阅 Red Hat OpenShift Service on AWS 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。

1.2. 了解 RED HAT OPENSHIFT SERVICE ON AWS 与 OPENSHIFT CONTAINER PLATFORM 的不同

Red Hat OpenShift Service on AWS 使用与 OpenShift Container Platform 相同的代码库，但以建议的方式安装，以优化性能、可扩展性和安全性。Red Hat OpenShift Service on AWS 是一个完全管理的服务，因此很多在 OpenShift Container Platform 中手动设置的 Red Hat OpenShift Service on AWS 组件和设置会被默认设置。

查看 Red Hat OpenShift Service on AWS 和您自己的基础架构上 OpenShift Container Platform 标准安装之间的以下区别：

OpenShift Container Platform	Red Hat OpenShift Service on AWS
客户安装并配置 OpenShift Container Platform。	Red Hat OpenShift Service on AWS 通过 Red Hat OpenShift Cluster Manager 或 ROSA CLI (**rosa**)安装，并以针对性能、可扩展性和安全性进行了优化的标准化方式安装。

OpenShift Container Platform	Red Hat OpenShift Service on AWS
客户可以选择其计算资源。	Red Hat OpenShift Service on AWS 在客户提供的公共云(Amazon Web Services)中托管和管理。
客户对基础架构具有顶级管理访问权限。	客户有一个内置管理员组(**dedicated-admin**)，但提供顶级管理访问权限。
客户可以使用 OpenShift Container Platform 中提供的所有支持的功能和配置设置。	Red Hat OpenShift Service on AWS 上可能无法提供或更改一些 OpenShift Container Platform 功能和配置设置。
您可以在获取 **control** 角色的机器上设置 control plane 组件，如 API 服务器和 etcd。您可以修改 control plane 组件，但负责备份、恢复和使 control plane 数据高度可用。	红帽设置 control plane 并管理 control plane 组件。control plane 具有高可用性。
您需要为 control plane 和 worker 节点更新底层基础架构。您可以使用 OpenShift Web 控制台更新 OpenShift Container Platform 版本。	当有更新可用时，红帽会自动通知客户。您可以在 OpenShift Cluster Manager 中手动或自动调度更新。
根据红帽订阅或云供应商条款提供支持。	拥有 99.95% 的正常运行时间 SLA 和 24x7 覆盖范围的红帽设计、操作和支持。详情请查看 Red Hat Enterprise Agreement 附录 4 （在线订阅服务）。

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。Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 中是重要的组件，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. 关于准入插件

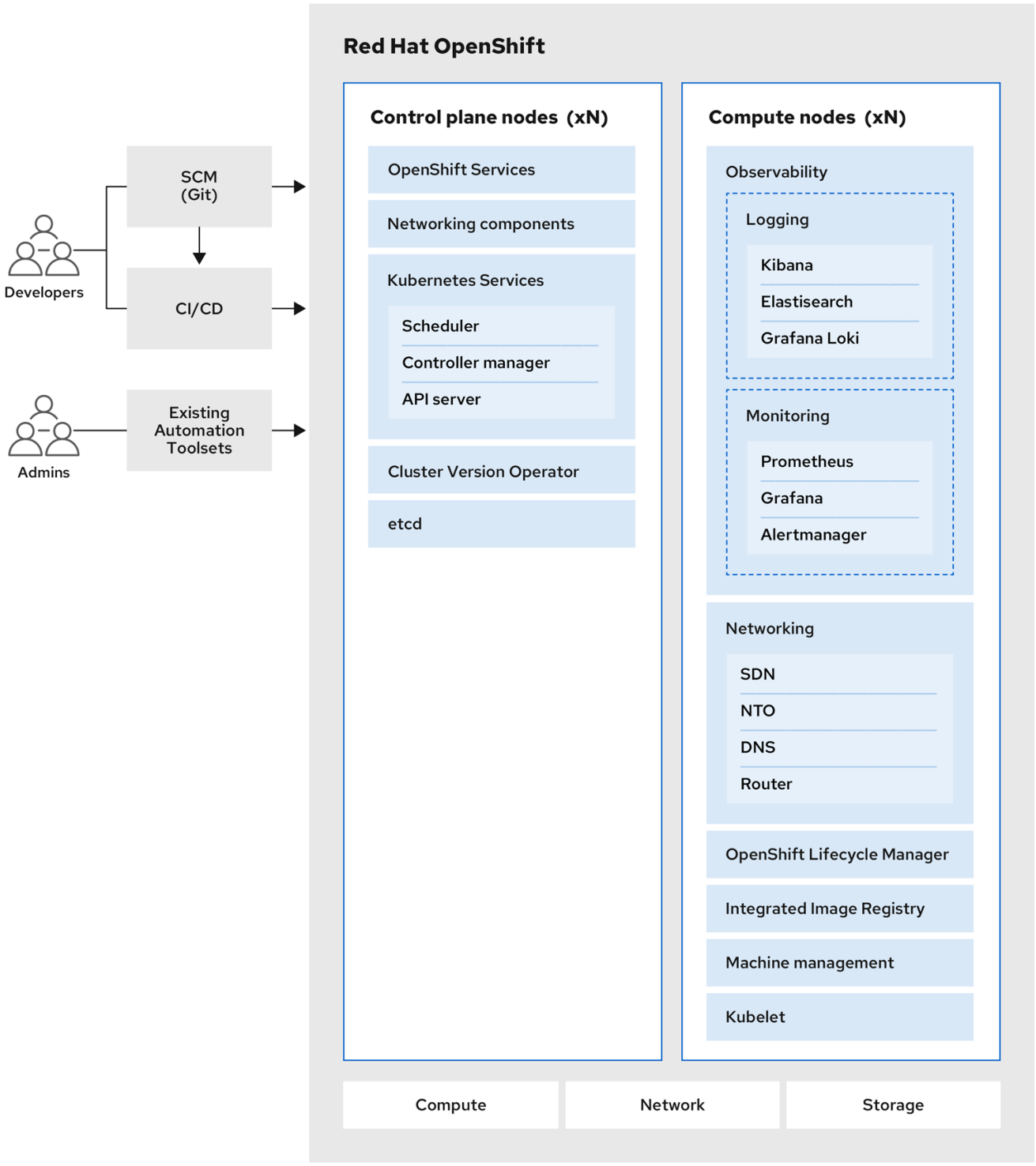
您可以使用 准入插件 规范 Red Hat OpenShift Service on AWS 的功能。在对资源请求进行身份验证和授权后，准入插件会截获对 master API 的资源请求以验证资源请求，并确保扩展策略得到遵循。准入插件用于强制执行安全策略、资源限制、配置要求和其他设置。

第 2 章 RED HAT OPENSHIFT SERVICE ON AWS 架构

2.1. RED HAT OPENSHIFT SERVICE ON AWS 简介

Red Hat OpenShift Service on AWS 是一个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。

随着 Kubernetes 的基础，Red Hat OpenShift Service on AWS 融合了相同的技术，它们充当大规模电信、流视频、游戏、银行和其他应用程序引擎。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。Red Hat OpenShift Service on AWS 提供了扩展任何容器化服务的简单标准方法。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. Red Hat OpenShift Service on AWS 概述

Red Hat OpenShift Service on AWS 为 Kubernetes 提供企业级的改进，包括以下改进：

- 集成了红帽技术。Red Hat OpenShift Service on AWS 中的主要组件来自 Red Hat Enterprise Linux (RHEL)和相关的红帽技术。Red Hat OpenShift Service on AWS 得益于红帽企业级质量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 Red Hat OpenShift Service on AWS 4 具备的重要优势。以下小节描述了 Red Hat OpenShift Service on AWS 的一些独特功能和优势。

2.1.3.1. 定制操作系统

Red Hat OpenShift Service on AWS 使用 Red Hat Enterprise Linux CoreOS (RHCOS)作为所有 control plane 和 worker 节点的操作系统。

RHCOS 包括：

- Ignition，Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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。

2.1.3.2. 简化的更新过程

更新或升级，Red Hat OpenShift Service on AWS 是一个简单、高度自动化的过程。因为 Red Hat OpenShift Service on AWS 完全控制每台机器上运行的系统和服务（包括操作系统本身），所以从中央 control plane 中升级被设计为自动事件。

2.1.3.3. 其他主要功能

Operator 既是 Red Hat OpenShift Service on AWS 4 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 Red Hat OpenShift Service on AWS 中，Operator 充当平台基础，不再需要手动升级操作系统和 control plane 应用程序。Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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。

Red Hat OpenShift Service on AWS 中的其他 Kubernetes 的改进包括软件定义的网络(SDN)、身份验证、日志聚合、监控和路由。Red Hat OpenShift Service on AWS 还提供全面的 Web 控制台和自定义 OpenShift CLI (**oc**)界面。

第 3 章 架构模型

Red Hat OpenShift Service on AWS (ROSA)具有以下集群拓扑：

- 托管 control plane (HCP)- control plane 托管在红帽帐户中，worker 节点部署到客户的 AWS 帐户中。
- Classic - control plane 和 worker 节点部署在客户的 AWS 帐户中。

3.1. 将 ROSA 与 HCP 和 ROSA CLASSIC 进行比较

表 3.1. ROSA 架构比较表

	托管 Control Plane (HCP)	经典
control plane 托管	control plane 组件（如 API 服务器 etcd 数据库）托管在红帽拥有的 AWS 帐户中。	control plane 组件（如 API 服务器 etcd 数据库）托管在客户拥有的 AWS 帐户中。
虚拟私有云 (VPC)	Worker 节点通过 AWS PrivateLink 与 control plane 通信。	Worker 节点和 control plane 节点部署在客户的 VPC 中。
多区部署	control plane 始终在多个可用区(AZ)之间部署。	control plane 可以部署到单个 AZ 或多个 AZ 中。
机器池	每个机器池都部署在单一 AZ （专用子网）中。	机器池可以在单一 AZ 或多个 AZ 中部署。
基础架构节点	不使用任何专用节点来托管平台组件，如入口和镜像 registry。	使用 2 (single-AZ)或 3 (multi-AZ)专用节点来托管平台组件。
OpenShift 功能	平台监控、镜像 registry 和入口控制器部署在 worker 节点上。	平台监控、镜像 registry 和入口控制器部署在专用基础架构节点中。
集群升级	control plane 和每个机器池都可以单独升级。	整个集群必须同时升级。
最小 EC2 占用空间	需要 2 个 EC2 实例来创建集群。	需要 7 (single-AZ)或 9 (multi-AZ) EC2 实例来创建集群。

其他资源

- 地区和可用性区域
- 安全和合规性

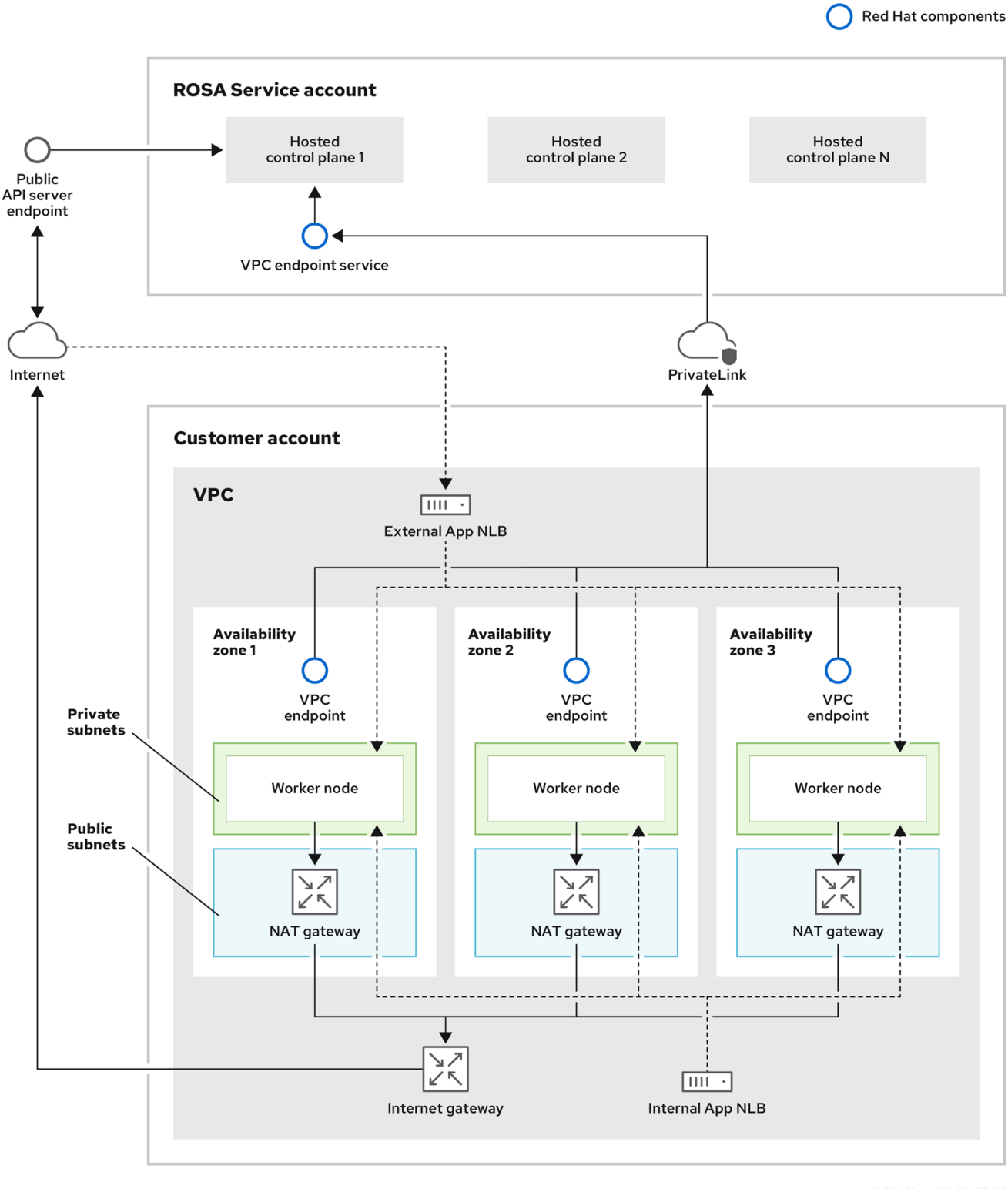
3.2. 使用 HCP 架构的 ROSA

在带有托管 control plane (HCP)的 Red Hat OpenShift Service on AWS (ROSA)中，ROSA 服务托管高可用性、单租户 OpenShift control plane。托管 control plane 部署在 3 个可用区，有 2 个 API 服务器实例和 3 个 etcd 实例。

您可以使用 HCP 集群或没有面向互联网的 API 服务器创建 ROSA。私有 API 服务器只能从 VPC 子网访问。您可以通过 AWS PrivateLink 端点访问托管的 control plane。

worker 节点部署在 AWS 帐户中，并在 VPC 专用子网中运行。您可以从一个或多个可用区添加额外的专用子网，以确保高可用性。Worker 节点由 OpenShift 组件和应用程序共享。OpenShift 组件（如入口控制器、镜像 registry 和监控）部署在 VPC 上托管的 worker 节点上。

图 3.1. 使用 HCP 架构的 ROSA



3.2.1. 在公共和私有网络上使用 HCP 架构的 ROSA

使用 HCP 的 ROSA，您可以在公共或私有网络上创建集群。以下镜像描述了公共和专用网络的架构。

图 3.2. 在公共网络上部署带有 HCP 的 ROSA

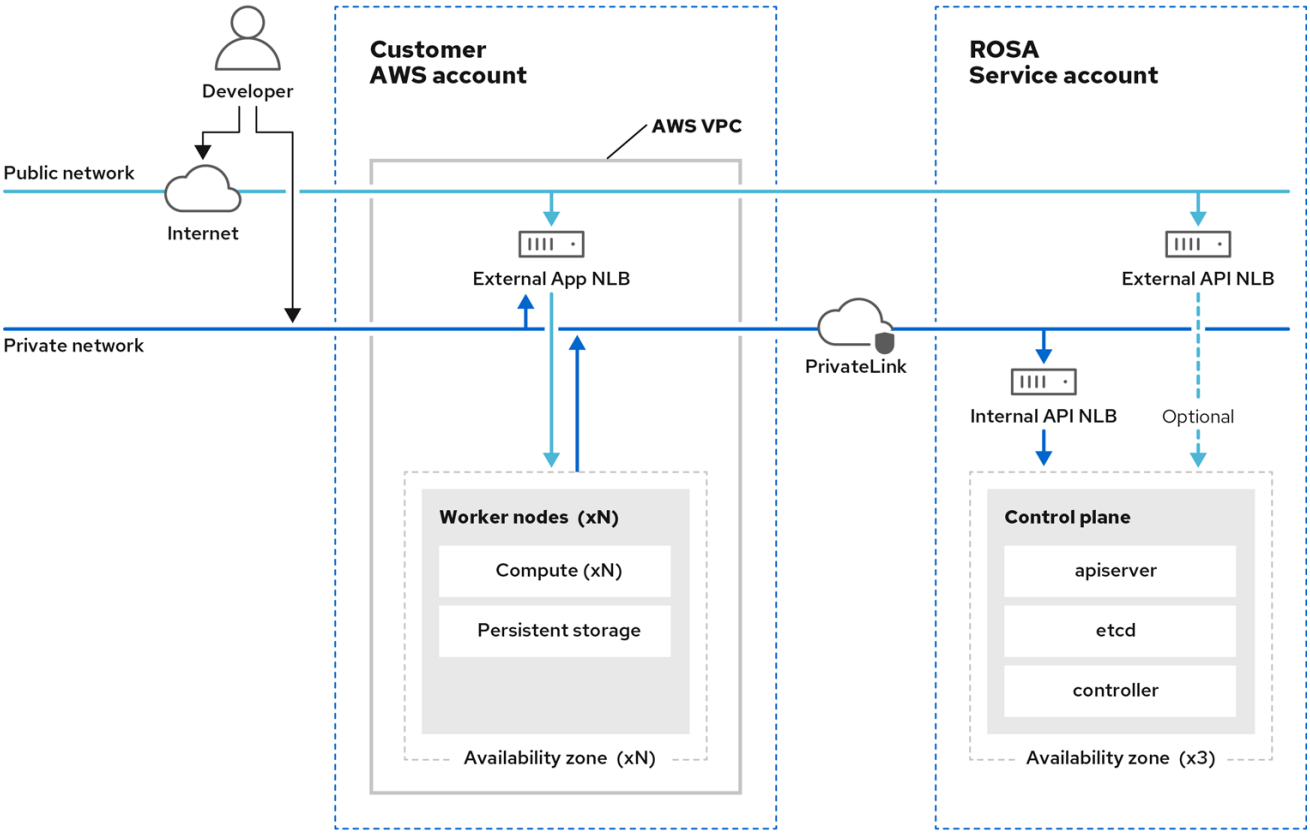
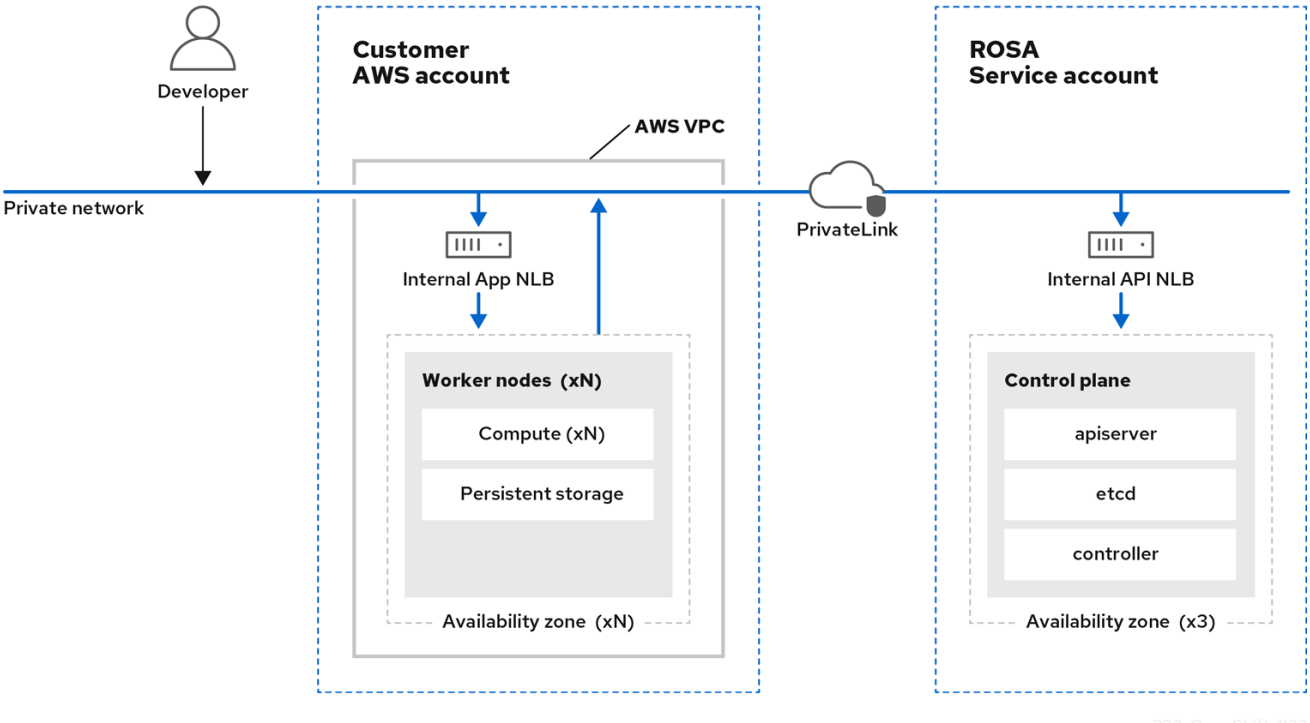


图 3.3. 在私有网络上部署带有 HCP 的 ROSA



3.3. ROSA CLASSIC 架构

在 Red Hat OpenShift Service on AWS (ROSA) Classic 中，control plane 和 worker 节点都部署在 VPC 子网中。

3.3.1. 公共和专用网络上的 ROSA 经典架构

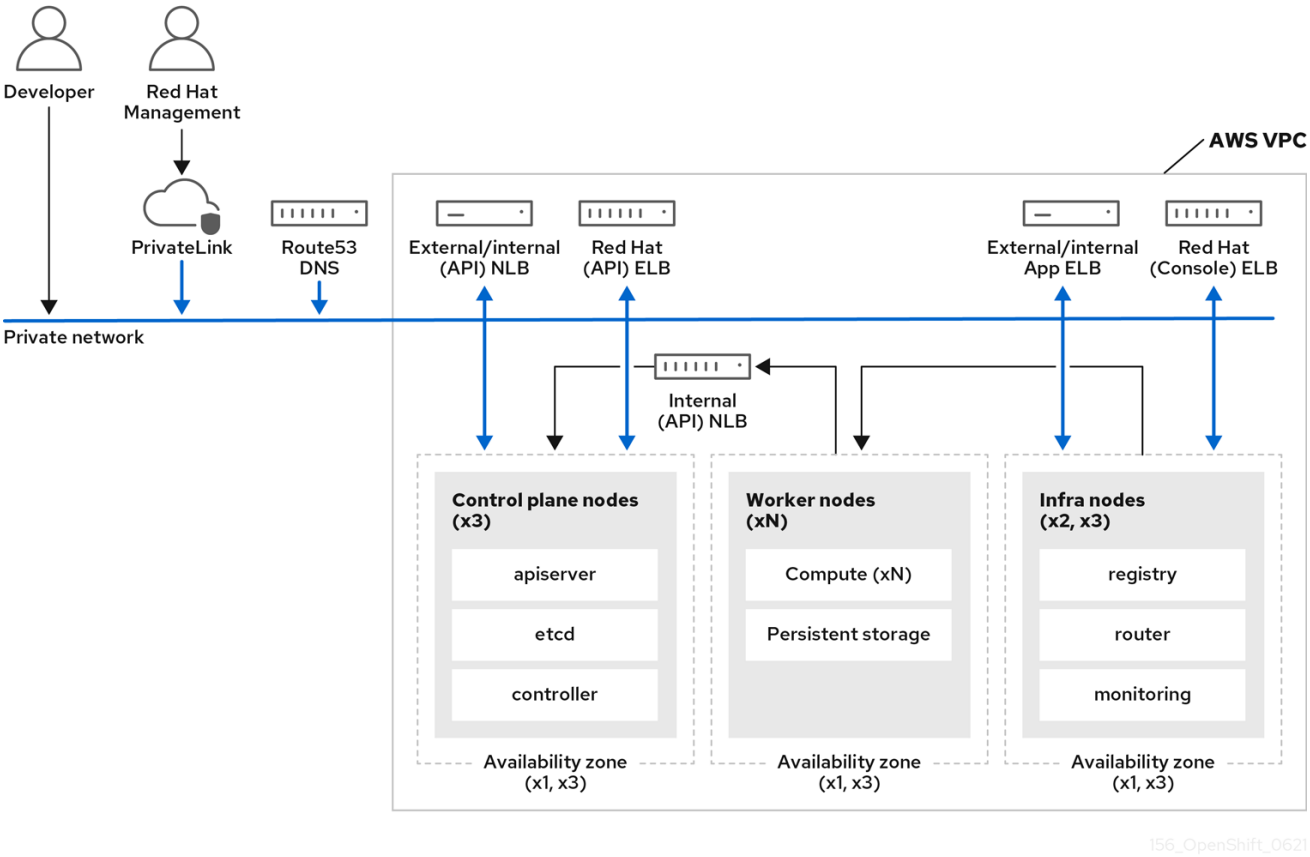
使用 ROSA Classic，您可以创建可通过公共或私有网络访问的集群。

您可以使用以下方法自定义 API 服务器端点和 Red Hat SRE 管理的访问模式：

- Public - API 服务器端点和应用程序路由是面向互联网的。
- private - API 服务器端点和应用程序路由是私有的。私有 ROSA Classic 集群使用一些公共子网，但没有在公共子网中部署 control plane 或 worker 节点。
- 使用 AWS PrivateLink - API 服务器端点和应用程序路由是私有的。对于出口，您的 VPC 不需要公共子网或 NAT 网关。ROSA SRE 管理使用 AWS PrivateLink。

下图描述了在公共和私有网络上部署的 ROSA Classic 集群架构。

图 3.4. 在公共和私有网络上部署的 ROSA Classic



ROSA Classic 集群包括部署 OpenShift 组件的基础架构节点，如入口控制器、镜像 registry 和监控。基础架构节点及其上部署的 OpenShift 组件由 ROSA Service SREs 管理。

ROSA Classic 提供了以下类型的集群：

- 单区集群 - control plane 和 worker 节点托管在单个可用区中。

- 多区集群 - control plane 托管在三个可用区上，可以选择在一个或多个可用区上运行 worker 节点。

3.3.2. AWS PrivateLink 架构

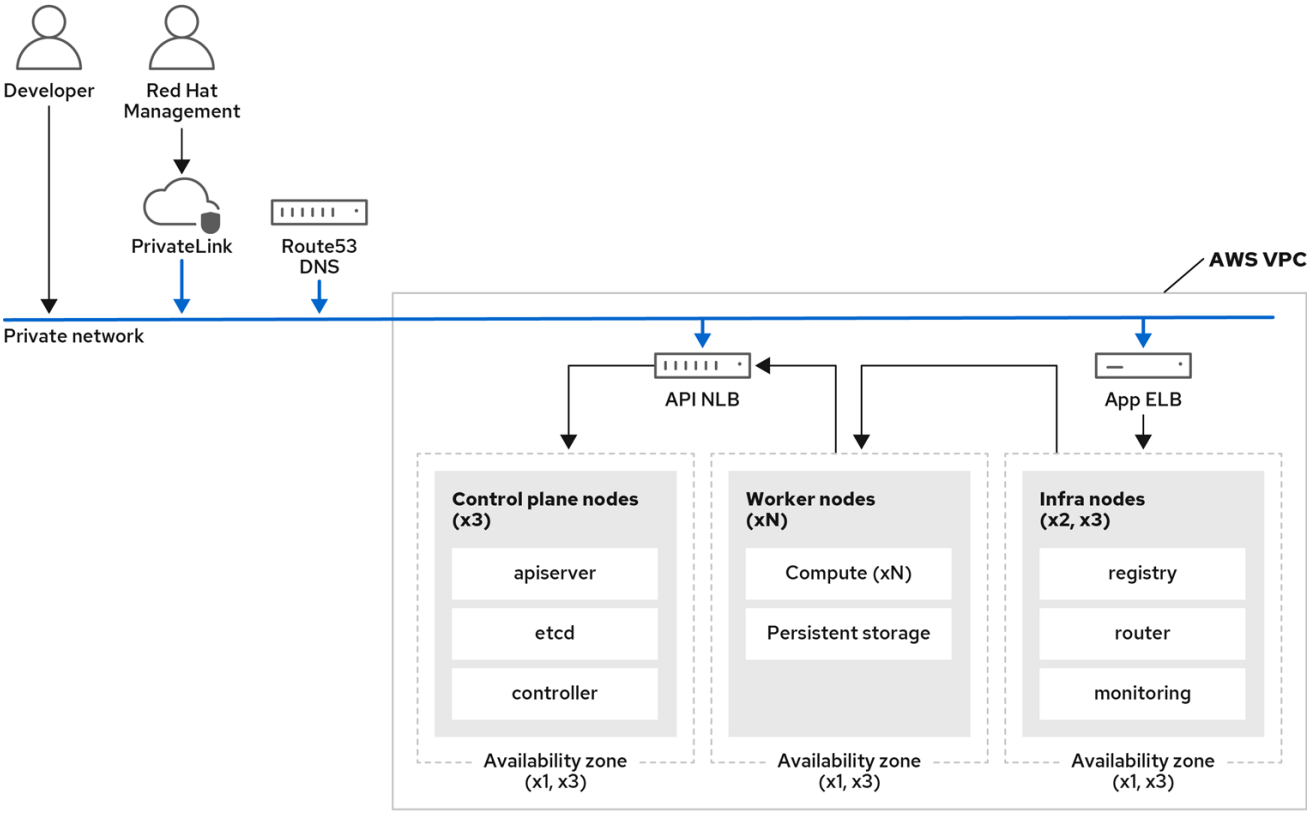
创建 AWS PrivateLink 集群的红帽受管基础架构托管在专用子网上。红帽与用户提供的基础架构之间的连接通过 AWS PrivateLink VPC 端点创建。

注意

仅在现有的 VPC 上支持 AWS PrivateLink。

下图显示了 PrivateLink 集群的网络连接。

图 3.5. 在私有子网中部署的 Multi-AZ AWS PrivateLink 集群



3.3.2.1. AWS 参考架构

在计划如何使用 AWS PrivateLink 设置配置时，AWS 提供多个参考架构。下面是三个示例：

注意

公共子网通过互联网网关直接连接到互联网。**专用子网**通过网络地址转换 (NAT) 网关连接到互联网。

- 带有专用子网和 AWS Site-to-Site VPN 访问的 VPC。
  此配置允许您在不向互联网公开网络的情况下将网络扩展至云中。

要启用通过互联网协议安全性 (IPsec) VPN 隧道与网络通信，此配置包含一个专用子网和虚拟专用网关的虚拟私有云 (VPC)。通过互联网的通信不使用互联网网关。

如需更多信息，请参阅 AWS 文档中的私有子网的 VPC 和 AWS Site-to-Site VPN 访问 。

- 带有公共和私有子网的 VPC (NAT)
此配置允许您隔离网络，以便可以从互联网访问公共子网，但专用子网并非如此。

  只有公共子网可以直接向互联网发送出站流量。专用子网可以通过使用位于公共子网中的网络地址转换 (NAT) 网关访问互联网。这允许数据库服务器使用 NAT 网关连接到互联网以获取软件更新，但不允许连接直接从互联网到数据库服务器。

  如需更多信息，请参阅 AWS 文档中的使用公共和私有子网(NAT)的 VPC。

- 带有公共和私有子网的 VPC 和 AWS Site-to-Site VPN 访问
此配置可让您将网络扩展至云中，并直接从 VPC 访问互联网。

  您可以使用公共子网中的可扩展 Web 前端运行多层应用程序，并将数据存储在由 IPsec AWS Site-to-Site VPN 连接连接的专用子网中。

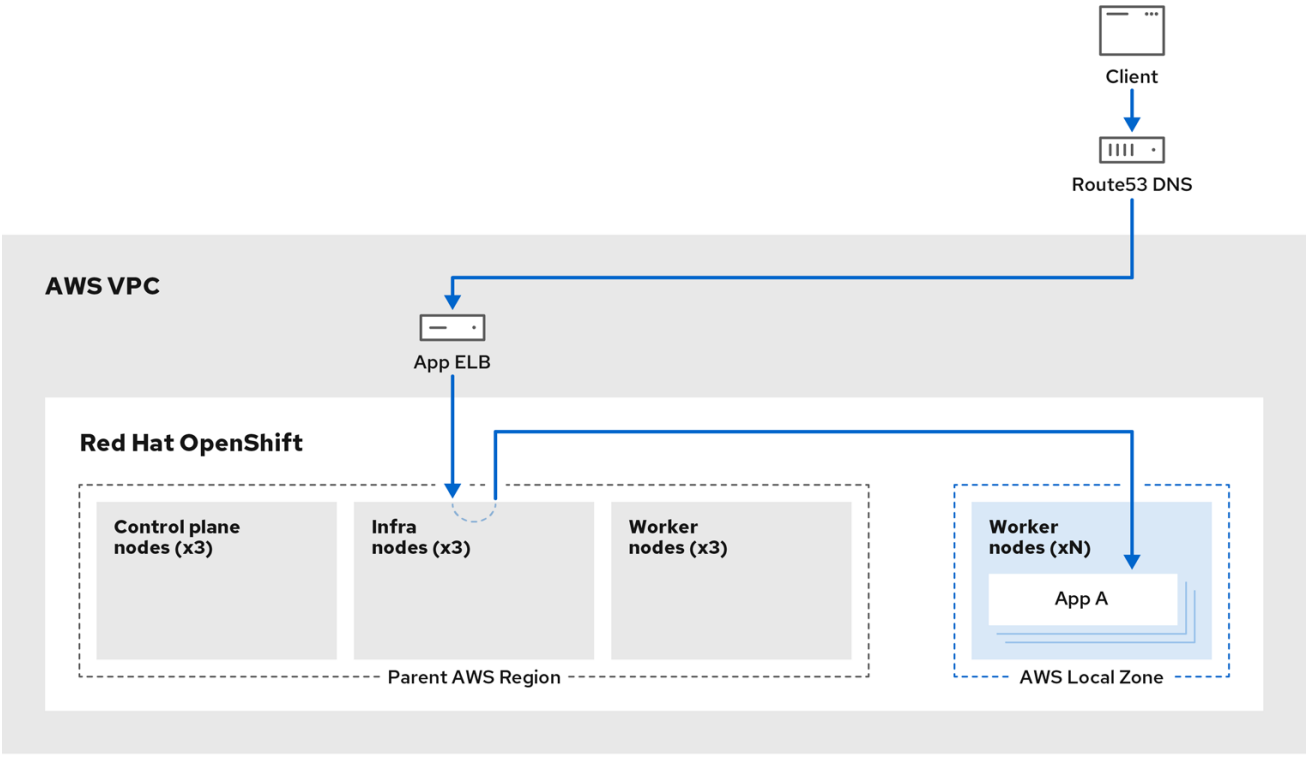
  如需更多信息，请参阅 AWS 文档中的带有公共和私有子网的 VPC 和 AWS Site-to-Site VPN 访问。

3.3.3. 使用本地区域的 ROSA 架构

ROSA 支持使用 AWS Local Zones，这些区域可以满足ropolis-centralized availability 区域，客户可在 VPC 中放置对延迟敏感的应用程序工作负载。本地区是 AWS 区域的扩展，默认情况下不启用。启用并配置 Local Zones 时，流量将扩展到 Local Zones，以获得更大的灵活性和较低延迟。如需更多信息，请参阅"在本地区中配置机器池"。

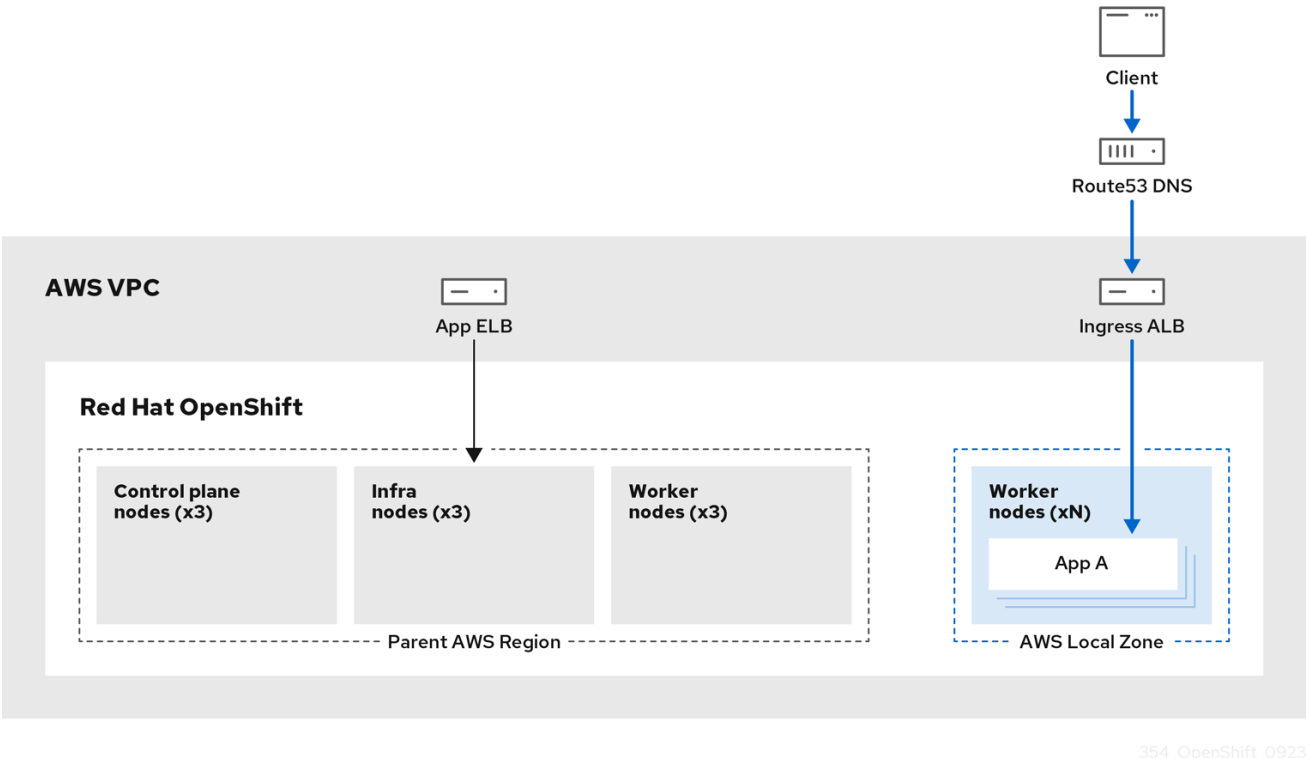
下图显示了没有路由到 Local Zone 的 ROSA 集群。

图 3.6. 没有流量路由到本地区域的 ROSA 集群



下图显示了将流量路由到 Local Zone 的 ROSA 集群。

图 3.7. 将流量路由到本地区域的 ROSA 集群

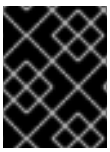


其他资源

- 在本地区中配置机器池

第 4 章 CONTROL PLANE 架构

control plane 由 control plane 机器组成，负责管理 Red Hat OpenShift Service on AWS 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)和一组单独 Operator 的操作来管理对机器的所有升级。



重要

带有托管 control plane (HCP)的 Red Hat OpenShift Service on AWS (ROSA)不支持此 control plane 架构。

4.1. RED HAT OPENSHIFT SERVICE ON AWS 中的机器角色

Red Hat OpenShift Service on AWS 分配主机不同的角色。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。

4.1.1. 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 Red Hat OpenShift Service on AWS 中，计算机器集控制分配了 **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。因为 Red Hat OpenShift Service on AWS 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 *计算机器*。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。在以后的 Red Hat OpenShift Service on AWS 版本中，默认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。

4.1.2. 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 Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 集群的 Kubernetes 服务。

对于大多数 Red Hat OpenShift Service on AWS 集群，control plane 机器由一系列独立的机器 API 资源定义。control plane 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

单可用区集群和多个可用区集群至少需要三个 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 4.1. 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 4.2. 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。

组件	描述
OpenShift OAuth API 服务器	OpenShift OAuth API 服务器验证并配置数据，以向 AWS 上的 Red Hat OpenShift Service 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。Red Hat OpenShift Service on AWS 4 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**

4.2. RED HAT OPENSHIFT SERVICE ON AWS 中的 OPERATOR

Operator 是 Red Hat OpenShift Service on AWS 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。

虽然它们遵循类似的 Operator 概念和目标，但 Red Hat OpenShift Service on AWS 中的 Operator 由两个不同的系统管理，具体取决于它们的目的：

- 由 Cluster Version Operator (CVO) 管理的 Cluster Operator 被默认安装来执行集群功能。

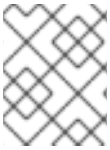
- 可选的附加组件 Operator 由 Operator Lifecycle Manager(OLM)管理，供用户在其应用程序中运行。

4.2.1. 附加组件 Operator

Operator Lifecycle Manager (OLM)和 OperatorHub 是 Red Hat OpenShift Service on AWS 中的默认组件，可帮助将 Kubernetes 原生应用程序作为 Operator 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 Red Hat OpenShift Service on AWS Web 控制台中使用 OperatorHub，具有 **dedicated-admin** 角色和授权用户可以选择要从 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。具有 **dedicated-admin** 角色的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 Operator。



注意

Operator Hub 市场中列出的所有 Operator 都应该可用于安装。这些 Operator 被视为客户工作负载，不受 Red Hat Site Reliability Engineering (SRE) 监控。

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 Red Hat OpenShift Service on AWS 架构的集群 Operator。

其他资源

- 有关在 AWS 上的 Red Hat OpenShift Service 中运行附加组件 Operator 的详情，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。

4.3. 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。

4.3.1. 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

4.3.2. 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 Red Hat OpenShift Service on AWS 等 Kubernetes 容器平台上使用 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。

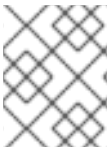
etcd 会维持集群状态，它会持续更新。这个状态会被持续保留，这会导致高频率的、大量的小更改。因此，Red Hat Site Reliability Engineering (SRE)使用快速、低延迟 I/O 支持 etcd 集群成员。

第 5 章 NVIDIA GPU 架构概述

NVIDIA 支持在 Red Hat OpenShift Service on AWS 上使用图形处理单元(GPU)资源。Red Hat OpenShift Service on AWS 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。Red Hat OpenShift Service on AWS 包括对 Kubernetes 的改进，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 Red Hat OpenShift Service on AWS 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。

5.1. 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。

5.2. GPU 和 ROSA

您可以在 NVIDIA GPU 实例类型上部署 Red Hat OpenShift Service on AWS。

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

5.3. GPU 共享方法

红帽和 NVIDIA 已开发 GPU 并发和共享机制，以简化在企业级 Red Hat OpenShift Service on AWS 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

其他资源

- 提高 GPU 利用率

5.3.1. 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

5.3.2. 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。

5.3.3. 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

5.3.4. 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

5.3.5. 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

5.4. RED HAT OPENSHIFT SERVICE ON AWS 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 Service on AWS 的支持。支持以下安装方法：

- 带有 GPU Passthrough 的裸机或 VMware vSphere 上的 Red Hat OpenShift Service on AWS。
- 使用 NVIDIA vGPU 的 VMware vSphere 上的 Red Hat OpenShift Service on AWS。

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 Red Hat OpenShift Service on AWS web 控制台的 cluster **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

第 6 章 了解 RED HAT OPENSHIFT SERVICE ON AWS 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了在 Red Hat OpenShift Service on AWS 中构建和部署容器化 Kubernetes 应用程序时可以创建的资产选项。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。

6.1. 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

6.2. 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。

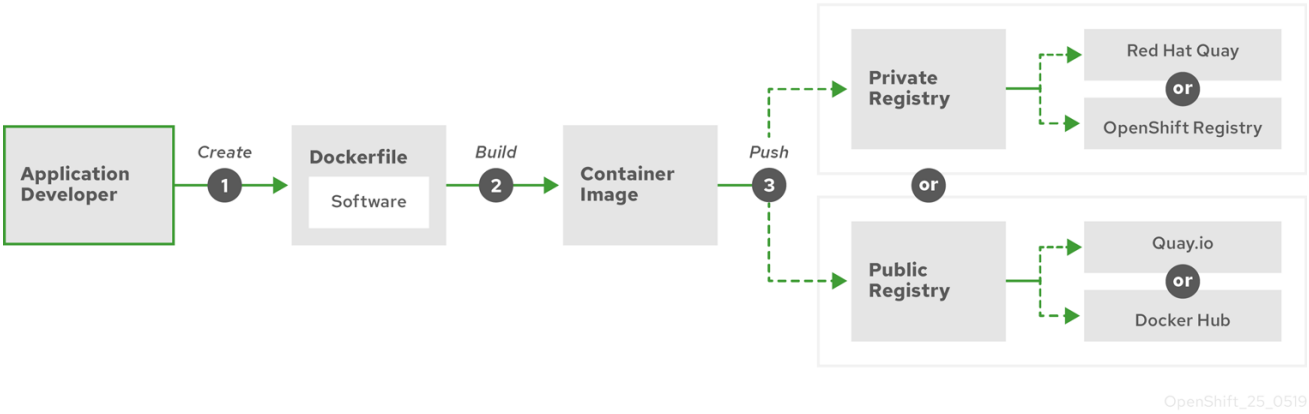
首先，您需要一个用于构建容器的工具，如 buildah 或 docker，还需要一个描述容器中内容的文件，通常是 Dockerfile。

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 6.1. 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。

6.2.1. 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，其中包含专门在 Red Hat OpenShift Service on AWS 或其他 Kubernetes 环境中部署容器的功能。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 Red Hat OpenShift Service on AWS 上运行容器时，您可以使用 CRI-O 容器引擎。CRI-O 在 Red Hat OpenShift Service on AWS 集群中的每一 worker 和 control plane 机器上运行，但 CRI-O 尚不支持作为 Red Hat OpenShift Service on AWS 以外的独立运行时。

6.2.2. 基础镜像选项

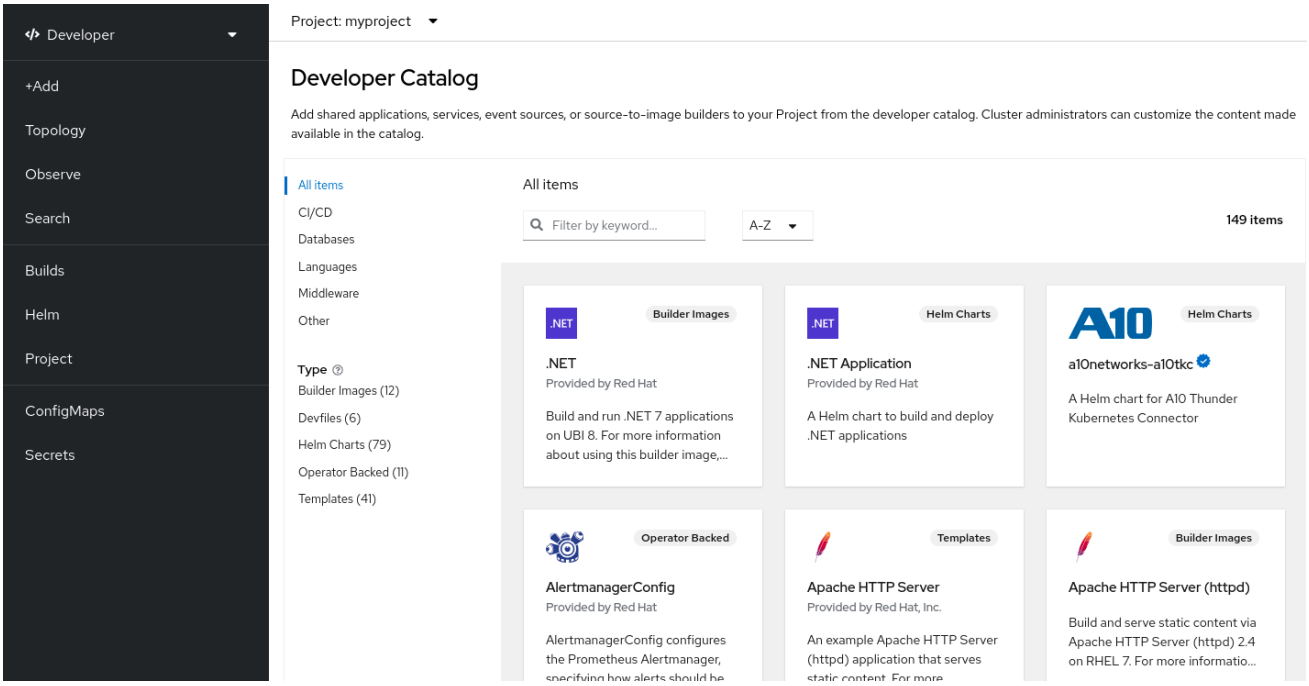
选择用来构建应用程序的基础镜像包含一组软件，这些软件为应用程序提供一个 Linux 系统。在您构建自己的镜像时，您的软件将放置到该文件系统中，可以像对待操作系统一样看待该文件系统。基础镜像的选择会对容器未来的安全性、效率和可升级性产生重大影响。

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 Red Hat OpenShift Service on AWS 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 6.2. 为需要特定运行时的应用选择 S2I 基础镜像



6.2.3. 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。Red Hat OpenShift Service on AWS 中使用的许多组件都存储在 Quay.io 中，包括容器镜像和用于在 AWS 本身上部署 Red Hat OpenShift Service 的 Operator。Quay.io 还提供了存储其他类型内容的方法，包括 Helm Charts。

如果您希望自己的私有容器 registry，Red Hat OpenShift Service on AWS 本身包括一个私有容器

registry，随 Red Hat OpenShift Service on AWS 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。其中一些凭证会根据 AWS 上的 Red Hat OpenShift Service，而其他凭证可以分配给个人。

6.3. 为 RED HAT OPENSHIFT SERVICE ON AWS 创建 KUBERNETES 清单

虽然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 环境中（如 Red Hat OpenShift Service on AWS）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

6.3.1. 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。使用该服务时，按名称请求该服务，Red Hat OpenShift Service on AWS 会将该名称解析为您可访问组成该服务的 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。

6.3.2. 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 Red Hat OpenShift Service on AWS 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。

6.3.3. 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。为了满足这一需求，您可以从 Red Hat OpenShift Service on AWS Web 控制台中提供的以下目录获取所需的组件：

- OperatorHub，可在每个 Red Hat OpenShift Service on AWS 4 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。

6.3.4. 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。

6.3.5. 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。

6.4. 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 7 章 准入插件

准入(Admission)插件用于帮助规范 Red Hat OpenShift Service on AWS 的功能。

7.1. 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。

Red Hat OpenShift Service on AWS 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。

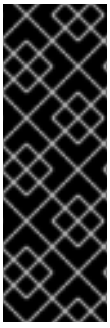
通过一个变异准入插件调用 webhook 服务器可能会对与目标对象相关的资源产生副作用。在这种情况下,，必须执行相应的步骤来验证最终结果是正确的。

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 Red Hat OpenShift Service on AWS 4 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。

7.2. 默认准入插件

Red Hat OpenShift Service on AWS 4 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 7.1. 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 7.2. 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

7.3. WEBHOOK 准入插件

除了 Red Hat OpenShift Service on AWS 默认准入插件外，也可以通过调用 webhook 服务器的 webhook 准入插件来实施动态准入，以扩展准入链的功能。Webhook 服务器通过 HTTP 在定义的端点调用。

Red Hat OpenShift Service on AWS 中有两种 webhook 准入插件：

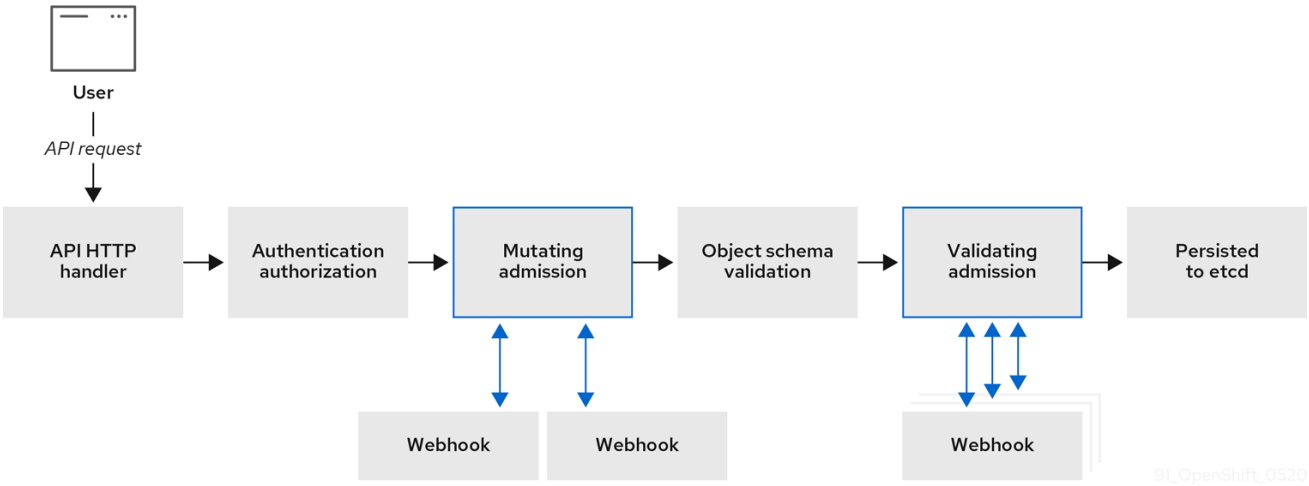
- 在准入过程中，*变异（mutating）准入插件*可以执行任务，如注入关联性标签。
- 在准入过程结束时，*验证（validating）准入插件*会用来确保对象被正确配置。例如，确保关联性标签与预期一样。如果验证通过，Red Hat OpenShift Service on AWS 会按照配置来调度对象。

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。

下图演示了调用多个 webhook 服务器的序列准入链进程。

图 7.1. 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。然后，Red Hat OpenShift Service on AWS 会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。
- Pod 优先级类验证。

注意

Red Hat OpenShift Service on AWS 中的最大默认 webhook 超时值为 13 秒，且无法更改。

7.4. 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。

7.4.1. 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。

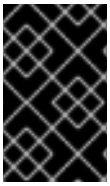
7 用于准入请求的 webhook URL。将 **<webhook_url>** 替换为适当的值。

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 Red Hat OpenShift Service on AWS 4 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。

7.4.2. 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。

7.5. 其他资源

- Pod 优先级名称